My Project

Generated by Doxygen 1.9.5

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 Algebra Namespace Reference

Classes

- class ESolver
- class Matrix
- class MatrixDiag
- class MatrixSkyline

Enumerations

- enum class Solvers {
  BiCGStab , GMRES , GMRES_BiCGStab , Gauss ,
  PARDISO }

5.1.1 Enumeration Type Documentation

5.1.1.1 Solvers

```
enum class Algebra::Solvers  [strong]
```

Enumerator

BiCGStab	
GMRES	
GMRES_BiCGStab	
Gauss	
PARDISO	

5.2 corenc Namespace Reference

Namespaces

- namespace color
- namespace Mesh
- namespace method
- namespace solvers
- namespace tests

Classes

- class CBurgersScalar
- class CDiffusionScalar
- class CFESolution
- class CFEweights
- class CFiniteSolver
- class CProblem
- class CShallowWater
- class CSolution
- class CVecSolution
- struct GaussianKernel
- struct GaussianProcess
- class multi_vector
- class test_case_elliptic_fem
- class test_case_solver
- class test_cases
- class test_conv_diff

Typedefs

- using scalar_func = std::function< const double(const Mesh::Point &)>
- using vector_func = std::function< const Mesh::Point(const Mesh::Point &)>

Enumerations

- enum class Terms {
  IUV , IDUDV , IDUV , IUDV ,
  EUV , EDUDV , EDUV , EUDV ,
  EFV , RUV , SUPG }
- enum class Parameters { DIFFUSION , MASS , ADVECTION }

5.2.1 Detailed Description

Usually it is a vector<double> but some methods required different types like vector of vectors or double/tripple values The interface for dealing with solutions; NOT IN USE

5.2.2 Typedef Documentation

5.2.2.1 scalar_func

```
using corenc::scalar_func = typedef std::function<const double(const Mesh::Point&)>
```

5.2.2.2 vector_func

```
using corenc::vector_func = typedef std::function<const Mesh::Point(const Mesh::Point&)>
```

5.2.3 Enumeration Type Documentation

5.2.3.1 Parameters

```
enum class corenc::Parameters [strong]
```

Enumerator

DIFFUSION	
MASS	
ADVECTION	

5.2.3.2 Terms

```
enum class corenc::Terms [strong]
```

Enumerator

IUV	
IDUDV	
IDUV	
IUDV	
EUV	
EDUDV	
EDUV	
EUDV	
EFV	
RUV	
SUPG	

5.3 corenc::color Namespace Reference

Variables

- const std::string ESCAPE = "\u001b[0m"
- const std::string BLACK = "\u001b[30m"
- const std::string RED = "\u001b[31m"
- const std::string GREEN = "\u001b[32m"
- const std::string YELLOW = "\u001b[33m"
- const std::string BLUE = "\u001b[34m"
- const std::string MAGENTA = "\u001b[35m"
- const std::string CYAN = "\u001b[36m"
- const std::string WHITE = "\u001b[37m"
- const std::string PURPLE = "\e[1;35m"
- const std::string BBLACK = "\u001b[30;1m"
- const std::string BRED = "\u001b[31;1m"
- const std::string BGREEN = "\u001b[32;1m"
- const std::string BYELLOW = "\u001b[33;1m"
- const std::string BBLUE = "\u001b[34;1m"
- const std::string BMAGENTA = "\u001b[35;1m"
- const std::string BCYAN = "\u001b[36;1m"
- const std::string BWHITE = "\u001b[37;1m"

5.3.1 Variable Documentation

5.3.1.1 BBLACK

```
const std::string corenc::color::BBLACK = "\u001b[30;1m"
```

5.3.1.2 BBLUE

```
const std::string corenc::color::BBLUE = "\u001b[34;1m"
```

5.3.1.3 BCYAN

```
const std::string corenc::color::BCYAN = "\u001b[36;1m"
```

5.3.1.4 BGREEN

```
const std::string corenc::color::BGREEN = "\u001b[32;1m"
```

5.3.1.5 BLACK

```
const std::string corenc::color::BLACK = "\u001b[30m"
```

5.3.1.6 BLUE

```
const std::string corenc::color::BLUE = "\u001b[34m"
```

5.3.1.7 BMAGENTA

```
const std::string corenc::color::BMAGENTA = "\u001b[35;1m"
```

5.3.1.8 BRED

```
const std::string corenc::color::BRED = "\u001b[31;1m"
```

5.3.1.9 BWHITE

```
const std::string corenc::color::BWHITE = "\u001b[37;1m"
```

5.3.1.10 BYELLOW

```
const std::string corenc::color::BYELLOW = "\u001b[33;1m"
```

5.3.1.11 CYAN

```
const std::string corenc::color::CYAN = "\u001b[36m"
```

5.3.1.12 ESCAPE

```
const std::string corenc::color::ESCAPE = "\u001b[0m"
```

5.3.1.13 GREEN

```
const std::string corenc::color::GREEN = "\u001b[32m"
```

5.3.1.14 MAGENTA

```
const std::string corenc::color::MAGENTA = "\u001b[35m"
```

5.3.1.15 PURPLE

```
const std::string corenc::color::PURPLE = "\e[1;35m"
```

5.3.1.16 RED

```
const std::string corenc::color::RED = "\u001b[31m"
```

5.3.1.17 WHITE

```
const std::string corenc::color::WHITE = "\u001b[37m"
```

5.3.1.18 YELLOW

```
const std::string corenc::color::YELLOW = "\u001b[33m"
```

5.4 corenc::Mesh Namespace Reference

Classes

- class CCube
- class CCubeBasis
- class CEdge
- class CEdge2ndBasis
- class CEdgeConstantBasis
- class CEdgeHermiteBasis
- class CEdgeLinearBasis
- class CEdgeMultiBasis
- class CElement
- class CElement2D
- class CElement2D< bool >
- class CElement< bool >
- class CFiniteElement
- class CFiniteElement2D
- class CFiniteElement< Shape, ShapeFunction, bool, bool >
- class CFiniteElement< Shape, ShapeFunction, DoF, bool >
- class CMesh
- class CMesh1D
- class CMesh< bool >
- class CNode
- class CNodeBasis
- class CParameter
- class CRectangle
- class CRectangleBasis
- class CRectangleBasis2
- class CRectangleBasis2x
- class CRectangleBasis2y
- class CRectangleConstantBasis
- class CRectangleHBasis
- class CRegularMesh
- class CRegularMesh3D
- class CShape
- class CShapeFunction
- class CTriangle
- class CTriangleBasis
- class CTriangleLagrangeBasis
- class CTriangleLinear
- class CTriangleLinearBasis
- class CTriangularMesh
- class CTriangularMeshLinear
- struct Gauss1dim
- struct Gauss1dimN
- struct GaussRectangular
- struct GaussRectangularCubic
- struct GaussTetrahedron
- struct GaussTriangle
- class parameter
- class Point
- class point_source

Typedefs

- using function_dp = std::function< const double(const Point &)>

Enumerations

- enum Elements {
  Interval = 0 , Triangle = 1 , Rectangle = 2 , Tetrahedron = 3 ,
  Cube = 4 }
- enum class NODES { FIRST , LAST }
- enum Meshes { Mesh1D = 0 , TriangularMesh = 1 , TetrahedralMesh = 2 }

5.4.1 Typedef Documentation

5.4.1.1 function_dp

```
typedef std::function< const double(const Point &)> corenc::Mesh::function_dp
```

5.4.2 Enumeration Type Documentation

5.4.2.1 Elements

```
enum corenc::Mesh::Elements
```

Enumerator

Interval	
Triangle	
Rectangle	
Tetrahedron	
Cube	

5.4.2.2 Meshes

```
enum corenc::Mesh::Meshes
```

Enumerator

Mesh1D	
TriangularMesh	
TetrahedralMesh	

5.4.2.3 NODES

`enum class corenc::Mesh::NODES [strong]`

Enumerator

FIRST	
LAST	

5.5 corenc::method Namespace Reference

Classes

- class CDGMethod
- class CDGMethodZero
- class CFEMethod
- class CFEMethodZero
- class DGMethod
- class DGMethodZero
- class DGSolution
- class FEAnalysis
- class FEMethod
- class FEMethodZero
- class FVMethod1d
- class RungeKutta
- class STSolution
- class system_dg_method
- class system_dg_method< Grid, bool, bool >

Enumerations

- enum class DGFlux {
  EIP , EBaumannOden , EBaumannOdenIP , ENIPG ,
  EUpwind , ECentral , ELaxFriedrichs , IIP ,
  IBaumannOden , IBaumannOdenIP , INIPG , IUpwind ,
  ICentral , ILaxFriedrichs , CUSTOM , NOFLUX }
- enum class BoundaryType { MAIN , SECOND , THIRD , FREE }
- enum class FVFlux { LaxFriedrichs , Upwind , Central , NOFLUX }

5.5.1 Enumeration Type Documentation

5.5.1.1 BoundaryType

`enum class corenc::method::BoundaryType [strong]`

Enumerator

MAIN	
SECOND	
THIRD	
FREE	

5.5.1.2 DGFlux

```
enum class corenc::method::DGFlux [strong]
```

Enumerator

EIP	
EBaumannOden	
EBaumannOdenIP	
ENIPG	
EUpwind	
ECentral	
ELaxFriedrichs	
IIP	
IBaumannOden	
IBaumannOdenIP	
INIPG	
IUpwind	
ICentral	
ILaxFriedrichs	
CUSTOM	
NOFLUX	

5.5.1.3 FVFlux

```
enum class corenc::method::FVFlux [strong]
```

Enumerator

LaxFriedrichs	
Upwind	
Central	
NOFLUX	

5.6 corenc::solvers Namespace Reference

Classes

- class dg_shallow_water
- class dg_solver
- class dg_solver_shallow_water
- class eigen_solver
- class fem_solver
- class fem_solver_lib
- struct vector_solution

5.7 corenc::tests Namespace Reference

Classes

- class test_case_rectanglebasis
- class test_case_regular_mesh
- class test_case_trianglebasis

5.8 Methods Namespace Reference

Classes

- class CSMethod

5.9 wtf Namespace Reference

Functions

- const Point mid_point (const Point &p1, const Point &p2)
- const Point s_point (const Point &p1, const Point &p2, const double s)
- const Point center_point (const Point &p1, const Point &p2, const Point &p3)

5.9.1 Function Documentation

5.9.1.1 center_point()

```
const Point wtf::center_point (
            const Point & p1,
            const Point & p2,
            const Point & p3 )
```

5.9.1.2 mid_point()

```
const Point wtf::mid_point (
            const Point & p1,
            const Point & p2 )
```

5.9.1.3 s_point()

```
const Point wtf::s_point (
            const Point & p1,
            const Point & p2,
            const double s )
```

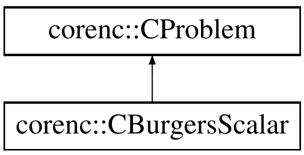
Chapter 6

Class Documentation

6.1 corenc::CBurgersScalar Class Reference

```
#include <BurgersScalar.h>
```

Inheritance diagram for corenc::CBurgersScalar:



Public Member Functions

- CBurgersScalar ()
- ∼CBurgersScalar ()
- Terms getTerm (const unsigned int) const
- const unsigned int getNumberOfTerms () const
- const int setTerm (const unsigned int, const Terms &)
- const int addTerm (const Terms &)
- const double getFlux (const double) const
- const int removeTerm (const Terms &)
- const int load_parameters (const std::string &file_name)

6.1.1 Constructor & Destructor Documentation

6.1.1.1 CBurgersScalar()

```
CBurgersScalar::CBurgersScalar ( )
```

6.1.1.2 ∼CBurgersScalar()

```
CBurgersScalar::~CBurgersScalar ( )
```

6.1.2 Member Function Documentation

6.1.2.1 addTerm()

```
const int CBurgersScalar::addTerm (
            const Terms & term ) [virtual]
```

Implements corenc::CProblem.

6.1.2.2 getFlux()

```
const double CBurgersScalar::getFlux (
            const double p ) const
```

6.1.2.3 getNumberOfTerms()

```
const unsigned int CBurgersScalar::getNumberOfTerms ( ) const [virtual]
```

Implements corenc::CProblem.

6.1.2.4 getTerm()

```
Terms CBurgersScalar::getTerm (
            const unsigned int i ) const [virtual]
```

Implements corenc::CProblem.

6.1.2.5 load_parameters()

```
const int CBurgersScalar::load_parameters (
            const std::string & file_name ) [virtual]
```

Implements corenc::CProblem.

6.1.2.6 removeTerm()

```
const int CBurgersScalar::removeTerm (
            const Terms & term )
```

6.1.2.7 setTerm()

```
const int CBurgersScalar::setTerm (
            const unsigned int i,
            const Terms & term )  [virtual]
```

Implements corenc::CProblem.

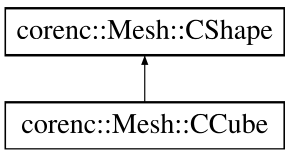
The documentation for this class was generated from the following files:

- Problems/BurgersScalar.h
- Problems/BurgersScalar.cpp

6.2 corenc::Mesh::CCube Class Reference

```
#include <Cube.h>
```

Inheritance diagram for corenc::Mesh::CCube:



Public Member Functions

- CCube ()
- CCube (const int n1, const int n2, const int n3, const int n4, const int order)
- CCube (const int n1, const int n2, const int n3, const int n4, const int e1, const int e2, const int e3, const int e4, const int order)
- CCube (const int ∗, const int order)
- CCube (const int ∗, const int ∗, const int order)
- CCube (const CCube &)
- CCube & operator= (const CCube &t)
- const bool operator== (const CCube &t)
- std::istream & operator>> (std::istream &is)
- ∼CCube ()
- const int GetNode (const int) const
- const int GetNode (const NODES &) const
- const int GetEdge (const int) const
- const int GetFacet (const int) const
- const int GetNumberOfNodes () const

- const int GetNumberOfEdges () const
- const int GetNumberOfFacets () const
- const double Integrate (const std::function< const double(const Point &)> &, const std::vector< Point > &v) const
- const Point Integrate (const std::function< const Point(const Point &)> &, const std::vector< Point > &v) const
- const std::vector< double > Integrate (const std::function< const std::vector< double >(const Point &)> &, const std::vector< Point > &) const
- void SetNode (const int k, const int node)
- const int IncreaseOrder ()
- const int SetOrder (const int px, const int py)
- void SetEdge (const int k, const int edge)
- void SetFacet (const int k, const int facet)

6.2.1 Constructor & Destructor Documentation

6.2.1.1 CCube() [1/6]

```
CCube::CCube ( )
```

6.2.1.2 CCube() [2/6]

```
CCube::CCube (
        const int n1,
        const int n2,
        const int n3,
        const int n4,
        const int order )
```

6.2.1.3 CCube() [3/6]

```
CCube::CCube (
        const int n1,
        const int n2,
        const int n3,
        const int n4,
        const int e1,
        const int e2,
        const int e3,
        const int e4,
        const int order )
```

6.2.1.4 CCube() [4/6]

```
CCube::CCube (
            const int * nodes,
            const int order )
```

6.2.1.5 CCube() [5/6]

```
CCube::CCube (
            const int * nodes,
            const int * edges,
            const int order )
```

6.2.1.6 CCube() [6/6]

```
CCube::CCube (
            const CCube & t )
```

6.2.1.7 ∼CCube()

```
corenc::Mesh::CCube::∼CCube ( )  [inline]
```

6.2.2 Member Function Documentation

6.2.2.1 GetEdge()

```
const int CCube::GetEdge (
            const int n ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.2.2.2 GetFacet()

```
const int CCube::GetFacet (
            const int  ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.2.2.3 GetNode() [1/2]

```
const int CCube::GetNode (
            const int n ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.2.2.4 GetNode() [2/2]

```
const int CCube::GetNode (
            const NODES & node ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.2.2.5 GetNumberOfEdges()

```
const int CCube::GetNumberOfEdges ( ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.2.2.6 GetNumberOfFacets()

```
const int CCube::GetNumberOfFacets ( ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.2.2.7 GetNumberOfNodes()

```
const int CCube::GetNumberOfNodes ( ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.2.2.8 IncreaseOrder()

```
const int CCube::IncreaseOrder ( )
```

6.2.2.9 Integrate() [1/3]

```
const double CCube::Integrate (
            const std::function< const double(const Point &)> & f,
            const std::vector< Point > & v ) const
```

6.2.2.10 Integrate() [2/3]

```
const Point CCube::Integrate (
            const std::function< const Point(const Point &)> & f,
            const std::vector< Point > & v ) const
```

6.2.2.11 Integrate() [3/3]

```
const vector< double > CCube::Integrate (
            const std::function< const std::vector< double >(const Point &)> & f,
            const std::vector< Point > & v ) const [virtual]
```

Implements corenc::Mesh::CShape.

6.2.2.12 operator=()

```
CCube & corenc::Mesh::CCube::operator= (
            const CCube & t ) [inline]
```

6.2.2.13 operator==()

```
const bool corenc::Mesh::CCube::operator== (
            const CCube & t ) [inline]
```

6.2.2.14 operator>>()

```
std::istream & corenc::Mesh::CCube::operator>> (
            std::istream & is ) [inline]
```

6.2.2.15 SetEdge()

```
void CCube::SetEdge (
             const int k,
             const int edge ) [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.2.2.16 SetFacet()

```
void CCube::SetFacet (
             const int k,
             const int facet ) [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.2.2.17 SetNode()

```
void CCube::SetNode (
             const int k,
             const int node ) [virtual]
```

Implements corenc::Mesh::CShape.

6.2.2.18 SetOrder()

```
const int CCube::SetOrder (
             const int px,
             const int py )
```

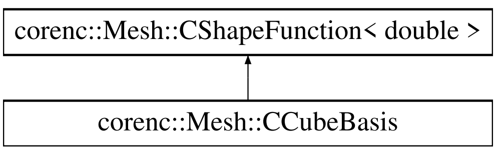
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/Cube.h
- CoreNCFEM/FiniteElements/Cube.cpp

6.3 corenc::Mesh::CCubeBasis Class Reference

```
#include <Cube.h>
```

Inheritance diagram for corenc::Mesh::CCubeBasis:



Public Member Functions

- CCubeBasis ()
- CCubeBasis (const Point &, const Point &, const Point &, const Point &, const int order)
- CCubeBasis (const Point ∗, const int order)
- CCubeBasis (const CCubeBasis &)
- CCubeBasis & operator= (const CCubeBasis &t)
- ∼CCubeBasis ()
- const int GetNumberOfShapeFunctions () const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetValue (const Point &) const
- const int IncreaseOrder ()
- const double GetMeasure () const
- const double GetWeight (const int, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const

6.3.1 Constructor & Destructor Documentation

6.3.1.1 CCubeBasis() [1/4]

```
CCubeBasis::CCubeBasis ( )
```

6.3.1.2 CCubeBasis() [2/4]

```
CCubeBasis::CCubeBasis (
            const Point & p1,
            const Point & p2,
            const Point & p3,
            const Point & p4,
            const int order )
```

6.3.1.3 CCubeBasis() [3/4]

```
CCubeBasis::CCubeBasis (
            const Point * p,
            const int order )
```

6.3.1.4 CCubeBasis() [4/4]

```
CCubeBasis::CCubeBasis (
            const CCubeBasis & t )
```

6.3.1.5 ∼CCubeBasis()

```
corenc::Mesh::CCubeBasis::∼CCubeBasis ( )  [inline]
```

6.3.2 Member Function Documentation

6.3.2.1 GetGradShapeFunction()

```
const Point CCubeBasis::GetGradShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.3.2.2 GetMeasure()

```
const double corenc::Mesh::CCubeBasis::GetMeasure ( ) const  [inline], [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.3.2.3 GetNormal()

```
const Point CCubeBasis::GetNormal ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.3.2.4 GetNumberOfShapeFunctions()

```
const int CCubeBasis::GetNumberOfShapeFunctions ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.3.2.5 GetShapeFunction()

```
const double CCubeBasis::GetShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.3.2.6 GetValue()

```
const double CCubeBasis::GetValue (
            const Point & p ) const
```

6.3.2.7 GetWeight()

```
const double CCubeBasis::GetWeight (
            const int node,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const
```

6.3.2.8 IncreaseOrder()

```
const int CCubeBasis::IncreaseOrder ( )
```

6.3.2.9 operator=()

```
CCubeBasis & corenc::Mesh::CCubeBasis::operator= (
            const CCubeBasis & t )  [inline]
```

6.3.2.10 ReverseNormal()

```
void CCubeBasis::ReverseNormal ( )  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

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

- CoreNCFEM/FiniteElements/Cube.h
- CoreNCFEM/FiniteElements/Cube.cpp

6.4 corenc::method::CDGMethod< Type > Class Template Reference

```
#include <DGMethod.h>
```

Public Member Functions

- CDGMethod ()
- virtual ∼CDGMethod ()
- virtual const int Assemble ()=0
- virtual const Type GetSolution (const std::vector< double > &point) const =0
- virtual const std::vector< Type > GetSolution () const =0
- virtual const Type GetMaxSolution () const =0
- virtual const Type GetMinSolution () const =0

6.4.1 Constructor & Destructor Documentation

6.4.1.1 CDGMethod()

```
template<class Type >
corenc::method::CDGMethod< Type >::CDGMethod ( ) [inline]
```

6.4.1.2 ∼CDGMethod()

```
template<class Type >
virtual corenc::method::CDGMethod< Type >::∼CDGMethod ( ) [inline], [virtual]
```

6.4.2 Member Function Documentation

6.4.2.1 Assemble()

```
template<class Type >
virtual const int corenc::method::CDGMethod< Type >::Assemble ( ) [pure virtual]
```

6.4.2.2 GetMaxSolution()

```
template<class Type >
virtual const Type corenc::method::CDGMethod< Type >::GetMaxSolution ( ) const [pure virtual]
```

6.4.2.3 GetMinSolution()

```
template<class Type >
virtual const Type corenc::method::CDGMethod< Type >::GetMinSolution ( ) const [pure virtual]
```

6.4.2.4 GetSolution() [1/2]

```
template<class Type >
virtual const std::vector< Type > corenc::method::CDGMethod< Type >::GetSolution ( ) const
[pure virtual]
```

6.4.2.5 GetSolution() [2/2]

```
template<class Type >
virtual const Type corenc::method::CDGMethod< Type >::GetSolution (
            const std::vector< double > & point ) const [pure virtual]
```

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

- CoreNCFEM/Methods/DGMethod.h

6.5 corenc::method::CDGMethodZero< Type > Class Template Reference

```
#include <DGMethodZero.h>
```

Public Member Functions

- CDGMethodZero ()
- virtual ~CDGMethodZero ()
- virtual const int Assemble ()=0
- virtual const Type GetSolution (const std::vector< double > &point) const =0
- virtual const std::vector< Type > GetSolution () const =0
- virtual const Type GetMaxSolution () const =0
- virtual const Type GetMinSolution () const =0

6.5.1 Constructor & Destructor Documentation

6.5.1.1 CDGMethodZero()

```
template<class Type >
corenc::method::CDGMethodZero< Type >::CDGMethodZero ( )  [inline]
```

6.5.1.2 ∼CDGMethodZero()

```
template<class Type >
virtual corenc::method::CDGMethodZero< Type >::~CDGMethodZero ( )  [inline], [virtual]
```

6.5.2 Member Function Documentation

6.5.2.1 Assemble()

```
template<class Type >
virtual const int corenc::method::CDGMethodZero< Type >::Assemble ( )  [pure virtual]
```

6.5.2.2 GetMaxSolution()

```
template<class Type >
virtual const Type corenc::method::CDGMethodZero< Type >::GetMaxSolution ( ) const  [pure
virtual]
```

6.5.2.3 GetMinSolution()

```
template<class Type >
virtual const Type corenc::method::CDGMethodZero< Type >::GetMinSolution ( ) const  [pure
virtual]
```

6.5.2.4 GetSolution() [1/2]

```
template<class Type >
virtual const std::vector< Type > corenc::method::CDGMethodZero< Type >::GetSolution ( )
const  [pure virtual]
```

6.5.2.5 GetSolution() [2/2]

```
template<class Type >
virtual const Type corenc::method::CDGMethodZero< Type >::GetSolution (
            const std::vector< double > & point ) const  [pure virtual]
```

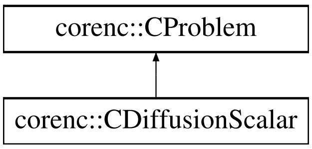
The documentation for this class was generated from the following file:

- CoreNCFEM/Methods/DGMethodZero.h

6.6 corenc::CDiffusionScalar Class Reference

```
#include <DiffusionScalar.h>
```

Inheritance diagram for corenc::CDiffusionScalar:



Public Member Functions

- CDiffusionScalar ()
- ∼CDiffusionScalar ()
- Terms getTerm (const unsigned int) const
- const unsigned int getNumberOfTerms () const
- const int findTerm (const Terms &) const
- const int setTerm (const unsigned int, const Terms &)
- const int addTerm (const Terms &)
- const int removeTerm (const Terms &)
- const int load_parameters (const std::string &file_name)
- const double get_parameter (const Terms &, const int element_type, const Mesh::Point &) const
- const double get_parameter (const Terms &, const int element_number, const int element_type, const Mesh::Point &) const
- const Mesh::Point get_parameter (const Terms &, const int element_number, const int element_type, const Mesh::Point &, const int) const
- const double get_parameter (const Terms &, const int element_type, const int element_number, const int node, const Mesh::Point &) const
- const Mesh::Point get_parameter (const Terms &, const int element_type, const int element_number, const int node, const Mesh::Point &, const int v) const
- const double get_boundary_parameter (const int type, const int element_type, const Mesh::Point &) const
- const double get_boundary_parameter (const int type, const int element_type, const int element_number, const Mesh::Point &) const
- const double get_boundary_parameter (const int type, const int element_type, const int element_number, const int node, const Mesh::Point &) const
- const int get_number_of_boundaries () const
- const int get_boundary_type (const int number) const
- const int add_parameter (const Terms &, const int element_type, const double &value)
- const int add_parameter (const Terms &, const int element_type, const Mesh::parameter< double > &value)

- const int add_parameter (const Terms &, const int element_type, const Mesh::parameter< Mesh::Point > &value)
- const int set_parameter (const Terms &, const int element_type, const Mesh::parameter< double > &value)
- const int set_parameter (const Terms &, const int element_type, const Mesh::parameter< Mesh::Point > &value)
- const int set_boundary_parameter (const int type, const int element_type, const boundary &value)
- const int add_boundary_parameter (const int type, const int element_type, const Mesh::parameter< double > &value)
- const int add_boundary_parameter (const int element_type, const Mesh::parameter< double > &value, const Mesh::parameter< double > &value2)
- const Mesh::point_source< double > get_point_source (const int number) const
- void set_point_source (const int number, const Mesh::point_source< double > &)
- const int get_total_sources () const

6.6.1 Constructor & Destructor Documentation

6.6.1.1 CDiffusionScalar()

```
CDiffusionScalar::CDiffusionScalar ( )
```

6.6.1.2 ∼CDiffusionScalar()

```
CDiffusionScalar::∼CDiffusionScalar ( )
```

6.6.2 Member Function Documentation

6.6.2.1 add_boundary_parameter() [1/2]

```
const int CDiffusionScalar::add_boundary_parameter (
            const int element_type,
            const Mesh::parameter< double > & value,
            const Mesh::parameter< double > & value2 )
```

6.6.2.2 add_boundary_parameter() [2/2]

```
const int CDiffusionScalar::add_boundary_parameter (
            const int type,
            const int element_type,
            const Mesh::parameter< double > & value )
```

6.6.2.3 add_parameter() [1/3]

```
const int CDiffusionScalar::add_parameter (
            const Terms & term,
            const int element_type,
            const double & value )
```

6.6.2.4 add_parameter() [2/3]

```
const int CDiffusionScalar::add_parameter (
            const Terms & term,
            const int element_type,
            const Mesh::parameter< double > & value )
```

6.6.2.5 add_parameter() [3/3]

```
const int CDiffusionScalar::add_parameter (
            const Terms & term,
            const int element_type,
            const Mesh::parameter< Mesh::Point > & value )
```

6.6.2.6 addTerm()

```
const int CDiffusionScalar::addTerm (
            const Terms & term )  [virtual]
```

Implements corenc::CProblem.

6.6.2.7 findTerm()

```
const int CDiffusionScalar::findTerm (
            const Terms & term ) const
```

6.6.2.8 get_boundary_parameter() [1/3]

```
const double CDiffusionScalar::get_boundary_parameter (
            const int type,
            const int element_type,
            const int element_number,
            const int node,
            const Mesh::Point & p ) const
```

6.6.2.9 get_boundary_parameter() [2/3]

```
const double CDiffusionScalar::get_boundary_parameter (
            const int type,
            const int element_type,
            const int element_number,
            const Mesh::Point & p ) const
```

6.6.2.10 get_boundary_parameter() [3/3]

```
const double CDiffusionScalar::get_boundary_parameter (
            const int type,
            const int element_type,
            const Mesh::Point & p ) const
```

6.6.2.11 get_boundary_type()

```
const int CDiffusionScalar::get_boundary_type (
            const int number ) const
```

6.6.2.12 get_number_of_boundaries()

```
const int CDiffusionScalar::get_number_of_boundaries ( ) const
```

6.6.2.13 get_parameter() [1/5]

```
const double CDiffusionScalar::get_parameter (
            const Terms & term,
            const int element_number,
            const int element_type,
            const Mesh::Point & p ) const
```

6.6.2.14 get_parameter() [2/5]

```
const Mesh::Point CDiffusionScalar::get_parameter (
            const Terms & term,
            const int element_number,
            const int element_type,
            const Mesh::Point & p,
            const int ) const
```

6.6.2.15 get_parameter() [3/5]

```
const double CDiffusionScalar::get_parameter (
            const Terms & term,
            const int element_type,
            const int element_number,
            const int node,
            const Mesh::Point & p ) const
```

6.6.2.16 get_parameter() [4/5]

```
const Mesh::Point CDiffusionScalar::get_parameter (
            const Terms & term,
            const int element_type,
            const int element_number,
            const int node,
            const Mesh::Point & p,
            const int v ) const
```

6.6.2.17 get_parameter() [5/5]

```
const double CDiffusionScalar::get_parameter (
            const Terms & term,
            const int element_type,
            const Mesh::Point & p ) const
```

6.6.2.18 get_point_source()

```
const Mesh::point_source< double > CDiffusionScalar::get_point_source (
            const int number ) const
```

6.6.2.19 get_total_sources()

```
const int CDiffusionScalar::get_total_sources ( ) const
```

6.6.2.20 getNumberOfTerms()

```
const unsigned int CDiffusionScalar::getNumberOfTerms ( ) const [virtual]
```

Implements corenc::CProblem.

6.6.2.21 getTerm()

```
Terms CDiffusionScalar::getTerm (
            const unsigned int i ) const  [virtual]
```

Implements corenc::CProblem.

6.6.2.22 load_parameters()

```
const int CDiffusionScalar::load_parameters (
            const std::string & file_name )  [virtual]
```

Implements corenc::CProblem.

6.6.2.23 removeTerm()

```
const int CDiffusionScalar::removeTerm (
            const Terms & term )
```

6.6.2.24 set_boundary_parameter()

```
const int CDiffusionScalar::set_boundary_parameter (
            const int type,
            const int element_type,
            const boundary & value )
```

6.6.2.25 set_parameter() [1/2]

```
const int CDiffusionScalar::set_parameter (
            const Terms & term,
            const int element_type,
            const Mesh::parameter< double > & value )
```

6.6.2.26 set_parameter() [2/2]

```
const int CDiffusionScalar::set_parameter (
            const Terms & term,
            const int element_type,
            const Mesh::parameter< Mesh::Point > & value )
```

6.6.2.27 set_point_source()

```
void CDiffusionScalar::set_point_source (
            const int number,
            const Mesh::point_source< double > & srs )
```

6.6.2.28 setTerm()

```
const int CDiffusionScalar::setTerm (
            const unsigned int i,
            const Terms & term ) [virtual]
```

Implements corenc::CProblem.

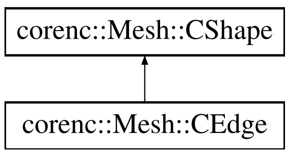
The documentation for this class was generated from the following files:

- Problems/DiffusionScalar.h
- Problems/DiffusionScalar.cpp

6.7 corenc::Mesh::CEdge Class Reference

```
#include <Edge.h>
```

Inheritance diagram for corenc::Mesh::CEdge:



Public Member Functions

- CEdge ()
- CEdge (const CEdge &)
- CEdge (const int n1, const int n2)
- CEdge (const int ∗)
- CEdge & operator= (const CEdge &e)
- ∼CEdge ()
- const int GetNode (const int) const
- const int GetNode (const NODES &) const
- const int GetNumberOfNodes () const
- void SetNode (const int k, const int node)
- const double Integrate (const std::function< const double(const Point &)> &, const std::vector< Point > &v) const
- const Point Integrate (const std::function< const Point(const Point &)> &, const std::vector< Point > &v) const
- const int IncreaseOrder ()
- const std::vector< double > Integrate (const std::function< const std::vector< double >(const Point &)> &, const std::vector< Point > &) const

Friends

- const bool operator== (const CEdge &e1, const CEdge &e2)
- std::istream & operator>> (std::istream &is, CEdge &e)

6.7.1 Constructor & Destructor Documentation

6.7.1.1 CEdge() [1/4]

```
CEdge::CEdge ( )
```

6.7.1.2 CEdge() [2/4]

```
CEdge::CEdge (
            const CEdge & e )
```

6.7.1.3 CEdge() [3/4]

```
CEdge::CEdge (
            const int n1,
            const int n2 )
```

6.7.1.4 CEdge() [4/4]

```
CEdge::CEdge (
            const int * n )
```

6.7.1.5 ∼CEdge()

```
corenc::Mesh::CEdge::∼CEdge ( )  [inline]
```

6.7.2 Member Function Documentation

6.7.2.1 GetNode() [1/2]

```
const int CEdge::GetNode (
            const int k ) const [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.7.2.2 GetNode() [2/2]

```
const int CEdge::GetNode (
            const NODES & node ) const [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.7.2.3 GetNumberOfNodes()

```
const int CEdge::GetNumberOfNodes ( ) const [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.7.2.4 IncreaseOrder()

```
const int CEdge::IncreaseOrder ( )
```

6.7.2.5 Integrate() [1/3]

```
const double CEdge::Integrate (
            const std::function< const double(const Point &)> & f,
            const std::vector< Point > & v ) const
```

6.7.2.6 Integrate() [2/3]

```
const Point CEdge::Integrate (
            const std::function< const Point(const Point &)> & f,
            const std::vector< Point > & v ) const
```

6.7.2.7 Integrate() [3/3]

```
const std::vector< double > corenc::Mesh::CEdge::Integrate (
            const std::function< const std::vector< double >(const Point &)> & ,
            const std::vector< Point > & ) const [virtual]
```

Implements corenc::Mesh::CShape.

6.7.2.8 operator=()

```
CEdge & corenc::Mesh::CEdge::operator= (
            const CEdge & e ) [inline]
```

6.7.2.9 SetNode()

```
void CEdge::SetNode (
            const int k,
            const int node ) [virtual]
```

Implements corenc::Mesh::CShape.

6.7.3 Friends And Related Function Documentation

6.7.3.1 operator==

```
const bool operator== (
            const CEdge & e1,
            const CEdge & e2 ) [friend]
```

6.7.3.2 operator>>

```
std::istream & operator>> (
            std::istream & is,
            CEdge & e ) [friend]
```

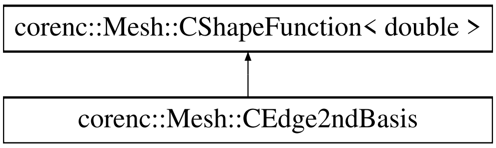
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/Edge.h
- CoreNCFEM/FiniteElements/Edge.cpp

6.8 corenc::Mesh::CEdge2ndBasis Class Reference

```
#include <Edge.h>
```

Inheritance diagram for corenc::Mesh::CEdge2ndBasis:



Public Member Functions

- CEdge2ndBasis ()
- CEdge2ndBasis (const Point &, const Point &)
- CEdge2ndBasis (const Point ∗)
- CEdge2ndBasis (const CEdge2ndBasis &)
- CEdge2ndBasis & operator= (const CEdge2ndBasis &e)
- ∼CEdge2ndBasis ()
- const int GetNumberOfShapeFunctions () const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetWeight (const int node, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const
- const int IncreaseOrder ()
- const double GetMeasure () const

6.8.1 Constructor & Destructor Documentation

6.8.1.1 CEdge2ndBasis() [1/4]

```
CEdge2ndBasis::CEdge2ndBasis ( )
```

6.8.1.2 CEdge2ndBasis() [2/4]

```
CEdge2ndBasis::CEdge2ndBasis (
              const Point & p0,
              const Point & p1 )
```

6.8.1.3 CEdge2ndBasis() [3/4]

```
CEdge2ndBasis::CEdge2ndBasis (
            const Point * p )
```

6.8.1.4 CEdge2ndBasis() [4/4]

```
CEdge2ndBasis::CEdge2ndBasis (
            const CEdge2ndBasis & e )
```

6.8.1.5 ∼CEdge2ndBasis()

```
corenc::Mesh::CEdge2ndBasis::∼CEdge2ndBasis ( ) [inline]
```

6.8.2 Member Function Documentation

6.8.2.1 GetGradShapeFunction()

```
const Point CEdge2ndBasis::GetGradShapeFunction (
            const int k,
            const Point & p ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.8.2.2 GetMeasure()

```
const double corenc::Mesh::CEdge2ndBasis::GetMeasure ( ) const [inline], [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.8.2.3 GetNormal()

```
const Point CEdge2ndBasis::GetNormal ( ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.8.2.4 GetNumberOfShapeFunctions()

```
const int CEdge2ndBasis::GetNumberOfShapeFunctions ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.8.2.5 GetShapeFunction()

```
const double CEdge2ndBasis::GetShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.8.2.6 GetWeight()

```
const double corenc::Mesh::CEdge2ndBasis::GetWeight (
            const int node,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const  [inline]
```

6.8.2.7 IncreaseOrder()

```
const int corenc::Mesh::CEdge2ndBasis::IncreaseOrder ( )  [inline]
```

6.8.2.8 operator=()

```
CEdge2ndBasis & corenc::Mesh::CEdge2ndBasis::operator= (
            const CEdge2ndBasis & e )  [inline]
```

6.8.2.9 ReverseNormal()

```
void CEdge2ndBasis::ReverseNormal ( )  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

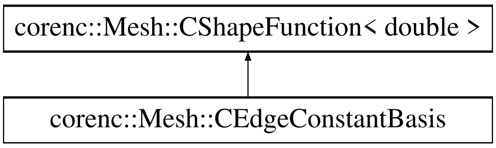
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/Edge.h
- CoreNCFEM/FiniteElements/Edge.cpp

6.9 corenc::Mesh::CEdgeConstantBasis Class Reference

```
#include <Edge.h>
```

Inheritance diagram for corenc::Mesh::CEdgeConstantBasis:



Public Member Functions

- CEdgeConstantBasis ()
- CEdgeConstantBasis (const Point &, const Point &)
- CEdgeConstantBasis (const Point ∗)
- CEdgeConstantBasis (const CEdgeConstantBasis &)
- CEdgeConstantBasis & operator= (const CEdgeConstantBasis &e)
- ∼CEdgeConstantBasis ()
- const int GetNumberOfShapeFunctions () const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetWeight (const int node, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const
- const int IncreaseOrder ()
- const double GetMeasure () const

6.9.1 Constructor & Destructor Documentation

6.9.1.1 CEdgeConstantBasis() [1/4]

```
CEdgeConstantBasis::CEdgeConstantBasis ( )
```

6.9.1.2 CEdgeConstantBasis() [2/4]

```
CEdgeConstantBasis::CEdgeConstantBasis (
              const Point & p0,
              const Point & p1 )
```

6.9.1.3 CEdgeConstantBasis() [3/4]

```
CEdgeConstantBasis::CEdgeConstantBasis (
            const Point * p )
```

6.9.1.4 CEdgeConstantBasis() [4/4]

```
CEdgeConstantBasis::CEdgeConstantBasis (
            const CEdgeConstantBasis & e )
```

6.9.1.5 ∼CEdgeConstantBasis()

```
corenc::Mesh::CEdgeConstantBasis::∼CEdgeConstantBasis ( ) [inline]
```

6.9.2 Member Function Documentation

6.9.2.1 GetGradShapeFunction()

```
const Point CEdgeConstantBasis::GetGradShapeFunction (
            const int ,
            const Point & ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.9.2.2 GetMeasure()

```
const double corenc::Mesh::CEdgeConstantBasis::GetMeasure ( ) const [inline], [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.9.2.3 GetNormal()

```
const Point CEdgeConstantBasis::GetNormal ( ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.9.2.4 GetNumberOfShapeFunctions()

```
const int CEdgeConstantBasis::GetNumberOfShapeFunctions ( ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.9.2.5 GetShapeFunction()

```
const double CEdgeConstantBasis::GetShapeFunction (
            const int ,
            const Point & p ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.9.2.6 GetWeight()

```
const double corenc::Mesh::CEdgeConstantBasis::GetWeight (
            const int node,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const [inline]
```

6.9.2.7 IncreaseOrder()

```
const int corenc::Mesh::CEdgeConstantBasis::IncreaseOrder ( ) [inline]
```

6.9.2.8 operator=()

```
CEdgeConstantBasis & corenc::Mesh::CEdgeConstantBasis::operator= (
            const CEdgeConstantBasis & e ) [inline]
```

6.9.2.9 ReverseNormal()

```
void CEdgeConstantBasis::ReverseNormal ( ) [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

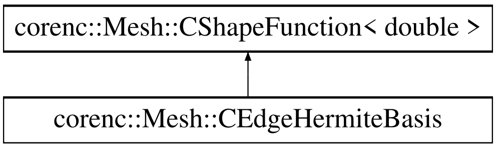
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/Edge.h
- CoreNCFEM/FiniteElements/Edge.cpp

6.10 corenc::Mesh::CEdgeHermiteBasis Class Reference

```
#include <Edge.h>
```

Inheritance diagram for corenc::Mesh::CEdgeHermiteBasis:



Public Member Functions

- CEdgeHermiteBasis ()
- CEdgeHermiteBasis (const Point &, const Point &)
- CEdgeHermiteBasis (const Point ∗)
- CEdgeHermiteBasis (const CEdgeHermiteBasis &)
- CEdgeHermiteBasis & operator= (const CEdgeHermiteBasis &e)
- ∼CEdgeHermiteBasis ()
- const int GetNumberOfShapeFunctions () const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const int IncreaseOrder ()
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetWeight (const int node, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const
- const double GetMeasure () const

6.10.1 Constructor & Destructor Documentation

6.10.1.1 CEdgeHermiteBasis() [1/4]

```
CEdgeHermiteBasis::CEdgeHermiteBasis ( )
```

6.10.1.2 CEdgeHermiteBasis() [2/4]

```
CEdgeHermiteBasis::CEdgeHermiteBasis (
            const Point & p0,
            const Point & p1 )
```

6.10.1.3 CEdgeHermiteBasis() [3/4]

```
CEdgeHermiteBasis::CEdgeHermiteBasis (
            const Point * p )
```

6.10.1.4 CEdgeHermiteBasis() [4/4]

```
CEdgeHermiteBasis::CEdgeHermiteBasis (
            const CEdgeHermiteBasis & e )
```

6.10.1.5 ∼CEdgeHermiteBasis()

```
corenc::Mesh::CEdgeHermiteBasis::∼CEdgeHermiteBasis ( ) [inline]
```

6.10.2 Member Function Documentation

6.10.2.1 GetGradShapeFunction()

```
const Point CEdgeHermiteBasis::GetGradShapeFunction (
            const int k,
            const Point & p ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.10.2.2 GetMeasure()

```
const double corenc::Mesh::CEdgeHermiteBasis::GetMeasure ( ) const [inline], [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.10.2.3 GetNormal()

```
const Point CEdgeHermiteBasis::GetNormal ( ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.10.2.4 GetNumberOfShapeFunctions()

```
const int CEdgeHermiteBasis::GetNumberOfShapeFunctions ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.10.2.5 GetShapeFunction()

```
const double CEdgeHermiteBasis::GetShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.10.2.6 GetWeight()

```
const double corenc::Mesh::CEdgeHermiteBasis::GetWeight (
            const int node,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const  [inline]
```

6.10.2.7 IncreaseOrder()

```
const int corenc::Mesh::CEdgeHermiteBasis::IncreaseOrder ( )  [inline]
```

6.10.2.8 operator=()

```
CEdgeHermiteBasis & corenc::Mesh::CEdgeHermiteBasis::operator= (
            const CEdgeHermiteBasis & e )  [inline]
```

6.10.2.9 ReverseNormal()

```
void CEdgeHermiteBasis::ReverseNormal ( )  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

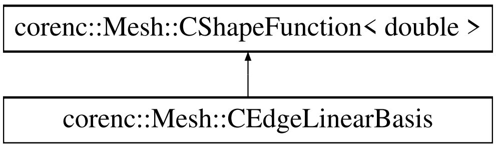
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/Edge.h
- CoreNCFEM/FiniteElements/Edge.cpp

6.11 corenc::Mesh::CEdgeLinearBasis Class Reference

```
#include <Edge.h>
```

Inheritance diagram for corenc::Mesh::CEdgeLinearBasis:



Public Member Functions

- CEdgeLinearBasis ()
- CEdgeLinearBasis (const Point &, const Point &)
- CEdgeLinearBasis (const Point ∗)
- CEdgeLinearBasis (const CEdgeLinearBasis &)
- CEdgeLinearBasis & operator= (const CEdgeLinearBasis &e)
- ∼CEdgeLinearBasis ()
- const int GetNumberOfShapeFunctions () const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const int IncreaseOrder ()
- const double GetMeasure () const
- const double GetWeight (const int, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const

6.11.1 Constructor & Destructor Documentation

6.11.1.1 CEdgeLinearBasis() [1/4]

```
CEdgeLinearBasis::CEdgeLinearBasis ( )
```

6.11.1.2 CEdgeLinearBasis() [2/4]

```
CEdgeLinearBasis::CEdgeLinearBasis (
            const Point & p0,
            const Point & p1 )
```

6.11.1.3 CEdgeLinearBasis() [3/4]

```
CEdgeLinearBasis::CEdgeLinearBasis (
             const Point * p )
```

6.11.1.4 CEdgeLinearBasis() [4/4]

```
CEdgeLinearBasis::CEdgeLinearBasis (
             const CEdgeLinearBasis & e )
```

6.11.1.5 ∼CEdgeLinearBasis()

```
corenc::Mesh::CEdgeLinearBasis::∼CEdgeLinearBasis ( )  [inline]
```

6.11.2 Member Function Documentation

6.11.2.1 GetGradShapeFunction()

```
const Point CEdgeLinearBasis::GetGradShapeFunction (
             const int k,
             const Point &  ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.11.2.2 GetMeasure()

```
const double corenc::Mesh::CEdgeLinearBasis::GetMeasure ( ) const  [inline], [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.11.2.3 GetNormal()

```
const Point CEdgeLinearBasis::GetNormal ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.11.2.4 GetNumberOfShapeFunctions()

```
const int CEdgeLinearBasis::GetNumberOfShapeFunctions ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.11.2.5 GetShapeFunction()

```
const double CEdgeLinearBasis::GetShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.11.2.6 GetWeight()

```
const double CEdgeLinearBasis::GetWeight (
            const int ,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const
```

6.11.2.7 IncreaseOrder()

```
const int CEdgeLinearBasis::IncreaseOrder ( )
```

6.11.2.8 operator=()

```
CEdgeLinearBasis & corenc::Mesh::CEdgeLinearBasis::operator= (
            const CEdgeLinearBasis & e )  [inline]
```

6.11.2.9 ReverseNormal()

```
void CEdgeLinearBasis::ReverseNormal ( )  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

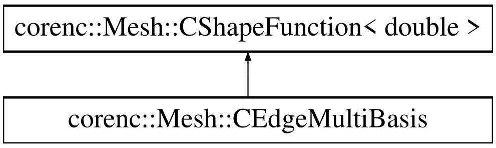
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/Edge.h
- CoreNCFEM/FiniteElements/Edge.cpp

6.12 corenc::Mesh::CEdgeMultiBasis Class Reference

```
#include <Edge.h>
```

Inheritance diagram for corenc::Mesh::CEdgeMultiBasis:



Public Member Functions

- CEdgeMultiBasis ()
- CEdgeMultiBasis (const Point &, const Point &)
- CEdgeMultiBasis (const Point ∗)
- CEdgeMultiBasis (const CEdgeMultiBasis &)
- CEdgeMultiBasis & operator= (const CEdgeMultiBasis &e)
- ∼CEdgeMultiBasis ()
- const int GetNumberOfShapeFunctions () const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetWeight (const int node, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const
- const int IncreaseOrder ()
- const double GetMeasure () const

6.12.1 Constructor & Destructor Documentation

6.12.1.1 CEdgeMultiBasis() [1/4]

```
CEdgeMultiBasis::CEdgeMultiBasis ( )
```

6.12.1.2 CEdgeMultiBasis() [2/4]

```
CEdgeMultiBasis::CEdgeMultiBasis (
            const Point & p0,
            const Point & p1 )
```

6.12.1.3 CEdgeMultiBasis() [3/4]

```
CEdgeMultiBasis::CEdgeMultiBasis (
            const Point * p )
```

6.12.1.4 CEdgeMultiBasis() [4/4]

```
CEdgeMultiBasis::CEdgeMultiBasis (
            const CEdgeMultiBasis & e )
```

6.12.1.5 ∼CEdgeMultiBasis()

```
corenc::Mesh::CEdgeMultiBasis::~CEdgeMultiBasis ( )  [inline]
```

6.12.2 Member Function Documentation

6.12.2.1 GetGradShapeFunction()

```
const Point CEdgeMultiBasis::GetGradShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.12.2.2 GetMeasure()

```
const double corenc::Mesh::CEdgeMultiBasis::GetMeasure ( ) const  [inline], [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.12.2.3 GetNormal()

```
const Point CEdgeMultiBasis::GetNormal ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.12.2.4 GetNumberOfShapeFunctions()

```
const int CEdgeMultiBasis::GetNumberOfShapeFunctions ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.12.2.5 GetShapeFunction()

```
const double CEdgeMultiBasis::GetShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.12.2.6 GetWeight()

```
const double corenc::Mesh::CEdgeMultiBasis::GetWeight (
            const int node,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const  [inline]
```

6.12.2.7 IncreaseOrder()

```
const int corenc::Mesh::CEdgeMultiBasis::IncreaseOrder ( )  [inline]
```

6.12.2.8 operator=()

```
CEdgeMultiBasis & corenc::Mesh::CEdgeMultiBasis::operator= (
            const CEdgeMultiBasis & e )  [inline]
```

6.12.2.9 ReverseNormal()

```
void CEdgeMultiBasis::ReverseNormal ( )  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

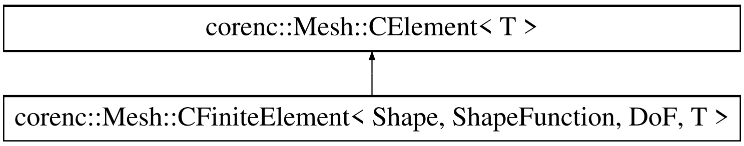
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/Edge.h
- CoreNCFEM/FiniteElements/Edge.cpp

6.13 corenc::Mesh::CElement< T > Class Template Reference

```
#include <FiniteElement.h>
```

Inheritance diagram for corenc::Mesh::CElement< T >:



Public Member Functions

- CElement ()
- virtual ~CElement ()
- virtual const int GetType () const =0
- virtual CElement * Clone () const =0
- virtual const int GetDoFs () const =0
- virtual const int GetNode (const int) const =0
- virtual const int GetNeighbour (const int) const =0
- virtual void SetNeighbour (const int k, const int elem)=0
- virtual void SetType (const int)=0
- virtual void SetNode (const int, const int)=0
- virtual const int GetNumberOfNodes () const =0
- virtual const double GetShapeFunction (const int, const Point &) const =0
- virtual const Point GetGradShapeFunction (const int, const Point &) const =0
- virtual const Point GetNormal () const =0
- virtual void ReverseNormal ()=0
- virtual const int IncreaseOrder ()=0
- virtual const double Integrate (const std::function< const double(const Point &)> &, const std::vector< Point > &v) const =0
- virtual const Point Integrate (const std::function< const Point(const Point &)> &, const std::vector< Point > &v) const =0
- virtual const std::vector< double > Integrate (const std::function< const std::vector< double >(const Point &)> &, const std::vector< Point > &) const =0
- virtual const double GetMeasure () const =0
- virtual const double GetWeight (const int, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const =0

6.13.1 Constructor & Destructor Documentation

6.13.1.1 CElement()

```
template<class T >
corenc::Mesh::CElement< T >::CElement ( )  [inline]
```

6.13.1.2 ∼CElement()

```
template<class T >
virtual corenc::Mesh::CElement< T >::∼CElement ( ) [inline], [virtual]
```

6.13.2 Member Function Documentation

6.13.2.1 Clone()

```
template<class T >
virtual CElement * corenc::Mesh::CElement< T >::Clone ( ) const [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.2 GetDoFs()

```
template<class T >
virtual const int corenc::Mesh::CElement< T >::GetDoFs ( ) const [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.3 GetGradShapeFunction()

```
template<class T >
virtual const Point corenc::Mesh::CElement< T >::GetGradShapeFunction (
            const int ,
            const Point & ) const [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.4 GetMeasure()

```
template<class T >
virtual const double corenc::Mesh::CElement< T >::GetMeasure ( ) const [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.5 GetNeighbour()

```
template<class T >
virtual const int corenc::Mesh::CElement< T >::GetNeighbour (
            const int  ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.6 GetNode()

```
template<class T >
virtual const int corenc::Mesh::CElement< T >::GetNode (
            const int  ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.7 GetNormal()

```
template<class T >
virtual const Point corenc::Mesh::CElement< T >::GetNormal ( ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.8 GetNumberOfNodes()

```
template<class T >
virtual const int corenc::Mesh::CElement< T >::GetNumberOfNodes ( ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.9 GetShapeFunction()

```
template<class T >
virtual const double corenc::Mesh::CElement< T >::GetShapeFunction (
            const int ,
            const Point &  ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.10 GetType()

```
template<class T >
virtual const int corenc::Mesh::CElement< T >::GetType ( ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.11 GetWeight()

```
template<class T >
virtual const double corenc::Mesh::CElement< T >::GetWeight (
            const int ,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.12 IncreaseOrder()

```
template<class T >
virtual const int corenc::Mesh::CElement< T >::IncreaseOrder ( )  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.13 Integrate() [1/3]

```
template<class T >
virtual const double corenc::Mesh::CElement< T >::Integrate (
            const std::function< const double(const Point &)> & ,
            const std::vector< Point > & v ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.14 Integrate() [2/3]

```
template<class T >
virtual const Point corenc::Mesh::CElement< T >::Integrate (
            const std::function< const Point(const Point &)> & ,
            const std::vector< Point > & v ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.15 Integrate() [3/3]

```
template<class T >
virtual const std::vector< double > corenc::Mesh::CElement< T >::Integrate (
            const std::function< const std::vector< double >(const Point &)> & ,
            const std::vector< Point > & ) const [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.16 ReverseNormal()

```
template<class T >
virtual void corenc::Mesh::CElement< T >::ReverseNormal ( ) [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.17 SetNeighbour()

```
template<class T >
virtual void corenc::Mesh::CElement< T >::SetNeighbour (
            const int k,
            const int elem ) [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.18 SetNode()

```
template<class T >
virtual void corenc::Mesh::CElement< T >::SetNode (
            const int ,
            const int ) [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

6.13.2.19 SetType()

```
template<class T >
virtual void corenc::Mesh::CElement< T >::SetType (
            const int  )  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >, corenc::Mesh::CFiniteElement< Shape, ShapeFu
and corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >.

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

- CoreNCFEM/FiniteElements/FiniteElement.h

6.14 corenc::Mesh::CElement2D< T > Class Template Reference

```
#include <FiniteElement2D.h>
```

Public Member Functions

- CElement2D ()
- virtual ~CElement2D ()
- virtual const int GetType () const =0
- virtual CElement2D * Clone () const =0
- virtual const int GetDoFs () const =0
- virtual const int GetNode (const int) const =0
- virtual const int GetNeighbour (const int) const =0
- virtual void SetNeighbour (const int k, const int elem)=0
- virtual void SetType (const int)=0
- virtual void SetNode (const int, const int)=0
- virtual const int GetNumberOfNodes () const =0
- virtual const double GetShapeFunction (const int, const Point &) const =0
- virtual const Point GetGradShapeFunction (const int, const Point &) const =0
- virtual const Point GetNormal () const =0
- virtual void ReverseNormal ()=0
- virtual const int IncreaseOrder ()=0
- virtual const int SetOrder (const int px, const int py)=0
- virtual const double Integrate (const std::function< const double(const Point &)> &, const std::vector< Point > &v) const =0
- virtual const Point Integrate (const std::function< const Point(const Point &)> &, const std::vector< Point > &v) const =0
- virtual const std::vector< double > Integrate (const std::function< const std::vector< double >(const Point &)> &, const std::vector< Point > &) const =0
- virtual const double GetMeasure () const =0
- virtual const double GetWeight (const int, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const =0

6.14.1 Constructor & Destructor Documentation

6.14.1.1 CElement2D()

```
template<class T >
corenc::Mesh::CElement2D< T >::CElement2D ( )  [inline]
```

6.14.1.2 ∼CElement2D()

```
template<class T >
virtual corenc::Mesh::CElement2D< T >::∼CElement2D ( )  [inline], [virtual]
```

6.14.2 Member Function Documentation

6.14.2.1 Clone()

```
template<class T >
virtual CElement2D * corenc::Mesh::CElement2D< T >::Clone ( ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.2 GetDoFs()

```
template<class T >
virtual const int corenc::Mesh::CElement2D< T >::GetDoFs ( ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.3 GetGradShapeFunction()

```
template<class T >
virtual const Point corenc::Mesh::CElement2D< T >::GetGradShapeFunction (
            const int ,
            const Point &  ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.4 GetMeasure()

```
template<class T >
virtual const double corenc::Mesh::CElement2D< T >::GetMeasure ( ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.5 GetNeighbour()

```
template<class T >
virtual const int corenc::Mesh::CElement2D< T >::GetNeighbour (
            const int  ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.6 GetNode()

```
template<class T >
virtual const int corenc::Mesh::CElement2D< T >::GetNode (
            const int  ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.7 GetNormal()

```
template<class T >
virtual const Point corenc::Mesh::CElement2D< T >::GetNormal ( ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.8 GetNumberOfNodes()

```
template<class T >
virtual const int corenc::Mesh::CElement2D< T >::GetNumberOfNodes ( ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.9 GetShapeFunction()

```
template<class T >
virtual const double corenc::Mesh::CElement2D< T >::GetShapeFunction (
            const int ,
            const Point & ) const [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.10 GetType()

```
template<class T >
virtual const int corenc::Mesh::CElement2D< T >::GetType ( ) const [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.11 GetWeight()

```
template<class T >
virtual const double corenc::Mesh::CElement2D< T >::GetWeight (
            const int ,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.12 IncreaseOrder()

```
template<class T >
virtual const int corenc::Mesh::CElement2D< T >::IncreaseOrder ( ) [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.13 Integrate() [1/3]

```
template<class T >
virtual const double corenc::Mesh::CElement2D< T >::Integrate (
            const std::function< const double(const Point &)> & ,
            const std::vector< Point > & v ) const [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.14 Integrate() [2/3]

```
template<class T >
virtual const Point corenc::Mesh::CElement2D< T >::Integrate (
            const std::function< const Point(const Point &)> & ,
            const std::vector< Point > & v ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.15 Integrate() [3/3]

```
template<class T >
virtual const std::vector< double > corenc::Mesh::CElement2D< T >::Integrate (
            const std::function< const std::vector< double >(const Point &)> & ,
            const std::vector< Point > &  ) const  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.16 ReverseNormal()

```
template<class T >
virtual void corenc::Mesh::CElement2D< T >::ReverseNormal ( )  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.17 SetNeighbour()

```
template<class T >
virtual void corenc::Mesh::CElement2D< T >::SetNeighbour (
            const int k,
            const int elem )  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.18 SetNode()

```
template<class T >
virtual void corenc::Mesh::CElement2D< T >::SetNode (
            const int ,
            const int  )  [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.19 SetOrder()

```
template<class T >
virtual const int corenc::Mesh::CElement2D< T >::SetOrder (
            const int px,
            const int py ) [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

6.14.2.20 SetType()

```
template<class T >
virtual void corenc::Mesh::CElement2D< T >::SetType (
            const int ) [pure virtual]
```

Implemented in corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >.

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

- CoreNCFEM/FiniteElements/FiniteElement2D.h

6.15 corenc::Mesh::CElement2D< bool > Class Reference

```
#include <FiniteElement2D.h>
```

Public Member Functions

- CElement2D ()
- virtual ~CElement2D ()
- virtual const int GetType () const =0
- virtual CElement2D * Clone () const =0
- virtual const int GetDoFs () const =0
- virtual const int GetNode (const int) const =0
- virtual const int GetNeighbour (const int) const =0
- virtual void SetNeighbour (const int k, const int elem)=0
- virtual void SetType (const int)=0
- virtual void SetNode (const int, const int)=0
- virtual const int GetNumberOfNodes () const =0
- virtual const double GetShapeFunction (const int, const Point &) const =0
- virtual const Point GetGradShapeFunction (const int, const Point &) const =0
- virtual const Point GetNormal () const =0
- virtual void ReverseNormal ()=0
- virtual const int SetOrder (const int px, const int py)=0
- virtual const double Integrate (const function_dp &, const std::vector< Point > &v) const =0
- virtual const Point Integrate (const std::function< const Point(const Point &)> &, const std::vector< Point > &v) const =0
- virtual const std::vector< double > Integrate (const std::function< const std::vector< double >(const Point &)> &, const std::vector< Point > &) const =0
- virtual const double GetWeight (const int, const std::vector< Point > &verts, const function_dp &f) const =0
- virtual const int IncreaseOrder ()=0
- virtual const double GetMeasure () const =0

6.15.1 Constructor & Destructor Documentation

6.15.1.1 CElement2D()

`corenc::Mesh::CElement2D< bool >::CElement2D ( ) [inline]`

6.15.1.2 ~CElement2D()

`virtual corenc::Mesh::CElement2D< bool >::~CElement2D ( ) [inline], [virtual]`

6.15.2 Member Function Documentation

6.15.2.1 Clone()

`virtual CElement2D * corenc::Mesh::CElement2D< bool >::Clone ( ) const [pure virtual]`

6.15.2.2 GetDoFs()

`virtual const int corenc::Mesh::CElement2D< bool >::GetDoFs ( ) const [pure virtual]`

6.15.2.3 GetGradShapeFunction()

```
virtual const Point corenc::Mesh::CElement2D< bool >::GetGradShapeFunction (
            const int ,
            const Point & ) const [pure virtual]
```

6.15.2.4 GetMeasure()

`virtual const double corenc::Mesh::CElement2D< bool >::GetMeasure ( ) const [pure virtual]`

6.15.2.5 GetNeighbour()

```
virtual const int corenc::Mesh::CElement2D< bool >::GetNeighbour (
            const int ) const [pure virtual]
```

6.15.2.6 GetNode()

```
virtual const int corenc::Mesh::CElement2D< bool >::GetNode (
            const int ) const [pure virtual]
```

6.15.2.7 GetNormal()

```
virtual const Point corenc::Mesh::CElement2D< bool >::GetNormal ( ) const [pure virtual]
```

6.15.2.8 GetNumberOfNodes()

```
virtual const int corenc::Mesh::CElement2D< bool >::GetNumberOfNodes ( ) const [pure virtual]
```

6.15.2.9 GetShapeFunction()

```
virtual const double corenc::Mesh::CElement2D< bool >::GetShapeFunction (
            const int ,
            const Point & ) const [pure virtual]
```

6.15.2.10 GetType()

```
virtual const int corenc::Mesh::CElement2D< bool >::GetType ( ) const [pure virtual]
```

6.15.2.11 GetWeight()

```
virtual const double corenc::Mesh::CElement2D< bool >::GetWeight (
            const int ,
            const std::vector< Point > & verts,
            const function_dp & f ) const [pure virtual]
```

6.15.2.12 IncreaseOrder()

```
virtual const int corenc::Mesh::CElement2D< bool >::IncreaseOrder ( )  [pure virtual]
```

6.15.2.13 Integrate() [1/3]

```
virtual const double corenc::Mesh::CElement2D< bool >::Integrate (
            const function_dp & ,
            const std::vector< Point > & v ) const  [pure virtual]
```

6.15.2.14 Integrate() [2/3]

```
virtual const Point corenc::Mesh::CElement2D< bool >::Integrate (
            const std::function< const Point(const Point &)> & ,
            const std::vector< Point > & v ) const  [pure virtual]
```

6.15.2.15 Integrate() [3/3]

```
virtual const std::vector< double > corenc::Mesh::CElement2D< bool >::Integrate (
            const std::function< const std::vector< double >(const Point &)> & ,
            const std::vector< Point > &  ) const  [pure virtual]
```

6.15.2.16 ReverseNormal()

```
virtual void corenc::Mesh::CElement2D< bool >::ReverseNormal ( )  [pure virtual]
```

6.15.2.17 SetNeighbour()

```
virtual void corenc::Mesh::CElement2D< bool >::SetNeighbour (
            const int k,
            const int elem )  [pure virtual]
```

6.15.2.18 SetNode()

```
virtual void corenc::Mesh::CElement2D< bool >::SetNode (
            const int ,
            const int  )  [pure virtual]
```

6.15.2.19 SetOrder()

```
virtual const int corenc::Mesh::CElement2D< bool >::SetOrder (
            const int px,
            const int py )  [pure virtual]
```

6.15.2.20 SetType()

```
virtual void corenc::Mesh::CElement2D< bool >::SetType (
            const int  )  [pure virtual]
```

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

- CoreNCFEM/FiniteElements/FiniteElement2D.h

6.16 corenc::Mesh::CElement< bool > Class Reference

```
#include <FiniteElement.h>
```

Public Member Functions

- CElement ()
- virtual ~CElement ()
- virtual const int GetType () const =0
- virtual CElement * Clone () const =0
- virtual const int GetDoFs () const =0
- virtual const int GetNode (const int) const =0
- virtual const int GetNeighbour (const int) const =0
- virtual void SetNeighbour (const int k, const int elem)=0
- virtual void SetType (const int)=0
- virtual void SetNode (const int, const int)=0
- virtual const int GetNumberOfNodes () const =0
- virtual const double GetShapeFunction (const int, const Point &) const =0
- virtual const Point GetGradShapeFunction (const int, const Point &) const =0
- virtual const Point GetNormal () const =0
- virtual void ReverseNormal ()=0
- virtual const double Integrate (const function_dp &, const std::vector< Point > &v) const =0
- virtual const Point Integrate (const std::function< const Point(const Point &)> &, const std::vector< Point > &v) const =0
- virtual const std::vector< double > Integrate (const std::function< const std::vector< double >(const Point &)> &, const std::vector< Point > &) const =0
- virtual const double GetWeight (const int, const std::vector< Point > &verts, const function_dp &f) const =0
- virtual const int IncreaseOrder ()=0
- virtual const double GetMeasure () const =0

6.16.1 Constructor & Destructor Documentation

6.16.1.1 CElement()

```
corenc::Mesh::CElement< bool >::CElement ( ) [inline]
```

6.16.1.2 ∼CElement()

```
virtual corenc::Mesh::CElement< bool >::~CElement ( ) [inline], [virtual]
```

6.16.2 Member Function Documentation

6.16.2.1 Clone()

```
virtual CElement * corenc::Mesh::CElement< bool >::Clone ( ) const [pure virtual]
```

6.16.2.2 GetDoFs()

```
virtual const int corenc::Mesh::CElement< bool >::GetDoFs ( ) const [pure virtual]
```

6.16.2.3 GetGradShapeFunction()

```
virtual const Point corenc::Mesh::CElement< bool >::GetGradShapeFunction (
            const int ,
            const Point & ) const [pure virtual]
```

6.16.2.4 GetMeasure()

```
virtual const double corenc::Mesh::CElement< bool >::GetMeasure ( ) const [pure virtual]
```

6.16.2.5 GetNeighbour()

```
virtual const int corenc::Mesh::CElement< bool >::GetNeighbour (
            const int ) const [pure virtual]
```

6.16.2.6 GetNode()

```
virtual const int corenc::Mesh::CElement< bool >::GetNode (
            const int ) const [pure virtual]
```

6.16.2.7 GetNormal()

```
virtual const Point corenc::Mesh::CElement< bool >::GetNormal ( ) const [pure virtual]
```

6.16.2.8 GetNumberOfNodes()

```
virtual const int corenc::Mesh::CElement< bool >::GetNumberOfNodes ( ) const [pure virtual]
```

6.16.2.9 GetShapeFunction()

```
virtual const double corenc::Mesh::CElement< bool >::GetShapeFunction (
            const int ,
            const Point & ) const [pure virtual]
```

6.16.2.10 GetType()

```
virtual const int corenc::Mesh::CElement< bool >::GetType ( ) const [pure virtual]
```

6.16.2.11 GetWeight()

```
virtual const double corenc::Mesh::CElement< bool >::GetWeight (
            const int ,
            const std::vector< Point > & verts,
            const function_dp & f ) const [pure virtual]
```

6.16.2.12 IncreaseOrder()

```
virtual const int corenc::Mesh::CElement< bool >::IncreaseOrder ( ) [pure virtual]
```

6.16.2.13 Integrate() [1/3]

```
virtual const double corenc::Mesh::CElement< bool >::Integrate (
            const function_dp & ,
            const std::vector< Point > & v ) const [pure virtual]
```

6.16.2.14 Integrate() [2/3]

```
virtual const Point corenc::Mesh::CElement< bool >::Integrate (
            const std::function< const Point(const Point &)> & ,
            const std::vector< Point > & v ) const [pure virtual]
```

6.16.2.15 Integrate() [3/3]

```
virtual const std::vector< double > corenc::Mesh::CElement< bool >::Integrate (
            const std::function< const std::vector< double >(const Point &)> & ,
            const std::vector< Point > & ) const [pure virtual]
```

6.16.2.16 ReverseNormal()

```
virtual void corenc::Mesh::CElement< bool >::ReverseNormal ( ) [pure virtual]
```

6.16.2.17 SetNeighbour()

```
virtual void corenc::Mesh::CElement< bool >::SetNeighbour (
            const int k,
            const int elem ) [pure virtual]
```

6.16.2.18 SetNode()

```
virtual void corenc::Mesh::CElement< bool >::SetNode (
            const int ,
            const int ) [pure virtual]
```

6.16.2.19 SetType()

```
virtual void corenc::Mesh::CElement< bool >::SetType (
            const int ) [pure virtual]
```

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

- CoreNCFEM/FiniteElements/FiniteElement.h

6.17 corenc::method::CFEMethod< Type > Class Template Reference

```
#include <FEMethod.h>
```

Public Member Functions

- CFEMethod ()
- virtual ∼CFEMethod ()
- virtual const int Assemble ()=0
- virtual const Type GetSolution (const std::vector< double > &point) const =0
- virtual const std::vector< Type > GetSolution () const =0
- virtual const Type GetMaxSolution () const =0
- virtual const Type GetMinSolution () const =0

6.17.1 Constructor & Destructor Documentation

6.17.1.1 CFEMethod()

```
template<class Type >
corenc::method::CFEMethod< Type >::CFEMethod ( ) [inline]
```

6.17.1.2 ∼CFEMethod()

```
template<class Type >
virtual corenc::method::CFEMethod< Type >::∼CFEMethod ( ) [inline], [virtual]
```

6.17.2 Member Function Documentation

6.17.2.1 Assemble()

```
template<class Type >
virtual const int corenc::method::CFEMethod< Type >::Assemble ( ) [pure virtual]
```

6.17.2.2 GetMaxSolution()

```
template<class Type >
virtual const Type corenc::method::CFEMethod< Type >::GetMaxSolution ( ) const [pure virtual]
```

6.17.2.3 GetMinSolution()

```
template<class Type >
virtual const Type corenc::method::CFEMethod< Type >::GetMinSolution ( ) const [pure virtual]
```

6.17.2.4 GetSolution() [1/2]

```
template<class Type >
virtual const std::vector< Type > corenc::method::CFEMethod< Type >::GetSolution ( ) const
[pure virtual]
```

6.17.2.5 GetSolution() [2/2]

```
template<class Type >
virtual const Type corenc::method::CFEMethod< Type >::GetSolution (
            const std::vector< double > & point ) const [pure virtual]
```

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

- CoreNCFEM/Methods/FEMethod.h

6.18 corenc::method::CFEMethodZero< Type > Class Template Reference

```
#include <FEMethodZero.h>
```

Public Member Functions

- CFEMethodZero ()
- virtual ∼CFEMethodZero ()
- virtual const int Assemble ()=0
- virtual const Type GetSolution (const std::vector< double > &point) const =0
- virtual const std::vector< Type > GetSolution () const =0
- virtual const Type GetMaxSolution () const =0
- virtual const Type GetMinSolution () const =0

6.18.1 Constructor & Destructor Documentation

6.18.1.1 CFEMethodZero()

```
template<class Type >
corenc::method::CFEMethodZero< Type >::CFEMethodZero ( )  [inline]
```

6.18.1.2 ∼CFEMethodZero()

```
template<class Type >
virtual corenc::method::CFEMethodZero< Type >::∼CFEMethodZero ( )  [inline], [virtual]
```

6.18.2 Member Function Documentation

6.18.2.1 Assemble()

```
template<class Type >
virtual const int corenc::method::CFEMethodZero< Type >::Assemble ( )  [pure virtual]
```

6.18.2.2 GetMaxSolution()

```
template<class Type >
virtual const Type corenc::method::CFEMethodZero< Type >::GetMaxSolution ( ) const  [pure
virtual]
```

6.18.2.3 GetMinSolution()

```
template<class Type >
virtual const Type corenc::method::CFEMethodZero< Type >::GetMinSolution ( ) const  [pure
virtual]
```

6.18.2.4 GetSolution() [1/2]

```
template<class Type >
virtual const std::vector< Type > corenc::method::CFEMethodZero< Type >::GetSolution ( )
const  [pure virtual]
```

6.18.2.5 GetSolution() [2/2]

```
template<class Type >
virtual const Type corenc::method::CFEMethodZero< Type >::GetSolution (
            const std::vector< double > & point ) const  [pure virtual]
```

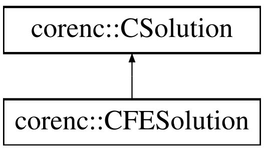
The documentation for this class was generated from the following file:

- CoreNCFEM/Methods/FEMethodZero.h

6.19 corenc::CFESolution Class Reference

```
#include <FESolution.h>
```

Inheritance diagram for corenc::CFESolution:



Public Member Functions

- CFESolution ()
- ∼CFESolution ()
- CFESolution & operator= (const CFESolution &fe)
- CFESolution & operator= (const double fe)
- CFESolution (const CFESolution &fe)
- CFESolution (const double &fe)
- operator double () const
- const bool operator== (const CFESolution &fe)
- const bool operator!= (const CFESolution &fe)
- CFESolution & operator+= (const CFESolution &fe)
- CFESolution & operator-= (const CFESolution &fe)
- CFESolution & operator∗= (const CFESolution &fe)
- CFESolution & operator/= (const CFESolution &fe)

Friends

- const double operator* (const CFESolution &lhs, const CFESolution &rhs)
- const double operator* (const CFESolution &lhs, const double rhs)
- const double operator* (const double lhs, const CFESolution &rhs)
- const double operator- (const CFESolution &lhs, const CFESolution &rhs)
- const double operator+ (const CFESolution &lhs, const CFESolution &rhs)
- const double operator/ (const CFESolution &lhs, const CFESolution &rhs)

6.19.1 Constructor & Destructor Documentation

6.19.1.1 CFESolution() [1/3]

```
corenc::CFESolution::CFESolution ( ) [inline]
```

6.19.1.2 ∼CFESolution()

```
corenc::CFESolution::∼CFESolution ( ) [inline]
```

6.19.1.3 CFESolution() [2/3]

```
corenc::CFESolution::CFESolution (
            const CFESolution & fe ) [inline]
```

6.19.1.4 CFESolution() [3/3]

```
corenc::CFESolution::CFESolution (
            const double & fe ) [inline]
```

6.19.2 Member Function Documentation

6.19.2.1 operator double()

```
corenc::CFESolution::operator double ( ) const [inline]
```

6.19.2.2 operator"!=()

```
const bool corenc::CFESolution::operator!= (
              const CFESolution & fe )  [inline]
```

6.19.2.3 operator*=()

```
CFESolution & corenc::CFESolution::operator*= (
              const CFESolution & fe )  [inline]
```

6.19.2.4 operator+=()

```
CFESolution & corenc::CFESolution::operator+= (
              const CFESolution & fe )  [inline]
```

6.19.2.5 operator-=()

```
CFESolution & corenc::CFESolution::operator-= (
              const CFESolution & fe )  [inline]
```

6.19.2.6 operator/=()

```
CFESolution & corenc::CFESolution::operator/= (
              const CFESolution & fe )  [inline]
```

6.19.2.7 operator=() [1/2]

```
CFESolution & corenc::CFESolution::operator= (
              const CFESolution & fe )  [inline]
```

6.19.2.8 operator=() [2/2]

```
CFESolution & corenc::CFESolution::operator= (
              const double fe )  [inline]
```

6.19.2.9 operator==()

```
const bool corenc::CFESolution::operator== (
            const CFESolution & fe )  [inline]
```

6.19.3 Friends And Related Function Documentation

6.19.3.1 operator* [1/3]

```
const double operator* (
            const CFESolution & lhs,
            const CFESolution & rhs )  [friend]
```

6.19.3.2 operator* [2/3]

```
const double operator* (
            const CFESolution & lhs,
            const double rhs )  [friend]
```

6.19.3.3 operator* [3/3]

```
const double operator* (
            const double lhs,
            const CFESolution & rhs )  [friend]
```

6.19.3.4 operator+

```
const double operator+ (
            const CFESolution & lhs,
            const CFESolution & rhs )  [friend]
```

6.19.3.5 operator-

```
const double operator- (
            const CFESolution & lhs,
            const CFESolution & rhs )  [friend]
```

6.19.3.6 operator/

```
const double operator/ (
            const CFESolution & lhs,
            const CFESolution & rhs )  [friend]
```

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

- CoreNCFEM/FESolution.h

6.20 corenc::CFEweights Class Reference

```
#include <FESolution.h>
```

Public Member Functions

- CFEweights ()
- ∼CFEweights ()
- const CFESolution getWeight (const unsigned int i) const
- const int updateWeight (const unsigned int i, const CFESolution &cfe)

6.20.1 Constructor & Destructor Documentation

6.20.1.1 CFEweights()

```
corenc::CFEweights::CFEweights ( )  [inline]
```

6.20.1.2 ∼CFEweights()

```
corenc::CFEweights::∼CFEweights ( )  [inline]
```

6.20.2 Member Function Documentation

6.20.2.1 getWeight()

```
const CFESolution corenc::CFEweights::getWeight (
            const unsigned int i ) const  [inline]
```

6.20.2.2 updateWeight()

```
const int corenc::CFEweights::updateWeight (
            const unsigned int i,
            const CFESolution & cfe )  [inline]
```

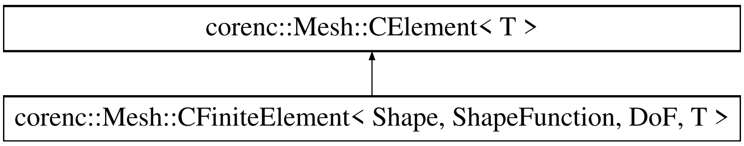
The documentation for this class was generated from the following file:

- CoreNCFEM/FESolution.h

6.21 corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T > Class Template Reference

```
#include <FiniteElement.h>
```

Inheritance diagram for corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >:



Public Member Functions

- CFiniteElement ()
- CFiniteElement (const int ∗nodes, const Point ∗points, const int dofs)
- CFiniteElement (const int ∗nodes, const Point ∗points)
- CFiniteElement (const Shape &shape, const ShapeFunction &f, const DoF &d)
- CFiniteElement (const Shape &shape, const ShapeFunction &shfunc, const DoF &dofs, const int type)
- CFiniteElement (const Shape &shape, const ShapeFunction &shfunc, const DoF &dofs, const int type, const int ∗neigs)
- CFiniteElement (const CFiniteElement< Shape, ShapeFunction, DoF > &e)
- CElement< T > ∗ Clone () const
- ∼CFiniteElement ()
- const int GetType () const
- const int GetNode (const int) const
- const int GetNeighbour (const int) const
- const Shape GetShape () const
- const ShapeFunction GetShapeFunctions () const
- const DoF GetDoF () const
- const int GetDoFs () const
- void SetNeighbour (const int k, const int elem)
- void SetType (const int)
- void SetShapeFunction (const int, const ShapeFunction &)
- void SetDoF (const DoF &)
- void SetShape (const Shape &)
- const int IncreaseOrder ()
- void SetNode (const int, const int)
- const int GetNumberOfNodes () const

- const double GetMeasure () const
- const double Integrate (const std::function< const double(const Point &)> &, const std::vector< Point > &v) const
- const Point Integrate (const std::function< const Point(const Point &)> &, const std::vector< Point > &v) const
- const std::vector< double > Integrate (const std::function< const std::vector< double >(const Point &)> &, const std::vector< Point > &) const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetWeight (const int, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const
- CFiniteElement & operator= (const CFiniteElement &e)

Friends

- const bool operator== (const CFiniteElement &e1, const CFiniteElement &e2)
- std::istream & operator>> (std::istream &is, CFiniteElement &k)

6.21.1 Constructor & Destructor Documentation

6.21.1.1 CFiniteElement() [1/7]

```
template<class Shape , class ShapeFunction , class DoF , class T >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::CFiniteElement ( )  [inline]
```

6.21.1.2 CFiniteElement() [2/7]

```
template<class Shape , class ShapeFunction , class DoF , class T >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::CFiniteElement (
            const int * nodes,
            const Point * points,
            const int dofs )  [inline]
```

6.21.1.3 CFiniteElement() [3/7]

```
template<class Shape , class ShapeFunction , class DoF , class T >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::CFiniteElement (
            const int * nodes,
            const Point * points )  [inline]
```

6.21.1.4 CFiniteElement() [4/7]

```
template<class Shape , class ShapeFunction , class DoF , class T >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::CFiniteElement (
            const Shape & shape,
            const ShapeFunction & f,
            const DoF & d )  [inline]
```

6.21.1.5 CFiniteElement() [5/7]

```
template<class Shape , class ShapeFunction , class DoF , class T >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::CFiniteElement (
            const Shape & shape,
            const ShapeFunction & shfunc,
            const DoF & dofs,
            const int type )  [inline]
```

6.21.1.6 CFiniteElement() [6/7]

```
template<class Shape , class ShapeFunction , class DoF , class T >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::CFiniteElement (
            const Shape & shape,
            const ShapeFunction & shfunc,
            const DoF & dofs,
            const int type,
            const int * neigs )  [inline]
```

6.21.1.7 CFiniteElement() [7/7]

```
template<class Shape , class ShapeFunction , class DoF , class T >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::CFiniteElement (
            const CFiniteElement< Shape, ShapeFunction, DoF > & e )  [inline]
```

6.21.1.8 ∼CFiniteElement()

```
template<class Shape , class ShapeFunction , class DoF , class T >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::~CFiniteElement ( )  [inline]
```

6.21.2 Member Function Documentation

6.21.2.1 Clone()

```
template<class Shape , class ShapeFunction , class DoF , class T >
CElement< T > * corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::Clone ( ) const
[inline], [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.2 GetDoF()

```
template<class Shape , class ShapeFunction , class DoF , class T >
const DoF corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::GetDoF
```

6.21.2.3 GetDoFs()

```
template<class Shape , class ShapeFunction , class DoF , class T >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::GetDoFs [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.4 GetGradShapeFunction()

```
template<class Shape , class ShapeFunction , class DoF , class T >
const Point corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::GetGradShapeFunction
(
            const int k,
            const Point & p ) const [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.5 GetMeasure()

```
template<class Shape , class ShapeFunction , class DoF , class T >
const double corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::GetMeasure [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.6 GetNeighbour()

```
template<class Shape , class ShapeFunction , class DoF , class T >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::GetNeighbour (
            const int k ) const  [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.7 GetNode()

```
template<class Shape , class ShapeFunction , class DoF , class T >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::GetNode (
            const int k ) const  [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.8 GetNormal()

```
template<class Shape , class ShapeFunction , class DoF , class T >
const Point corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::GetNormal  [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.9 GetNumberOfNodes()

```
template<class Shape , class ShapeFunction , class DoF , class T >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::GetNumberOfNodes
[virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.10 GetShape()

```
template<class Shape , class ShapeFunction , class DoF , class T >
const Shape corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::GetShape
```

6.21.2.11 GetShapeFunction()

```
template<class Shape , class ShapeFunction , class DoF , class T >
const double corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::GetShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.12 GetShapeFunctions()

```
template<class Shape , class ShapeFunction , class DoF , class T >
const ShapeFunction corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::GetShape↩
Functions
```

6.21.2.13 GetType()

```
template<class Shape , class ShapeFunction , class DoF , class T >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::GetType  [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.14 GetWeight()

```
template<class Shape , class ShapeFunction , class DoF , class T >
const double corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::GetWeight (
            const int node,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const  [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.15 IncreaseOrder()

```
template<class Shape , class ShapeFunction , class DoF , class T >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::IncreaseOrder  [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.16 Integrate() [1/3]

```
template<class Shape , class ShapeFunction , class DoF , class T >
const double corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::Integrate (
            const std::function< const double(const Point &)> & f,
            const std::vector< Point > & v ) const  [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.17 Integrate() [2/3]

```
template<class Shape , class ShapeFunction , class DoF , class T >
const Point corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::Integrate (
            const std::function< const Point(const Point &)> & f,
            const std::vector< Point > & v ) const  [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.18 Integrate() [3/3]

```
template<class Shape , class ShapeFunction , class DoF , class T >
const std::vector< double > corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::↩
Integrate (
            const std::function< const std::vector< double >(const Point &)> & f,
            const std::vector< Point > & v ) const  [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.19 operator=()

```
template<class Shape , class ShapeFunction , class DoF , class T >
CFiniteElement & corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::operator= (
            const CFiniteElement< Shape, ShapeFunction, DoF, T > & e )  [inline]
```

6.21.2.20 ReverseNormal()

```
template<class Shape , class ShapeFunction , class DoF , class T >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::ReverseNormal  [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.21 SetDoF()

```
template<class Shape , class ShapeFunction , class DoF , class T >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::SetDoF (
            const DoF & dof )
```

6.21.2.22 SetNeighbour()

```
template<class Shape , class ShapeFunction , class DoF , class T >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::SetNeighbour (
            const int k,
            const int elem )  [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.23 SetNode()

```
template<class Shape , class ShapeFunction , class DoF , class T >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::SetNode (
            const int k,
            const int node )  [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.2.24 SetShape()

```
template<class Shape , class ShapeFunction , class DoF , class T >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::SetShape (
            const Shape & shape )
```

6.21.2.25 SetShapeFunction()

```
template<class Shape , class ShapeFunction , class DoF , class T >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::SetShapeFunction (
            const int k,
            const ShapeFunction & func )
```

6.21.2.26 SetType()

```
template<class Shape , class ShapeFunction , class DoF , class T >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >::SetType (
            const int k )  [virtual]
```

Implements corenc::Mesh::CElement< T >.

6.21.3 Friends And Related Function Documentation

6.21.3.1 operator==

```
template<class Shape , class ShapeFunction , class DoF , class T >
const bool operator== (
            const CFiniteElement< Shape, ShapeFunction, DoF, T > & e1,
            const CFiniteElement< Shape, ShapeFunction, DoF, T > & e2 )  [friend]
```

6.21.3.2 operator>>

```
template<class Shape , class ShapeFunction , class DoF , class T >
std::istream & operator>> (
            std::istream & is,
            CFiniteElement< Shape, ShapeFunction, DoF, T > & k )  [friend]
```

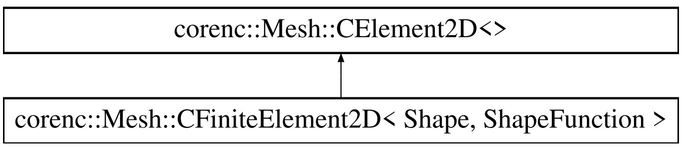
The documentation for this class was generated from the following file:

- CoreNCFEM/FiniteElements/FiniteElement.h

6.22 corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction > Class Template Reference

```
#include <FiniteElement2D.h>
```

Inheritance diagram for corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >:



Public Member Functions

- CFiniteElement2D ()
- CFiniteElement2D (const int ∗nodes, const Point ∗points, const int dofs)
- CFiniteElement2D (const int ∗nodes, const Point ∗points, const int dofs, const int type)
- CFiniteElement2D (const int ∗nodes, const Point ∗points)
- CFiniteElement2D (const Shape &shape, const ShapeFunction &f)
- CFiniteElement2D (const Shape &shape, const ShapeFunction &shfunc, const int type)
- CFiniteElement2D (const Shape &shape, const ShapeFunction &shfunc, const int type, const int ∗neigs)
- CFiniteElement2D (const CFiniteElement2D &e)
- CElement2D ∗ Clone () const
- ∼CFiniteElement2D ()
- const int GetType () const
- const int GetNode (const int) const
- const int GetNeighbour (const int) const
- const Shape GetShape () const
- const ShapeFunction GetShapeFunctions () const
- const int GetDoFs () const
- void SetNeighbour (const int k, const int elem)
- void SetType (const int)
- void SetShapeFunction (const int, const ShapeFunction &)
- void SetShape (const Shape &)
- const int SetOrder (const int px, const int py)
- void SetNode (const int, const int)
- const int GetNumberOfNodes () const
- const int IncreaseOrder ()
- const double GetMeasure () const
- const double Integrate (const std::function< const double(const Point &)> &, const std::vector< Point > &v) const
- const Point Integrate (const std::function< const Point(const Point &)> &, const std::vector< Point > &v) const
- const std::vector< double > Integrate (const std::function< const std::vector< double >(const Point &)> &, const std::vector< Point > &) const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetWeight (const int, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const
- CFiniteElement2D & operator= (const CFiniteElement2D &e)

Friends

- const bool operator== (const CFiniteElement2D &e1, const CFiniteElement2D &e2)
- std::istream & operator>> (std::istream &is, CFiniteElement2D &k)

6.22.1 Constructor & Destructor Documentation

6.22.1.1 CFiniteElement2D() [1/8]

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::CFiniteElement2D ( ) [inline]
```

6.22.1.2 CFiniteElement2D() [2/8]

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::CFiniteElement2D (
            const int * nodes,
            const Point * points,
            const int dofs ) [inline]
```

6.22.1.3 CFiniteElement2D() [3/8]

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::CFiniteElement2D (
            const int * nodes,
            const Point * points,
            const int dofs,
            const int type ) [inline]
```

6.22.1.4 CFiniteElement2D() [4/8]

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::CFiniteElement2D (
            const int * nodes,
            const Point * points ) [inline]
```

6.22.1.5 CFiniteElement2D() [5/8]

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::CFiniteElement2D (
            const Shape & shape,
            const ShapeFunction & f ) [inline]
```

6.22.1.6 CFiniteElement2D() [6/8]

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::CFiniteElement2D (
            const Shape & shape,
            const ShapeFunction & shfunc,
            const int type ) [inline]
```

6.22.1.7 CFiniteElement2D() [7/8]

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::CFiniteElement2D (
            const Shape & shape,
            const ShapeFunction & shfunc,
            const int type,
            const int * neigs ) [inline]
```

6.22.1.8 CFiniteElement2D() [8/8]

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::CFiniteElement2D (
            const CFiniteElement2D< Shape, ShapeFunction > & e ) [inline]
```

6.22.1.9 ∼CFiniteElement2D()

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::∼CFiniteElement2D ( ) [inline]
```

6.22.2 Member Function Documentation

6.22.2.1 Clone()

```
template<class Shape , class ShapeFunction >
CElement2D * corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::Clone ( ) const [inline],
[virtual]
```

Implements corenc::Mesh::CElement2D<>.

6.22.2.2 GetDoFs()

```
template<class Shape , class ShapeFunction >
const int corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::GetDoFs  [virtual]
```

Implements corenc::Mesh::CElement2D<>.

6.22.2.3 GetGradShapeFunction()

```
template<class Shape , class ShapeFunction >
const Point corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::GetGradShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CElement2D<>.

6.22.2.4 GetMeasure()

```
template<class Shape , class ShapeFunction >
const double corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::GetMeasure  [virtual]
```

Implements corenc::Mesh::CElement2D<>.

6.22.2.5 GetNeighbour()

```
template<class Shape , class ShapeFunction >
const int corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::GetNeighbour (
            const int k ) const  [virtual]
```

Implements corenc::Mesh::CElement2D<>.

6.22.2.6 GetNode()

```
template<class Shape , class ShapeFunction >
const int corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::GetNode (
            const int k ) const  [virtual]
```

Implements corenc::Mesh::CElement2D<>.

6.22.2.7 GetNormal()

template<class Shape , class ShapeFunction >
const Point corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::GetNormal [virtual]

Implements corenc::Mesh::CElement2D<>.

6.22.2.8 GetNumberOfNodes()

template<class Shape , class ShapeFunction >
const int corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::GetNumberOfNodes [virtual]

Implements corenc::Mesh::CElement2D<>.

6.22.2.9 GetShape()

template<class Shape , class ShapeFunction >
const Shape corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::GetShape

6.22.2.10 GetShapeFunction()

template<class Shape , class ShapeFunction >
const double corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::GetShapeFunction (
const int *k*,
const Point & *p*) const [virtual]

Implements corenc::Mesh::CElement2D<>.

6.22.2.11 GetShapeFunctions()

template<class Shape , class ShapeFunction >
const ShapeFunction corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::GetShapeFunctions

6.22.2.12 GetType()

template<class Shape , class ShapeFunction >
const int corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::GetType [virtual]

Implements corenc::Mesh::CElement2D<>.

6.22.2.13 GetWeight()

```
template<class Shape , class ShapeFunction >
const double corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::GetWeight (
            const int node,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const  [virtual]
```

Implements corenc::Mesh::CElement2D<>.

6.22.2.14 IncreaseOrder()

```
template<class Shape , class ShapeFunction >
const int corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::IncreaseOrder  [virtual]
```

Implements corenc::Mesh::CElement2D<>.

6.22.2.15 Integrate() [1/3]

```
template<class Shape , class ShapeFunction >
const double corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::Integrate (
            const std::function< const double(const Point &)> & f,
            const std::vector< Point > & v ) const  [virtual]
```

Implements corenc::Mesh::CElement2D<>.

6.22.2.16 Integrate() [2/3]

```
template<class Shape , class ShapeFunction >
const Point corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::Integrate (
            const std::function< const Point(const Point &)> & f,
            const std::vector< Point > & v ) const  [virtual]
```

Implements corenc::Mesh::CElement2D<>.

6.22.2.17 Integrate() [3/3]

```
template<class Shape , class ShapeFunction >
const std::vector< double > corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::Integrate
(
            const std::function< const std::vector< double >(const Point &)> & f,
            const std::vector< Point > & v ) const  [virtual]
```

Implements corenc::Mesh::CElement2D<>.

6.22.2.18 operator=()

```
template<class Shape , class ShapeFunction >
CFiniteElement2D & corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::operator= (
            const CFiniteElement2D< Shape, ShapeFunction > & e )  [inline]
```

6.22.2.19 ReverseNormal()

```
template<class Shape , class ShapeFunction >
void corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::ReverseNormal  [virtual]
```

Implements corenc::Mesh::CElement2D<>.

6.22.2.20 SetNeighbour()

```
template<class Shape , class ShapeFunction >
void corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::SetNeighbour (
            const int k,
            const int elem )  [virtual]
```

Implements corenc::Mesh::CElement2D<>.

6.22.2.21 SetNode()

```
template<class Shape , class ShapeFunction >
void corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::SetNode (
            const int k,
            const int node )  [virtual]
```

Implements corenc::Mesh::CElement2D<>.

6.22.2.22 SetOrder()

```
template<class Shape , class ShapeFunction >
const int corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::SetOrder (
            const int px,
            const int py )  [virtual]
```

Implements corenc::Mesh::CElement2D<>.

6.22.2.23 SetShape()

```
template<class Shape , class ShapeFunction >
void corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::SetShape (
            const Shape & shape )
```

6.22.2.24 SetShapeFunction()

```
template<class Shape , class ShapeFunction >
void corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::SetShapeFunction (
            const int k,
            const ShapeFunction & func )
```

6.22.2.25 SetType()

```
template<class Shape , class ShapeFunction >
void corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >::SetType (
            const int k )  [virtual]
```

Implements corenc::Mesh::CElement2D<>.

6.22.3 Friends And Related Function Documentation

6.22.3.1 operator==

```
template<class Shape , class ShapeFunction >
const bool operator== (
            const CFiniteElement2D< Shape, ShapeFunction > & e1,
            const CFiniteElement2D< Shape, ShapeFunction > & e2 )  [friend]
```

6.22.3.2 operator>>

```
template<class Shape , class ShapeFunction >
std::istream & operator>> (
            std::istream & is,
            CFiniteElement2D< Shape, ShapeFunction > & k )  [friend]
```

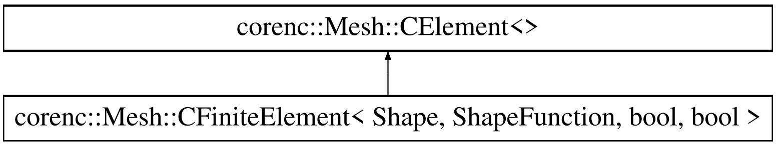
The documentation for this class was generated from the following file:

- CoreNCFEM/FiniteElements/FiniteElement2D.h

6.23 corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool > Class Template Reference

```
#include <FiniteElement.h>
```

Inheritance diagram for corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >:



Public Member Functions

- CFiniteElement ()
- CFiniteElement (const int *nodes, const Point *points, const int dofs)
- CFiniteElement (const int *nodes, const Point *points, const int dofs, const int type)
- CFiniteElement (const int *nodes, const Point *points)
- CFiniteElement (const Shape &shape, const ShapeFunction &f)
- CFiniteElement (const Shape &shape, const ShapeFunction &shfunc, const int type)
- CFiniteElement (const Shape &shape, const ShapeFunction &shfunc, const int type, const int *neigs)
- CFiniteElement (const CFiniteElement< Shape, ShapeFunction > &e)
- CElement * Clone () const
- ~CFiniteElement ()
- const int GetType () const
- const int GetNode (const int) const
- const int GetNeighbour (const int) const
- const Shape GetShape () const
- const ShapeFunction GetShapeFunctions () const
- const int GetDoFs () const
- void SetNeighbour (const int k, const int elem)
- void SetType (const int)
- void SetShapeFunction (const int, const ShapeFunction &)
- void SetShape (const Shape &)
- void SetNode (const int, const int)
- const int GetNumberOfNodes () const
- const int IncreaseOrder ()
- const double GetMeasure () const
- const double Integrate (const std::function< const double(const Point &)> &, const std::vector< Point > &v) const
- const Point Integrate (const std::function< const Point(const Point &)> &, const std::vector< Point > &v) const
- const std::vector< double > Integrate (const std::function< const std::vector< double >(const Point &)> &, const std::vector< Point > &) const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetWeight (const int, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const
- CFiniteElement & operator= (const CFiniteElement &e)

Friends

- const bool operator== (const CFiniteElement &e1, const CFiniteElement &e2)
- std::istream & operator>> (std::istream &is, CFiniteElement &k)

6.23.1 Constructor & Destructor Documentation

6.23.1.1 CFiniteElement() [1/8]

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::CFiniteElement ( )  [inline]
```

6.23.1.2 CFiniteElement() [2/8]

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::CFiniteElement (
            const int * nodes,
            const Point * points,
            const int dofs )  [inline]
```

6.23.1.3 CFiniteElement() [3/8]

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::CFiniteElement (
            const int * nodes,
            const Point * points,
            const int dofs,
            const int type )  [inline]
```

6.23.1.4 CFiniteElement() [4/8]

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::CFiniteElement (
            const int * nodes,
            const Point * points )  [inline]
```

6.23.1.5 CFiniteElement() [5/8]

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::CFiniteElement (
            const Shape & shape,
            const ShapeFunction & f ) [inline]
```

6.23.1.6 CFiniteElement() [6/8]

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::CFiniteElement (
            const Shape & shape,
            const ShapeFunction & shfunc,
            const int type ) [inline]
```

6.23.1.7 CFiniteElement() [7/8]

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::CFiniteElement (
            const Shape & shape,
            const ShapeFunction & shfunc,
            const int type,
            const int * neigs ) [inline]
```

6.23.1.8 CFiniteElement() [8/8]

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::CFiniteElement (
            const CFiniteElement< Shape, ShapeFunction > & e ) [inline]
```

6.23.1.9 ∼CFiniteElement()

```
template<class Shape , class ShapeFunction >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::∼CFiniteElement ( ) [inline]
```

6.23.2 Member Function Documentation

6.23.2.1 Clone()

```
template<class Shape , class ShapeFunction >
CElement * corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::Clone ( ) const
[inline], [virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.2 GetDoFs()

```
template<class Shape , class ShapeFunction >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::GetDoFs  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.3 GetGradShapeFunction()

```
template<class Shape , class ShapeFunction >
const Point corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::GetGradShape←
Function (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.4 GetMeasure()

```
template<class Shape , class ShapeFunction >
const double corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::GetMeasure
[virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.5 GetNeighbour()

```
template<class Shape , class ShapeFunction >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::GetNeighbour (
            const int k ) const  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.6 GetNode()

```
template<class Shape , class ShapeFunction >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::GetNode (
            const int k ) const [virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.7 GetNormal()

```
template<class Shape , class ShapeFunction >
const Point corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::GetNormal [virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.8 GetNumberOfNodes()

```
template<class Shape , class ShapeFunction >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::GetNumberOfNodes
[virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.9 GetShape()

```
template<class Shape , class ShapeFunction >
const Shape corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::GetShape
```

6.23.2.10 GetShapeFunction()

```
template<class Shape , class ShapeFunction >
const double corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::GetShape↩
Function (
            const int k,
            const Point & p ) const [virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.11 GetShapeFunctions()

```
template<class Shape , class ShapeFunction >
const ShapeFunction corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::Get↩
ShapeFunctions
```

6.23.2.12 GetType()

```
template<class Shape , class ShapeFunction >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::GetType  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.13 GetWeight()

```
template<class Shape , class ShapeFunction >
const double corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::GetWeight (
            const int node,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.14 IncreaseOrder()

```
template<class Shape , class ShapeFunction >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::IncreaseOrder
[virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.15 Integrate() [1/3]

```
template<class Shape , class ShapeFunction >
const double corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::Integrate (
            const std::function< const double(const Point &)> & f,
            const std::vector< Point > & v ) const  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.16 Integrate() [2/3]

```
template<class Shape , class ShapeFunction >
const Point corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::Integrate (
            const std::function< const Point(const Point &)> & f,
            const std::vector< Point > & v ) const  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.17 Integrate() [3/3]

```
template<class Shape , class ShapeFunction >
const std::vector< double > corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool
>::Integrate (
            const std::function< const std::vector< double >(const Point &)> & f,
            const std::vector< Point > & v ) const  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.18 operator=()

```
template<class Shape , class ShapeFunction >
CFiniteElement & corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::operator= (
            const CFiniteElement< Shape, ShapeFunction, bool, bool > & e )  [inline]
```

6.23.2.19 ReverseNormal()

```
template<class Shape , class ShapeFunction >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::ReverseNormal  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.20 SetNeighbour()

```
template<class Shape , class ShapeFunction >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::SetNeighbour (
            const int k,
            const int elem )  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.21 SetNode()

```
template<class Shape , class ShapeFunction >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::SetNode (
            const int k,
            const int node ) [virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.2.22 SetShape()

```
template<class Shape , class ShapeFunction >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::SetShape (
            const Shape & shape )
```

6.23.2.23 SetShapeFunction()

```
template<class Shape , class ShapeFunction >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::SetShapeFunction (
            const int k,
            const ShapeFunction & func )
```

6.23.2.24 SetType()

```
template<class Shape , class ShapeFunction >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >::SetType (
            const int k ) [virtual]
```

Implements corenc::Mesh::CElement<>.

6.23.3 Friends And Related Function Documentation

6.23.3.1 operator==

```
template<class Shape , class ShapeFunction >
const bool operator== (
            const CFiniteElement< Shape, ShapeFunction, bool, bool > & e1,
            const CFiniteElement< Shape, ShapeFunction, bool, bool > & e2 ) [friend]
```

6.23.3.2 operator>>

```
template<class Shape , class ShapeFunction >
std::istream & operator>> (
            std::istream & is,
            CFiniteElement< Shape, ShapeFunction, bool, bool > & k )  [friend]
```

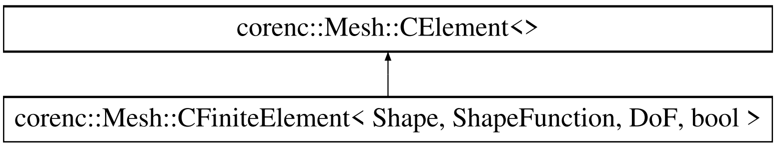
The documentation for this class was generated from the following file:

- CoreNCFEM/FiniteElements/FiniteElement.h

6.24 corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool > Class Template Reference

```
#include <FiniteElement.h>
```

Inheritance diagram for corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >:



Public Member Functions

- CFiniteElement ()
- CFiniteElement (const int ∗nodes, const Point ∗points, const int dofs)
- CFiniteElement (const int ∗nodes, const Point ∗points)
- CFiniteElement (const Shape &shape, const ShapeFunction &f, const DoF &d)
- CFiniteElement (const Shape &shape, const ShapeFunction &shfunc, const DoF &dofs, const int type)
- CFiniteElement (const Shape &shape, const ShapeFunction &shfunc, const DoF &dofs, const int type, const int ∗neigh)
- CFiniteElement (const CFiniteElement< Shape, ShapeFunction, DoF > &e)
- CElement ∗ Clone () const
- ∼CFiniteElement ()
- const int GetType () const
- const int GetNode (const int) const
- const int GetNeighbour (const int) const
- const Shape GetShape () const
- const ShapeFunction GetShapeFunctions () const
- const DoF GetDoF () const
- const int GetDoFs () const
- void SetNeighbour (const int k, const int elem)
- void SetType (const int)
- void SetShapeFunction (const int, const ShapeFunction &)
- void SetDoF (const DoF &)
- void SetShape (const Shape &)
- void SetNode (const int, const int)
- const int GetNumberOfNodes () const

- const int IncreaseOrder ()
- const double GetMeasure () const
- const double Integrate (const std::function< const double(const Point &)> &, const std::vector< Point > &v) const
- const Point Integrate (const std::function< const Point(const Point &)> &, const std::vector< Point > &v) const
- const std::vector< double > Integrate (const std::function< const std::vector< double >(const Point &)> &, const std::vector< Point > &) const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetWeight (const int, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const
- CFiniteElement & operator= (const CFiniteElement &e)

Friends

- const bool operator== (const CFiniteElement &e1, const CFiniteElement &e2)
- std::istream & operator>> (std::istream &is, CFiniteElement &k)

6.24.1 Constructor & Destructor Documentation

6.24.1.1 CFiniteElement() [1/7]

```
template<class Shape , class ShapeFunction , class DoF >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::CFiniteElement ( )  [inline]
```

6.24.1.2 CFiniteElement() [2/7]

```
template<class Shape , class ShapeFunction , class DoF >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::CFiniteElement (
            const int * nodes,
            const Point * points,
            const int dofs )  [inline]
```

6.24.1.3 CFiniteElement() [3/7]

```
template<class Shape , class ShapeFunction , class DoF >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::CFiniteElement (
            const int * nodes,
            const Point * points )  [inline]
```

6.24.1.4 CFiniteElement() [4/7]

```
template<class Shape , class ShapeFunction , class DoF >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::CFiniteElement (
            const Shape & shape,
            const ShapeFunction & f,
            const DoF & d ) [inline]
```

6.24.1.5 CFiniteElement() [5/7]

```
template<class Shape , class ShapeFunction , class DoF >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::CFiniteElement (
            const Shape & shape,
            const ShapeFunction & shfunc,
            const DoF & dofs,
            const int type ) [inline]
```

6.24.1.6 CFiniteElement() [6/7]

```
template<class Shape , class ShapeFunction , class DoF >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::CFiniteElement (
            const Shape & shape,
            const ShapeFunction & shfunc,
            const DoF & dofs,
            const int type,
            const int * neigh ) [inline]
```

6.24.1.7 CFiniteElement() [7/7]

```
template<class Shape , class ShapeFunction , class DoF >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::CFiniteElement (
            const CFiniteElement< Shape, ShapeFunction, DoF > & e ) [inline]
```

6.24.1.8 ∼CFiniteElement()

```
template<class Shape , class ShapeFunction , class DoF >
corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::∼CFiniteElement ( ) [inline]
```

6.24.2 Member Function Documentation

6.24.2.1 Clone()

```
template<class Shape , class ShapeFunction , class DoF >
CElement * corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::Clone ( ) const
[inline], [virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.2 GetDoF()

```
template<class Shape , class ShapeFunction , class DoF >
const DoF corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::GetDoF
```

6.24.2.3 GetDoFs()

```
template<class Shape , class ShapeFunction , class DoF >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::GetDoFs  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.4 GetGradShapeFunction()

```
template<class Shape , class ShapeFunction , class DoF >
const Point corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::GetGradShape←
Function (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.5 GetMeasure()

```
template<class Shape , class ShapeFunction , class DoF >
const double corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::GetMeasure
[virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.6 GetNeighbour()

```
template<class Shape , class ShapeFunction , class DoF >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::GetNeighbour (
            const int k ) const  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.7 GetNode()

```
template<class Shape , class ShapeFunction , class DoF >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::GetNode (
            const int k ) const  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.8 GetNormal()

```
template<class Shape , class ShapeFunction , class DoF >
const Point corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::GetNormal  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.9 GetNumberOfNodes()

```
template<class Shape , class ShapeFunction , class DoF >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::GetNumberOfNodes
[virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.10 GetShape()

```
template<class Shape , class ShapeFunction , class DoF >
const Shape corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::GetShape
```

6.24.2.11 GetShapeFunction()

```
template<class Shape , class ShapeFunction , class DoF >
const double corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::GetShapeFunction
(
            const int ,
            const Point &  ) const  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.12 GetShapeFunctions()

```
template<class Shape , class ShapeFunction , class DoF >
const ShapeFunction corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::Get←
ShapeFunctions
```

6.24.2.13 GetType()

```
template<class Shape , class ShapeFunction , class DoF >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::GetType  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.14 GetWeight()

```
template<class Shape , class ShapeFunction , class DoF >
const double corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::GetWeight (
            const int ,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.15 IncreaseOrder()

```
template<class Shape , class ShapeFunction , class DoF >
const int corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::IncreaseOrder
[virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.16 Integrate() [1/3]

```
template<class Shape , class ShapeFunction , class DoF >
const double corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::Integrate (
            const std::function< const double(const Point &)> & f,
            const std::vector< Point > & v ) const  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.17 Integrate() [2/3]

```
template<class Shape , class ShapeFunction , class DoF >
const Point corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::Integrate (
            const std::function< const Point(const Point &)> & f,
            const std::vector< Point > & v ) const  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.18 Integrate() [3/3]

```
template<class Shape , class ShapeFunction , class DoF >
const std::vector< double > corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >↩
::Integrate (
            const std::function< const std::vector< double >(const Point &)> & f,
            const std::vector< Point > & v ) const  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.19 operator=()

```
template<class Shape , class ShapeFunction , class DoF >
CFiniteElement & corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::operator= (
            const CFiniteElement< Shape, ShapeFunction, DoF, bool > & e )  [inline]
```

6.24.2.20 ReverseNormal()

```
template<class Shape , class ShapeFunction , class DoF >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::ReverseNormal  [virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.21 SetDoF()

```
template<class Shape , class ShapeFunction , class DoF >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::SetDoF (
            const DoF & dof )
```

6.24.2.22 SetNeighbour()

```
template<class Shape , class ShapeFunction , class DoF >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::SetNeighbour (
            const int k,
            const int elem ) [virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.23 SetNode()

```
template<class Shape , class ShapeFunction , class DoF >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::SetNode (
            const int k,
            const int node ) [virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.2.24 SetShape()

```
template<class Shape , class ShapeFunction , class DoF >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::SetShape (
            const Shape & shape )
```

6.24.2.25 SetShapeFunction()

```
template<class Shape , class ShapeFunction , class DoF >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::SetShapeFunction (
            const int k,
            const ShapeFunction & func )
```

6.24.2.26 SetType()

```
template<class Shape , class ShapeFunction , class DoF >
void corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >::SetType (
            const int k ) [virtual]
```

Implements corenc::Mesh::CElement<>.

6.24.3 Friends And Related Function Documentation

6.24.3.1 operator==

```
template<class Shape , class ShapeFunction , class DoF >
const bool operator== (
            const CFiniteElement< Shape, ShapeFunction, DoF, bool > & e1,
            const CFiniteElement< Shape, ShapeFunction, DoF, bool > & e2 ) [friend]
```

6.24.3.2 operator>>

```
template<class Shape , class ShapeFunction , class DoF >
std::istream & operator>> (
            std::istream & is,
            CFiniteElement< Shape, ShapeFunction, DoF, bool > & k ) [friend]
```

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

- CoreNCFEM/FiniteElements/FiniteElement.h

6.25 corenc::CFiniteSolver< Method, Mesh, Solver > Class Template Reference

```
#include <FiniteSolver.h>
```

Public Member Functions

- CFiniteSolver ()
- ~CFiniteSolver ()
- void Solve ()

6.25.1 Constructor & Destructor Documentation

6.25.1.1 CFiniteSolver()

```
template<class Method , class Mesh , class Solver >
corenc::CFiniteSolver< Method, Mesh, Solver >::CFiniteSolver ( ) [inline]
```

6.25.1.2 ∼CFiniteSolver()

```
template<class Method , class Mesh , class Solver >
corenc::CFiniteSolver< Method, Mesh, Solver >::∼CFiniteSolver ( ) [inline]
```

6.25.2 Member Function Documentation

6.25.2.1 Solve()

```
template<class Method , class Mesh , class Solver >
void corenc::CFiniteSolver< Method, Mesh, Solver >::Solve
```

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

- CoreNCFEM/FiniteSolver.h

6.26 corenc::Mesh::CMesh< T > Class Template Reference

```
#include <Mesh.h>
```

Public Member Functions

- CMesh ()
- virtual ∼CMesh ()
- virtual const unsigned int GetNumberOfNodes () const =0
- virtual const unsigned int GetNumberOfElements () const =0
- virtual const int FindElement (const Point &) const =0
- virtual const unsigned int GetNumberOfBoundaries () const =0
- virtual const CElement< T > ∗ GetElement (const unsigned int) const =0
- virtual const CElement< T > ∗ GetBoundary (const unsigned int) const =0
- virtual const Point GetNode (const unsigned int) const =0
- virtual const double getSolution (const unsigned int element, const unsigned int node) const =0
- virtual const int updateSolution (const unsigned int element, const unsigned int node, const double value)=0
- virtual const std::vector< double > getSolution () const =0
- virtual const int updateSolution (const std::vector< double > &)=0
- virtual const int updateSolution (const unsigned int element, const unsigned int node, CSolution ∗value)=0
- virtual const double getParameter (Parameters, const unsigned int, const Point &) const =0
- virtual const double getParameter (Parameters, const unsigned int, const int) const =0
- virtual const int setParameter (Parameters, const double, const unsigned int)=0
- virtual const double getMinSize () const =0
- virtual const int updateSolution (const unsigned int node, const double value)=0

6.26.1 Constructor & Destructor Documentation

6.26.1.1 CMesh()

```
template<class T >
corenc::Mesh::CMesh< T >::CMesh ( )  [inline]
```

6.26.1.2 ~CMesh()

```
template<class T >
virtual corenc::Mesh::CMesh< T >::~CMesh ( )  [inline], [virtual]
```

6.26.2 Member Function Documentation

6.26.2.1 FindElement()

```
template<class T >
virtual const int corenc::Mesh::CMesh< T >::FindElement (
            const Point & ) const  [pure virtual]
```

Implemented in corenc::Mesh::CMesh1D, corenc::Mesh::CTriangularMesh, and corenc::Mesh::CTriangularMeshLinear.

6.26.2.2 GetBoundary()

```
template<class T >
virtual const CElement< T > * corenc::Mesh::CMesh< T >::GetBoundary (
            const unsigned int ) const  [pure virtual]
```

Implemented in corenc::Mesh::CMesh1D, corenc::Mesh::CTriangularMesh, and corenc::Mesh::CTriangularMeshLinear.

6.26.2.3 GetElement()

```
template<class T >
virtual const CElement< T > * corenc::Mesh::CMesh< T >::GetElement (
            const unsigned int ) const  [pure virtual]
```

Implemented in corenc::Mesh::CMesh1D, corenc::Mesh::CTriangularMesh, and corenc::Mesh::CTriangularMeshLinear.

6.26.2.4 getMinSize()

```
template<class T >
virtual const double corenc::Mesh::CMesh< T >::getMinSize ( ) const  [pure virtual]
```

Implemented in corenc::Mesh::CMesh1D, corenc::Mesh::CTriangularMesh, and corenc::Mesh::CTriangularMeshLinear.

6.26.2.5 GetNode()

```
template<class T >
virtual const Point corenc::Mesh::CMesh< T >::GetNode (
            const unsigned int  ) const  [pure virtual]
```

Implemented in corenc::Mesh::CMesh1D, corenc::Mesh::CTriangularMesh, and corenc::Mesh::CTriangularMeshLinear.

6.26.2.6 GetNumberOfBoundaries()

```
template<class T >
virtual const unsigned int corenc::Mesh::CMesh< T >::GetNumberOfBoundaries ( ) const  [pure
virtual]
```

Implemented in corenc::Mesh::CMesh1D, corenc::Mesh::CTriangularMesh, and corenc::Mesh::CTriangularMeshLinear.

6.26.2.7 GetNumberOfElements()

```
template<class T >
virtual const unsigned int corenc::Mesh::CMesh< T >::GetNumberOfElements ( ) const  [pure
virtual]
```

Implemented in corenc::Mesh::CMesh1D, corenc::Mesh::CTriangularMesh, and corenc::Mesh::CTriangularMeshLinear.

6.26.2.8 GetNumberOfNodes()

```
template<class T >
virtual const unsigned int corenc::Mesh::CMesh< T >::GetNumberOfNodes ( ) const  [pure virtual]
```

Implemented in corenc::Mesh::CMesh1D, corenc::Mesh::CTriangularMesh, and corenc::Mesh::CTriangularMeshLinear.

6.26.2.9 getParameter() [1/2]

```
template<class T >
virtual const double corenc::Mesh::CMesh< T >::getParameter (
            Parameters ,
            const unsigned int ,
            const int  ) const  [pure virtual]
```

Implemented in corenc::Mesh::CMesh1D, corenc::Mesh::CTriangularMesh, and corenc::Mesh::CTriangularMeshLinear.

6.26.2.10 getParameter() [2/2]

```
template<class T >
virtual const double corenc::Mesh::CMesh< T >::getParameter (
            Parameters ,
            const unsigned int ,
            const Point &  ) const  [pure virtual]
```

Implemented in corenc::Mesh::CTriangularMesh, corenc::Mesh::CTriangularMeshLinear, and corenc::Mesh::CMesh1D.

6.26.2.11 getSolution() [1/2]

```
template<class T >
virtual const std::vector< double > corenc::Mesh::CMesh< T >::getSolution ( ) const  [pure
virtual]
```

Implemented in corenc::Mesh::CMesh1D, corenc::Mesh::CTriangularMesh, and corenc::Mesh::CTriangularMeshLinear.

6.26.2.12 getSolution() [2/2]

```
template<class T >
virtual const double corenc::Mesh::CMesh< T >::getSolution (
            const unsigned int element,
            const unsigned int node ) const  [pure virtual]
```

Implemented in corenc::Mesh::CMesh1D, corenc::Mesh::CTriangularMesh, and corenc::Mesh::CTriangularMeshLinear.

6.26.2.13 setParameter()

```
template<class T >
virtual const int corenc::Mesh::CMesh< T >::setParameter (
            Parameters ,
            const double ,
            const unsigned int  )  [pure virtual]
```

Implemented in corenc::Mesh::CMesh1D, corenc::Mesh::CTriangularMesh, and corenc::Mesh::CTriangularMeshLinear.

6.26.2.14 updateSolution() [1/4]

```
template<class T >
virtual const int corenc::Mesh::CMesh< T >::updateSolution (
            const std::vector< double > & ) [pure virtual]
```

Implemented in corenc::Mesh::CTriangularMesh, corenc::Mesh::CTriangularMeshLinear, and corenc::Mesh::CMesh1D.

6.26.2.15 updateSolution() [2/4]

```
template<class T >
virtual const int corenc::Mesh::CMesh< T >::updateSolution (
            const unsigned int element,
            const unsigned int node,
            const double value ) [pure virtual]
```

Implemented in corenc::Mesh::CMesh1D, corenc::Mesh::CTriangularMesh, and corenc::Mesh::CTriangularMeshLinear.

6.26.2.16 updateSolution() [3/4]

```
template<class T >
virtual const int corenc::Mesh::CMesh< T >::updateSolution (
            const unsigned int element,
            const unsigned int node,
            CSolution * value ) [pure virtual]
```

Implemented in corenc::Mesh::CMesh1D, corenc::Mesh::CTriangularMesh, and corenc::Mesh::CTriangularMeshLinear.

6.26.2.17 updateSolution() [4/4]

```
template<class T >
virtual const int corenc::Mesh::CMesh< T >::updateSolution (
            const unsigned int node,
            const double value ) [pure virtual]
```

Implemented in corenc::Mesh::CMesh1D, corenc::Mesh::CTriangularMesh, and corenc::Mesh::CTriangularMeshLinear.

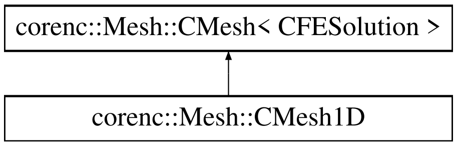
The documentation for this class was generated from the following file:

- CoreNCFEM/Mesh.h

6.27 corenc::Mesh::CMesh1D Class Reference

#include <Mesh1D.h>

Inheritance diagram for corenc::Mesh::CMesh1D:



Public Member Functions

- CMesh1D ()
- CMesh1D (const std::string &domain_name)
- CMesh1D (const std::string &domain_file, const std::string &init_file)
- CMesh1D (const double x0, const double x1, const unsigned n, const int order, const std::function< const double(const Point &)> &init_func)
- CMesh1D (const double x0, const double x1, const unsigned n, const int order, const std::function< const double(const Point &)> &init_func, const std::function< const double(const Point &)> &init_derivative)
- CMesh1D (const CMesh1D &)
- CMesh1D & operator= (const CMesh1D &m)
- const unsigned int GetNumberOfElements () const
- const unsigned int GetNumberOfNodes () const
- const unsigned int GetNumberOfBoundaries () const
- const int FindElement (const Point &) const
- const Point GetNode (const unsigned int) const
- const CElement< CFESolution > * GetElement (const unsigned int) const
- const CElement< CFESolution > * GetBoundary (const unsigned int) const
- const double getSolution (const unsigned int element, const unsigned int node) const
- const double getParameter (Parameters, const unsigned int, const Point &p) const
- const double getParameter (Parameters, const unsigned int, const int) const
- const std::vector< double > getSolution () const
- const int updateSolution (const std::vector< double > &new_solution)
- const int updateSolution (const unsigned int element, const unsigned int node, const double value)
- const int updateSolution (const unsigned int element, const unsigned int node, CSolution *value)
- const int updateSolution (const unsigned int node, const double value)
- const int setParameter (Parameters, const double, const unsigned int)
- const double getMinSize () const
- ~CMesh1D ()
- auto GetElements () -> decltype(m_elems)
- auto GetBoundary () -> decltype(m_bnds)

6.27.1 Constructor & Destructor Documentation

6.27.1.1 CMesh1D() [1/6]

```
CMesh1D::CMesh1D ( )
```

6.27.1.2 CMesh1D() [2/6]

```
CMesh1D::CMesh1D (
        const std::string & domain_name )
```

6.27.1.3 CMesh1D() [3/6]

```
CMesh1D::CMesh1D (
        const std::string & domain_file,
        const std::string & init_file )
```

6.27.1.4 CMesh1D() [4/6]

```
CMesh1D::CMesh1D (
        const double x0,
        const double x1,
        const unsigned n,
        const int order,
        const std::function< const double(const Point &)> & init_func )
```

6.27.1.5 CMesh1D() [5/6]

```
CMesh1D::CMesh1D (
        const double x0,
        const double x1,
        const unsigned n,
        const int order,
        const std::function< const double(const Point &)> & init_func,
        const std::function< const double(const Point &)> & init_derivative )
```

6.27.1.6 CMesh1D() [6/6]

```
CMesh1D::CMesh1D (
        const CMesh1D & m )
```

6.27.1.7 ~CMesh1D()

```
CMesh1D::~CMesh1D ( )
```

6.27.2 Member Function Documentation

6.27.2.1 FindElement()

```
const int CMesh1D::FindElement (
            const Point & test ) const  [virtual]
```

Implements corenc::Mesh::CMesh< CFESolution >.

6.27.2.2 GetBoundary() [1/2]

```
auto corenc::Mesh::CMesh1D::GetBoundary ( ) -> decltype(m_bnds)   [inline]
```

6.27.2.3 GetBoundary() [2/2]

```
const CElement< CFESolution > * CMesh1D::GetBoundary (
            const unsigned int n ) const  [virtual]
```

Implements corenc::Mesh::CMesh< CFESolution >.

6.27.2.4 GetElement()

```
const CElement< CFESolution > * CMesh1D::GetElement (
            const unsigned int n ) const  [virtual]
```

Implements corenc::Mesh::CMesh< CFESolution >.

6.27.2.5 GetElements()

```
auto corenc::Mesh::CMesh1D::GetElements ( ) -> decltype(m_elems)   [inline]
```

6.27.2.6 getMinSize()

```
const double corenc::Mesh::CMesh1D::getMinSize ( ) const  [inline], [virtual]
```

Implements corenc::Mesh::CMesh< CFESolution >.

6.27.2.7 GetNode()

```
const Point CMesh1D::GetNode (
            const unsigned int n ) const  [virtual]
```

Implements corenc::Mesh::CMesh< CFESolution >.

6.27.2.8 GetNumberOfBoundaries()

```
const unsigned int CMesh1D::GetNumberOfBoundaries ( ) const  [virtual]
```

Implements corenc::Mesh::CMesh< CFESolution >.

6.27.2.9 GetNumberOfElements()

```
const unsigned int CMesh1D::GetNumberOfElements ( ) const  [virtual]
```

Implements corenc::Mesh::CMesh< CFESolution >.

6.27.2.10 GetNumberOfNodes()

```
const unsigned int CMesh1D::GetNumberOfNodes ( ) const  [virtual]
```

Implements corenc::Mesh::CMesh< CFESolution >.

6.27.2.11 getParameter() [1/2]

```
const double CMesh1D::getParameter (
            Parameters param,
            const unsigned int ,
            const int  ) const  [virtual]
```

Implements corenc::Mesh::CMesh< CFESolution >.

6.27.2.12 getParameter() [2/2]

```
const double CMesh1D::getParameter (
            Parameters param,
            const unsigned int ,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CMesh< CFESolution >.

6.27.2.13 getSolution() [1/2]

```
const std::vector< double > corenc::Mesh::CMesh1D::getSolution ( ) const  [inline], [virtual]
```

Implements corenc::Mesh::CMesh< CFESolution >.

6.27.2.14 getSolution() [2/2]

```
const double CMesh1D::getSolution (
            const unsigned int element,
            const unsigned int node ) const  [virtual]
```

Implements corenc::Mesh::CMesh< CFESolution >.

6.27.2.15 operator=()

```
CMesh1D & corenc::Mesh::CMesh1D::operator= (
            const CMesh1D & m )  [inline]
```

6.27.2.16 setParameter()

```
const int CMesh1D::setParameter (
            Parameters param,
            const double value,
            const unsigned int  )  [virtual]
```

Implements corenc::Mesh::CMesh< CFESolution >.

6.27.2.17 updateSolution() [1/4]

```
const int corenc::Mesh::CMesh1D::updateSolution (
            const std::vector< double > & new_solution )  [virtual]
```

Implements corenc::Mesh::CMesh< CFESolution >.

6.27.2.18 updateSolution() [2/4]

```
const int CMesh1D::updateSolution (
            const unsigned int element,
            const unsigned int node,
            const double value )  [virtual]
```

Implements corenc::Mesh::CMesh< CFESolution >.

6.27.2.19 updateSolution() [3/4]

```
const int CMesh1D::updateSolution (
            const unsigned int element,
            const unsigned int node,
            CSolution * value )  [virtual]
```

Implements corenc::Mesh::CMesh< CFESolution >.

6.27.2.20 updateSolution() [4/4]

```
const int CMesh1D::updateSolution (
            const unsigned int node,
            const double value )  [virtual]
```

Implements corenc::Mesh::CMesh< CFESolution >.

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

- CoreNCFEM/Grids/Mesh1D.h
- CoreNCFEM/Grids/Mesh1D.cpp

6.28 corenc::Mesh::CMesh< bool > Class Reference

```
#include <Mesh.h>
```

Public Member Functions

- CMesh ()
- virtual ~CMesh ()
- virtual const unsigned int GetNumberOfNodes () const =0
- virtual const unsigned int GetNumberOfElements () const =0
- virtual const int FindElement (const Point &) const =0
- virtual const unsigned int GetNumberOfBoundaries () const =0
- virtual const CElement * GetElement (const unsigned int) const =0
- virtual const CElement * GetBoundary (const unsigned int) const =0
- virtual const Point GetNode (const unsigned int) const =0
- virtual const double getSolution (const unsigned int element, const unsigned int node) const =0
- virtual const int updateSolution (const unsigned int element, const unsigned int node, const double value)=0
- virtual const std::vector< double > getSolution () const =0
- virtual const int updateSolution (const std::vector< double > &)=0
- virtual const int updateSolution (const unsigned int element, const unsigned int node, CSolution *value)=0
- virtual const double getParameter (Parameters, const unsigned int, const Point &p) const =0
- virtual const double getParameter (Parameters, const unsigned int, const int) const =0
- virtual const int setParameter (Parameters, const double, const unsigned int)=0
- virtual const double getMinSize () const =0

6.28.1 Constructor & Destructor Documentation

6.28.1.1 CMesh()

```
corenc::Mesh::CMesh< bool >::CMesh ( )  [inline]
```

6.28.1.2 ~CMesh()

```
virtual corenc::Mesh::CMesh< bool >::~CMesh ( )  [inline], [virtual]
```

6.28.2 Member Function Documentation

6.28.2.1 FindElement()

```
virtual const int corenc::Mesh::CMesh< bool >::FindElement (
            const Point &  ) const  [pure virtual]
```

6.28.2.2 GetBoundary()

virtual const CElement * corenc::Mesh::CMesh< bool >::GetBoundary (
const unsigned int) const [pure virtual]

6.28.2.3 GetElement()

virtual const CElement * corenc::Mesh::CMesh< bool >::GetElement (
const unsigned int) const [pure virtual]

6.28.2.4 getMinSize()

virtual const double corenc::Mesh::CMesh< bool >::getMinSize ( ) const [pure virtual]

6.28.2.5 GetNode()

virtual const Point corenc::Mesh::CMesh< bool >::GetNode (
const unsigned int) const [pure virtual]

6.28.2.6 GetNumberOfBoundaries()

virtual const unsigned int corenc::Mesh::CMesh< bool >::GetNumberOfBoundaries ( ) const [pure virtual]

6.28.2.7 GetNumberOfElements()

virtual const unsigned int corenc::Mesh::CMesh< bool >::GetNumberOfElements ( ) const [pure virtual]

6.28.2.8 GetNumberOfNodes()

virtual const unsigned int corenc::Mesh::CMesh< bool >::GetNumberOfNodes ( ) const [pure virtual]

6.28.2.9 getParameter() [1/2]

```
virtual const double corenc::Mesh::CMesh< bool >::getParameter (
            Parameters ,
            const unsigned int ,
            const int ) const [pure virtual]
```

6.28.2.10 getParameter() [2/2]

```
virtual const double corenc::Mesh::CMesh< bool >::getParameter (
            Parameters ,
            const unsigned int ,
            const Point & p ) const [pure virtual]
```

6.28.2.11 getSolution() [1/2]

```
virtual const std::vector< double > corenc::Mesh::CMesh< bool >::getSolution ( ) const [pure
virtual]
```

6.28.2.12 getSolution() [2/2]

```
virtual const double corenc::Mesh::CMesh< bool >::getSolution (
            const unsigned int element,
            const unsigned int node ) const [pure virtual]
```

6.28.2.13 setParameter()

```
virtual const int corenc::Mesh::CMesh< bool >::setParameter (
            Parameters ,
            const double ,
            const unsigned int ) [pure virtual]
```

6.28.2.14 updateSolution() [1/3]

```
virtual const int corenc::Mesh::CMesh< bool >::updateSolution (
            const std::vector< double > & ) [pure virtual]
```

6.28.2.15 updateSolution() [2/3]

```
virtual const int corenc::Mesh::CMesh< bool >::updateSolution (
            const unsigned int element,
            const unsigned int node,
            const double value ) [pure virtual]
```

6.28.2.16 updateSolution() [3/3]

```
virtual const int corenc::Mesh::CMesh< bool >::updateSolution (
            const unsigned int element,
            const unsigned int node,
            CSolution * value ) [pure virtual]
```

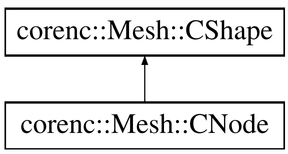
The documentation for this class was generated from the following file:

- CoreNCFEM/Mesh.h

6.29 corenc::Mesh::CNode Class Reference

```
#include <Node.h>
```

Inheritance diagram for corenc::Mesh::CNode:



Public Member Functions

- CNode ()
- CNode (const CNode &)
- CNode (const int n)
- CNode (const int ∗n)
- CNode & operator= (const CNode &e)
- ∼CNode ()
- const int GetNode (const int) const
- const int GetNode (const NODES &) const
- const int IncreaseOrder ()
- const int GetNumberOfNodes () const
- void SetNode (const int k, const int node)
- const double Integrate (const std::function< const double(const Point &)> &, const std::vector< Point > &v) const
- const Point Integrate (const std::function< const Point(const Point &)> &, const std::vector< Point > &v) const
- const std::vector< double > Integrate (const std::function< const std::vector< double >(const Point &)> &, const std::vector< Point > &) const

Friends

- const bool operator== (const CNode &e1, const CNode &e2)
- std::istream & operator>> (std::istream &is, CNode &e)

6.29.1 Constructor & Destructor Documentation

6.29.1.1 CNode() [1/4]

```
CNode::CNode ( )
```

6.29.1.2 CNode() [2/4]

```
CNode::CNode (
            const CNode & n )
```

6.29.1.3 CNode() [3/4]

```
corenc::Mesh::CNode::CNode (
            const int n )
```

6.29.1.4 CNode() [4/4]

```
CNode::CNode (
            const int * n )
```

6.29.1.5 ∼CNode()

```
corenc::Mesh::CNode::∼CNode ( ) [inline]
```

6.29.2 Member Function Documentation

6.29.2.1 GetNode() [1/2]

```
const int CNode::GetNode (
            const int n ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.29.2.2 GetNode() [2/2]

```
const int CNode::GetNode (
            const NODES & node ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.29.2.3 GetNumberOfNodes()

```
const int CNode::GetNumberOfNodes ( ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.29.2.4 IncreaseOrder()

```
const int corenc::Mesh::CNode::IncreaseOrder ( )  [inline]
```

6.29.2.5 Integrate() [1/3]

```
const double CNode::Integrate (
            const std::function< const double(const Point &)> & f,
            const std::vector< Point > & v ) const
```

6.29.2.6 Integrate() [2/3]

```
const Point CNode::Integrate (
            const std::function< const Point(const Point &)> & f,
            const std::vector< Point > & v ) const
```

6.29.2.7 Integrate() [3/3]

```
const std::vector< double > corenc::Mesh::CNode::Integrate (
            const std::function< const std::vector< double >(const Point &)> & ,
            const std::vector< Point > & ) const [virtual]
```

Implements corenc::Mesh::CShape.

6.29.2.8 operator=()

```
CNode & corenc::Mesh::CNode::operator= (
            const CNode & e ) [inline]
```

6.29.2.9 SetNode()

```
void CNode::SetNode (
            const int k,
            const int node ) [virtual]
```

Implements corenc::Mesh::CShape.

6.29.3 Friends And Related Function Documentation

6.29.3.1 operator==

```
const bool operator== (
            const CNode & e1,
            const CNode & e2 ) [friend]
```

6.29.3.2 operator>>

```
std::istream & operator>> (
            std::istream & is,
            CNode & e ) [friend]
```

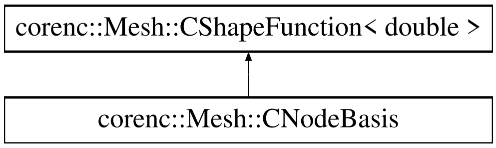
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/Node.h
- CoreNCFEM/FiniteElements/Node.cpp

6.30 corenc::Mesh::CNodeBasis Class Reference

```
#include <Node.h>
```

Inheritance diagram for corenc::Mesh::CNodeBasis:



Public Member Functions

- CNodeBasis ()
- CNodeBasis (const Point ∗)
- CNodeBasis (const CNodeBasis &e)
- CNodeBasis & operator= (const CNodeBasis &e)
- ∼CNodeBasis ()
- const int GetNumberOfShapeFunctions () const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetWeight (const int node, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const
- const int IncreaseOrder ()
- const double GetMeasure () const

6.30.1 Constructor & Destructor Documentation

6.30.1.1 CNodeBasis() [1/3]

```
CNodeBasis::CNodeBasis ( )
```

6.30.1.2 CNodeBasis() [2/3]

```
CNodeBasis::CNodeBasis (
            const Point * p )
```

6.30.1.3 CNodeBasis() [3/3]

```
corenc::Mesh::CNodeBasis::CNodeBasis (
            const CNodeBasis & e )  [inline]
```

6.30.1.4 ∼CNodeBasis()

```
corenc::Mesh::CNodeBasis::∼CNodeBasis ( )  [inline]
```

6.30.2 Member Function Documentation

6.30.2.1 GetGradShapeFunction()

```
const Point CNodeBasis::GetGradShapeFunction (
            const int ,
            const Point &  ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.30.2.2 GetMeasure()

```
const double corenc::Mesh::CNodeBasis::GetMeasure ( ) const  [inline], [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.30.2.3 GetNormal()

```
const Point CNodeBasis::GetNormal ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.30.2.4 GetNumberOfShapeFunctions()

```
const int CNodeBasis::GetNumberOfShapeFunctions ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.30.2.5 GetShapeFunction()

```
const double CNodeBasis::GetShapeFunction (
            const int ,
            const Point & ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.30.2.6 GetWeight()

```
const double corenc::Mesh::CNodeBasis::GetWeight (
            const int node,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const [inline]
```

6.30.2.7 IncreaseOrder()

```
const int corenc::Mesh::CNodeBasis::IncreaseOrder ( ) [inline]
```

6.30.2.8 operator=()

```
CNodeBasis & corenc::Mesh::CNodeBasis::operator= (
            const CNodeBasis & e ) [inline]
```

6.30.2.9 ReverseNormal()

```
void CNodeBasis::ReverseNormal ( ) [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

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

- CoreNCFEM/FiniteElements/Node.h
- CoreNCFEM/FiniteElements/Node.cpp

6.31 corenc::Mesh::CParameter Class Reference

```
#include <Parameter.h>
```

Public Member Functions

- CParameter ()
- CParameter (const parameter< double > &_diff, const parameter< double > &_adv, const parameter< double > &_mass)
- CParameter (const Parameters &, const parameter< double > &)
- ~CParameter ()
- const double GetDiffusion () const
- const double GetAdvection () const
- const double GetMass () const
- const double GetDiffusion (const Point &) const
- const double GetAdvection (const Point &) const
- const double GetMass (const Point &) const

6.31.1 Constructor & Destructor Documentation

6.31.1.1 CParameter() [1/3]

```
CParameter::CParameter ( )
```

6.31.1.2 CParameter() [2/3]

```
CParameter::CParameter (
            const parameter< double > & _diff,
            const parameter< double > & _adv,
            const parameter< double > & _mass )
```

6.31.1.3 CParameter() [3/3]

```
CParameter::CParameter (
            const Parameters & type,
            const parameter< double > & p )
```

6.31.1.4 ~CParameter()

```
CParameter::~CParameter ( )
```

6.31.2 Member Function Documentation

6.31.2.1 GetAdvection() [1/2]

```
const double CParameter::GetAdvection ( ) const
```

6.31.2.2 GetAdvection() [2/2]

```
const double CParameter::GetAdvection (
             const Point & p ) const
```

6.31.2.3 GetDiffusion() [1/2]

```
const double CParameter::GetDiffusion ( ) const
```

6.31.2.4 GetDiffusion() [2/2]

```
const double CParameter::GetDiffusion (
             const Point & p ) const
```

6.31.2.5 GetMass() [1/2]

```
const double CParameter::GetMass ( ) const
```

6.31.2.6 GetMass() [2/2]

```
const double CParameter::GetMass (
             const Point & p ) const
```

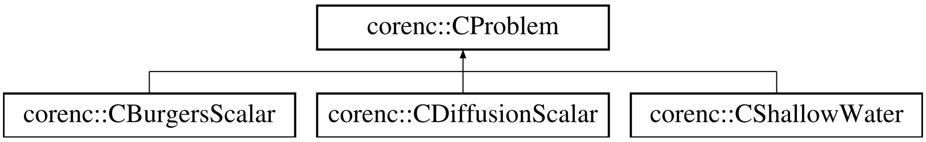
The documentation for this class was generated from the following files:

- CoreNCFEM/Parameter.h
- CoreNCFEM/Parameter.cpp

6.32 corenc::CProblem Class Reference

#include <Problems.h>

Inheritance diagram for corenc::CProblem:



Public Member Functions

- CProblem ()
- virtual ~CProblem ()
- virtual Terms getTerm (const unsigned int) const =0
- virtual const unsigned int getNumberOfTerms () const =0
- virtual const int setTerm (const unsigned int, const Terms &)=0
- virtual const int addTerm (const Terms &)=0
- virtual const int load_parameters (const std::string &file_name)=0

6.32.1 Constructor & Destructor Documentation

6.32.1.1 CProblem()

```
corenc::CProblem::CProblem ( )  [inline]
```

6.32.1.2 ~CProblem()

```
virtual corenc::CProblem::~CProblem ( )  [inline], [virtual]
```

6.32.2 Member Function Documentation

6.32.2.1 addTerm()

```
virtual const int corenc::CProblem::addTerm (
             const Terms & )  [pure virtual]
```

Implemented in corenc::CBurgersScalar, corenc::CDiffusionScalar, and corenc::CShallowWater.

6.32.2.2 getNumberOfTerms()

```
virtual const unsigned int corenc::CProblem::getNumberOfTerms ( ) const  [pure virtual]
```

Implemented in corenc::CBurgersScalar, corenc::CDiffusionScalar, and corenc::CShallowWater.

6.32.2.3 getTerm()

```
virtual Terms corenc::CProblem::getTerm (
            const unsigned int  ) const  [pure virtual]
```

Implemented in corenc::CBurgersScalar, corenc::CDiffusionScalar, and corenc::CShallowWater.

6.32.2.4 load_parameters()

```
virtual const int corenc::CProblem::load_parameters (
            const std::string & file_name )  [pure virtual]
```

Implemented in corenc::CBurgersScalar, corenc::CDiffusionScalar, and corenc::CShallowWater.

6.32.2.5 setTerm()

```
virtual const int corenc::CProblem::setTerm (
            const unsigned int ,
            const Terms &  )  [pure virtual]
```

Implemented in corenc::CBurgersScalar, corenc::CDiffusionScalar, and corenc::CShallowWater.

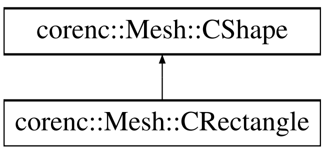
The documentation for this class was generated from the following file:

- Problems/Problems.h

6.33 corenc::Mesh::CRectangle Class Reference

```
#include <Rectangle.h>
```

Inheritance diagram for corenc::Mesh::CRectangle:



Public Member Functions

- CRectangle ()
- CRectangle (const int n1, const int n2, const int n3, const int n4, const int order)
- CRectangle (const int n1, const int n2, const int n3, const int n4, const int e1, const int e2, const int e3, const int e4, const int order)
- CRectangle (const int ∗, const int order)
- CRectangle (const int ∗, const int ∗, const int order)
- CRectangle (const CRectangle &)
- CRectangle & operator= (const CRectangle &t)
- const bool operator== (const CRectangle &t)
- std::istream & operator>> (std::istream &is)
- ∼CRectangle ()
- const int GetNode (const int) const
- const int GetNode (const NODES &) const
- const int GetEdge (const int) const
- const int GetFacet (const int) const
- const int GetNumberOfNodes () const
- const int GetNumberOfEdges () const
- const int GetNumberOfFacets () const
- const double Integrate (const std::function< const double(const Point &)> &, const std::vector< Point > &v) const
- const Point Integrate (const std::function< const Point(const Point &)> &, const std::vector< Point > &v) const
- const std::vector< double > Integrate (const std::function< const std::vector< double >(const Point &)> &, const std::vector< Point > &) const
- void SetNode (const int k, const int node)
- const int IncreaseOrder ()
- const int SetOrder (const int px, const int py)
- void SetEdge (const int k, const int edge)
- void SetFacet (const int k, const int facet)

6.33.1 Constructor & Destructor Documentation

6.33.1.1 CRectangle() [1/6]

```
CRectangle::CRectangle ( )
```

6.33.1.2 CRectangle() [2/6]

```
CRectangle::CRectangle (
            const int n1,
            const int n2,
            const int n3,
            const int n4,
            const int order )
```

6.33.1.3 CRectangle() [3/6]

```
CRectangle::CRectangle (
            const int n1,
            const int n2,
            const int n3,
            const int n4,
            const int e1,
            const int e2,
            const int e3,
            const int e4,
            const int order )
```

6.33.1.4 CRectangle() [4/6]

```
CRectangle::CRectangle (
            const int * nodes,
            const int order )
```

6.33.1.5 CRectangle() [5/6]

```
CRectangle::CRectangle (
            const int * nodes,
            const int * edges,
            const int order )
```

6.33.1.6 CRectangle() [6/6]

```
CRectangle::CRectangle (
            const CRectangle & t )
```

6.33.1.7 ∼CRectangle()

```
corenc::Mesh::CRectangle::∼CRectangle ( ) [inline]
```

6.33.2 Member Function Documentation

6.33.2.1 GetEdge()

```
const int CRectangle::GetEdge (
            const int n ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.33.2.2 GetFacet()

```
const int CRectangle::GetFacet (
            const int  ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.33.2.3 GetNode() [1/2]

```
const int CRectangle::GetNode (
            const int n ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.33.2.4 GetNode() [2/2]

```
const int CRectangle::GetNode (
            const NODES & node ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.33.2.5 GetNumberOfEdges()

```
const int CRectangle::GetNumberOfEdges ( ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.33.2.6 GetNumberOfFacets()

```
const int CRectangle::GetNumberOfFacets ( ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.33.2.7 GetNumberOfNodes()

```
const int CRectangle::GetNumberOfNodes ( ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.33.2.8 IncreaseOrder()

```
const int CRectangle::IncreaseOrder ( )
```

6.33.2.9 Integrate() [1/3]

```
const double CRectangle::Integrate (
            const std::function< const double(const Point &)> & f,
            const std::vector< Point > & v ) const
```

6.33.2.10 Integrate() [2/3]

```
const Point CRectangle::Integrate (
            const std::function< const Point(const Point &)> & f,
            const std::vector< Point > & v ) const
```

6.33.2.11 Integrate() [3/3]

```
const vector< double > CRectangle::Integrate (
            const std::function< const std::vector< double >(const Point &)> & f,
            const std::vector< Point > & v ) const  [virtual]
```

Implements corenc::Mesh::CShape.

6.33.2.12 operator=()

```
CRectangle & corenc::Mesh::CRectangle::operator= (
            const CRectangle & t )  [inline]
```

6.33.2.13 operator==()

```
const bool corenc::Mesh::CRectangle::operator== (
            const CRectangle & t )  [inline]
```

6.33.2.14 operator>>()

```
std::istream & corenc::Mesh::CRectangle::operator>> (
            std::istream & is )  [inline]
```

6.33.2.15 SetEdge()

```
void CRectangle::SetEdge (
            const int k,
            const int edge )  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.33.2.16 SetFacet()

```
void CRectangle::SetFacet (
            const int k,
            const int facet )  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.33.2.17 SetNode()

```
void CRectangle::SetNode (
            const int k,
            const int node )  [virtual]
```

Implements corenc::Mesh::CShape.

6.33.2.18 SetOrder()

```
const int CRectangle::SetOrder (
            const int px,
            const int py )
```

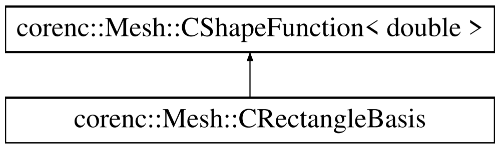
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/Rectangle.h
- CoreNCFEM/FiniteElements/Rectangle.cpp

6.34 corenc::Mesh::CRectangleBasis Class Reference

```
#include <Rectangle.h>
```

Inheritance diagram for corenc::Mesh::CRectangleBasis:



Public Member Functions

- CRectangleBasis ()
- CRectangleBasis (const Point &, const Point &, const Point &, const Point &, const int order)
- CRectangleBasis (const Point ∗, const int order)
- CRectangleBasis (const CRectangleBasis &)
- CRectangleBasis & operator= (const CRectangleBasis &t)
- ∼CRectangleBasis ()
- const int GetNumberOfShapeFunctions () const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetValue (const Point &) const
- const int IncreaseOrder ()
- const double GetMeasure () const
- const double GetWeight (const int, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const

6.34.1 Constructor & Destructor Documentation

6.34.1.1 CRectangleBasis() [1/4]

```
CRectangleBasis::CRectangleBasis ( )
```

6.34.1.2 CRectangleBasis() [2/4]

```
CRectangleBasis::CRectangleBasis (
            const Point & p1,
            const Point & p2,
            const Point & p3,
            const Point & p4,
            const int order )
```

6.34.1.3 CRectangleBasis() [3/4]

```
CRectangleBasis::CRectangleBasis (
            const Point * p,
            const int order )
```

6.34.1.4 CRectangleBasis() [4/4]

```
CRectangleBasis::CRectangleBasis (
            const CRectangleBasis & t )
```

6.34.1.5 ∼CRectangleBasis()

```
corenc::Mesh::CRectangleBasis::~CRectangleBasis ( ) [inline]
```

6.34.2 Member Function Documentation

6.34.2.1 GetGradShapeFunction()

```
const Point CRectangleBasis::GetGradShapeFunction (
            const int k,
            const Point & p ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.34.2.2 GetMeasure()

```
const double corenc::Mesh::CRectangleBasis::GetMeasure ( ) const  [inline], [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.34.2.3 GetNormal()

```
const Point CRectangleBasis::GetNormal ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.34.2.4 GetNumberOfShapeFunctions()

```
const int CRectangleBasis::GetNumberOfShapeFunctions ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.34.2.5 GetShapeFunction()

```
const double CRectangleBasis::GetShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.34.2.6 GetValue()

```
const double CRectangleBasis::GetValue (
            const Point & p ) const
```

6.34.2.7 GetWeight()

```
const double CRectangleBasis::GetWeight (
            const int node,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const
```

6.34.2.8 IncreaseOrder()

```
const int CRectangleBasis::IncreaseOrder ( )
```

6.34.2.9 operator=()

```
CRectangleBasis & corenc::Mesh::CRectangleBasis::operator= (
            const CRectangleBasis & t )  [inline]
```

6.34.2.10 ReverseNormal()

```
void CRectangleBasis::ReverseNormal ( )  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

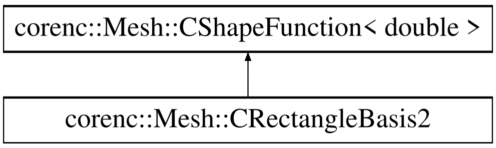
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/Rectangle.h
- CoreNCFEM/FiniteElements/Rectangle.cpp

6.35 corenc::Mesh::CRectangleBasis2 Class Reference

```
#include <Rectangle.h>
```

Inheritance diagram for corenc::Mesh::CRectangleBasis2:



Public Member Functions

- CRectangleBasis2 ()
- CRectangleBasis2 (const Point &, const Point &, const Point &, const Point &, const int order)
- CRectangleBasis2 (const Point ∗, const int order)
- CRectangleBasis2 (const CRectangleBasis2 &)
- CRectangleBasis2 & operator= (const CRectangleBasis2 &t)
- ∼CRectangleBasis2 ()
- const int GetNumberOfShapeFunctions () const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetValue (const Point &) const
- const int IncreaseOrder ()
- const double GetMeasure () const
- const double GetWeight (const int, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const

6.35.1 Constructor & Destructor Documentation

6.35.1.1 CRectangleBasis2() [1/4]

```
CRectangleBasis2::CRectangleBasis2 ( )
```

6.35.1.2 CRectangleBasis2() [2/4]

```
CRectangleBasis2::CRectangleBasis2 (
            const Point & p1,
            const Point & p2,
            const Point & p3,
            const Point & p4,
            const int order )
```

6.35.1.3 CRectangleBasis2() [3/4]

```
CRectangleBasis2::CRectangleBasis2 (
            const Point * p,
            const int order )
```

6.35.1.4 CRectangleBasis2() [4/4]

```
CRectangleBasis2::CRectangleBasis2 (
            const CRectangleBasis2 & t )
```

6.35.1.5 ∼CRectangleBasis2()

```
corenc::Mesh::CRectangleBasis2::∼CRectangleBasis2 ( )  [inline]
```

6.35.2 Member Function Documentation

6.35.2.1 GetGradShapeFunction()

```
const Point CRectangleBasis2::GetGradShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.35.2.2 GetMeasure()

```
const double corenc::Mesh::CRectangleBasis2::GetMeasure ( ) const  [inline], [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.35.2.3 GetNormal()

```
const Point CRectangleBasis2::GetNormal ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.35.2.4 GetNumberOfShapeFunctions()

```
const int CRectangleBasis2::GetNumberOfShapeFunctions ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.35.2.5 GetShapeFunction()

```
const double CRectangleBasis2::GetShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.35.2.6 GetValue()

```
const double CRectangleBasis2::GetValue (
            const Point & p ) const
```

6.35.2.7 GetWeight()

```
const double CRectangleBasis2::GetWeight (
            const int node,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const
```

6.35.2.8 IncreaseOrder()

```
const int CRectangleBasis2::IncreaseOrder ( )
```

6.35.2.9 operator=()

```
CRectangleBasis2 & corenc::Mesh::CRectangleBasis2::operator= (
            const CRectangleBasis2 & t ) [inline]
```

6.35.2.10 ReverseNormal()

```
void CRectangleBasis2::ReverseNormal ( ) [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

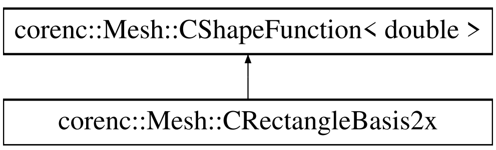
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/Rectangle.h
- CoreNCFEM/FiniteElements/RectangleBasis2.cpp

6.36 corenc::Mesh::CRectangleBasis2x Class Reference

```
#include <Rectangle.h>
```

Inheritance diagram for corenc::Mesh::CRectangleBasis2x:



Public Member Functions

- CRectangleBasis2x ()
- CRectangleBasis2x (const Point &, const Point &, const Point &, const Point &, const int order)
- CRectangleBasis2x (const Point ∗, const int order)
- CRectangleBasis2x (const CRectangleBasis2x &)
- CRectangleBasis2x & operator= (const CRectangleBasis2x &t)
- ∼CRectangleBasis2x ()
- const int GetNumberOfShapeFunctions () const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetValue (const Point &) const
- const int IncreaseOrder ()
- const double GetMeasure () const
- const double GetWeight (const int, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const

6.36.1 Constructor & Destructor Documentation

6.36.1.1 CRectangleBasis2x() [1/4]

```
CRectangleBasis2x::CRectangleBasis2x ( )
```

6.36.1.2 CRectangleBasis2x() [2/4]

```
CRectangleBasis2x::CRectangleBasis2x (
            const Point & p1,
            const Point & p2,
            const Point & p3,
            const Point & p4,
            const int order )
```

6.36.1.3 CRectangleBasis2x() [3/4]

```
CRectangleBasis2x::CRectangleBasis2x (
            const Point * p,
            const int order )
```

6.36.1.4 CRectangleBasis2x() [4/4]

```
CRectangleBasis2x::CRectangleBasis2x (
            const CRectangleBasis2x & t )
```

6.36.1.5 ∼CRectangleBasis2x()

```
corenc::Mesh::CRectangleBasis2x::∼CRectangleBasis2x ( )  [inline]
```

6.36.2 Member Function Documentation

6.36.2.1 GetGradShapeFunction()

```
const Point CRectangleBasis2x::GetGradShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.36.2.2 GetMeasure()

```
const double corenc::Mesh::CRectangleBasis2x::GetMeasure ( ) const  [inline], [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.36.2.3 GetNormal()

```
const Point CRectangleBasis2x::GetNormal ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.36.2.4 GetNumberOfShapeFunctions()

```
const int CRectangleBasis2x::GetNumberOfShapeFunctions ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.36.2.5 GetShapeFunction()

```
const double CRectangleBasis2x::GetShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.36.2.6 GetValue()

```
const double CRectangleBasis2x::GetValue (
            const Point & p ) const
```

6.36.2.7 GetWeight()

```
const double CRectangleBasis2x::GetWeight (
            const int node,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const
```

6.36.2.8 IncreaseOrder()

```
const int CRectangleBasis2x::IncreaseOrder ( )
```

6.36.2.9 operator=()

```
CRectangleBasis2x & corenc::Mesh::CRectangleBasis2x::operator= (
            const CRectangleBasis2x & t )  [inline]
```

6.36.2.10 ReverseNormal()

```
void CRectangleBasis2x::ReverseNormal ( )  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

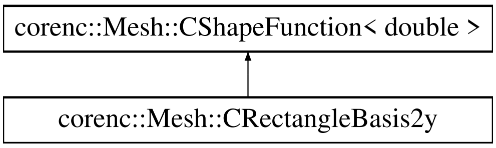
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/Rectangle.h
- CoreNCFEM/FiniteElements/CRectangleBasis2x.cpp

6.37 corenc::Mesh::CRectangleBasis2y Class Reference

```
#include <Rectangle.h>
```

Inheritance diagram for corenc::Mesh::CRectangleBasis2y:



Public Member Functions

- CRectangleBasis2y ()
- CRectangleBasis2y (const Point &, const Point &, const Point &, const Point &, const int order)
- CRectangleBasis2y (const Point ∗, const int order)
- CRectangleBasis2y (const CRectangleBasis2y &)
- CRectangleBasis2y & operator= (const CRectangleBasis2y &t)
- ∼CRectangleBasis2y ()
- const int GetNumberOfShapeFunctions () const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetValue (const Point &) const
- const int IncreaseOrder ()
- const double GetMeasure () const
- const double GetWeight (const int, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const

6.37.1 Constructor & Destructor Documentation

6.37.1.1 CRectangleBasis2y() [1/4]

```
CRectangleBasis2y::CRectangleBasis2y ( )
```

6.37.1.2 CRectangleBasis2y() [2/4]

```
CRectangleBasis2y::CRectangleBasis2y (
            const Point & p1,
            const Point & p2,
            const Point & p3,
            const Point & p4,
            const int order )
```

6.37.1.3 CRectangleBasis2y() [3/4]

```
CRectangleBasis2y::CRectangleBasis2y (
            const Point * p,
            const int order )
```

6.37.1.4 CRectangleBasis2y() [4/4]

```
CRectangleBasis2y::CRectangleBasis2y (
            const CRectangleBasis2y & t )
```

6.37.1.5 ∼CRectangleBasis2y()

```
corenc::Mesh::CRectangleBasis2y::~CRectangleBasis2y ( ) [inline]
```

6.37.2 Member Function Documentation

6.37.2.1 GetGradShapeFunction()

```
const Point CRectangleBasis2y::GetGradShapeFunction (
            const int k,
            const Point & p ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.37.2.2 GetMeasure()

```
const double corenc::Mesh::CRectangleBasis2y::GetMeasure ( ) const [inline], [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.37.2.3 GetNormal()

```
const Point CRectangleBasis2y::GetNormal ( ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.37.2.4 GetNumberOfShapeFunctions()

```
const int CRectangleBasis2y::GetNumberOfShapeFunctions ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.37.2.5 GetShapeFunction()

```
const double CRectangleBasis2y::GetShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.37.2.6 GetValue()

```
const double CRectangleBasis2y::GetValue (
            const Point & p ) const
```

6.37.2.7 GetWeight()

```
const double CRectangleBasis2y::GetWeight (
            const int node,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const
```

6.37.2.8 IncreaseOrder()

```
const int CRectangleBasis2y::IncreaseOrder ( )
```

6.37.2.9 operator=()

```
CRectangleBasis2y & corenc::Mesh::CRectangleBasis2y::operator= (
            const CRectangleBasis2y & t )  [inline]
```

6.37.2.10 ReverseNormal()

```
void CRectangleBasis2y::ReverseNormal ( )  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

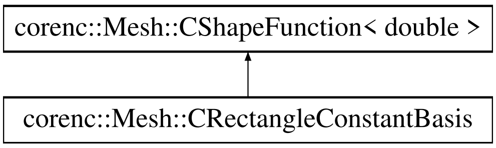
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/Rectangle.h
- CoreNCFEM/FiniteElements/RectangleBasis2y.cpp

6.38 corenc::Mesh::CRectangleConstantBasis Class Reference

```
#include <Rectangle.h>
```

Inheritance diagram for corenc::Mesh::CRectangleConstantBasis:



Public Member Functions

- CRectangleConstantBasis ()
- CRectangleConstantBasis (const Point &, const Point &, const Point &, const Point &, const int order)
- CRectangleConstantBasis (const Point ∗, const int order)
- CRectangleConstantBasis (const CRectangleConstantBasis &)
- CRectangleConstantBasis & operator= (const CRectangleConstantBasis &t)
- ∼CRectangleConstantBasis ()
- const int GetNumberOfShapeFunctions () const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetValue (const Point &) const
- const int IncreaseOrder ()
- const double GetMeasure () const

6.38.1 Constructor & Destructor Documentation

6.38.1.1 CRectangleConstantBasis() [1/4]

```
CRectangleConstantBasis::CRectangleConstantBasis ( )
```

6.38.1.2 CRectangleConstantBasis() [2/4]

```
CRectangleConstantBasis::CRectangleConstantBasis (
            const Point & p1,
            const Point & p2,
            const Point & p3,
            const Point & p4,
            const int order )
```

6.38.1.3 CRectangleConstantBasis() [3/4]

```
CRectangleConstantBasis::CRectangleConstantBasis (
            const Point * p,
            const int order )
```

6.38.1.4 CRectangleConstantBasis() [4/4]

```
CRectangleConstantBasis::CRectangleConstantBasis (
            const CRectangleConstantBasis & t )
```

6.38.1.5 ∼CRectangleConstantBasis()

```
corenc::Mesh::CRectangleConstantBasis::∼CRectangleConstantBasis ( ) [inline]
```

6.38.2 Member Function Documentation

6.38.2.1 GetGradShapeFunction()

```
const Point CRectangleConstantBasis::GetGradShapeFunction (
            const int k,
            const Point & p ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.38.2.2 GetMeasure()

```
const double corenc::Mesh::CRectangleConstantBasis::GetMeasure ( ) const  [inline], [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.38.2.3 GetNormal()

```
const Point CRectangleConstantBasis::GetNormal ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.38.2.4 GetNumberOfShapeFunctions()

```
const int CRectangleConstantBasis::GetNumberOfShapeFunctions ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.38.2.5 GetShapeFunction()

```
const double CRectangleConstantBasis::GetShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.38.2.6 GetValue()

```
const double CRectangleConstantBasis::GetValue (
            const Point & p ) const
```

6.38.2.7 IncreaseOrder()

```
const int CRectangleConstantBasis::IncreaseOrder ( )
```

6.38.2.8 operator=()

```
CRectangleConstantBasis & corenc::Mesh::CRectangleConstantBasis::operator= (
            const CRectangleConstantBasis & t ) [inline]
```

6.38.2.9 ReverseNormal()

```
void CRectangleConstantBasis::ReverseNormal ( ) [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

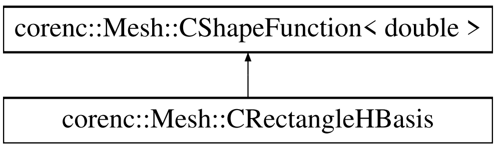
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/Rectangle.h
- CoreNCFEM/FiniteElements/Rectangle.cpp

6.39 corenc::Mesh::CRectangleHBasis Class Reference

```
#include <Rectangle.h>
```

Inheritance diagram for corenc::Mesh::CRectangleHBasis:



Public Member Functions

- CRectangleHBasis ()
- CRectangleHBasis (const Point &, const Point &, const Point &, const Point &, const int order)
- CRectangleHBasis (const Point &, const Point &, const Point &, const Point &, const int px, const int py)
- CRectangleHBasis (const Point ∗, const int order)
- CRectangleHBasis (const Point ∗, const int px, const int py)
- CRectangleHBasis (const CRectangleHBasis &)
- CRectangleHBasis & operator= (const CRectangleHBasis &t)
- ∼CRectangleHBasis ()
- const int GetNumberOfShapeFunctions () const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetValue (const Point &) const
- const int IncreaseOrder ()
- const int SetOrder (const int px, const int py)
- const double GetMeasure () const
- const double GetWeight (const int, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const

6.39.1 Constructor & Destructor Documentation

6.39.1.1 CRectangleHBasis() [1/6]

```
CRectangleHBasis::CRectangleHBasis ( )
```

6.39.1.2 CRectangleHBasis() [2/6]

```
CRectangleHBasis::CRectangleHBasis (
            const Point & p1,
            const Point & p2,
            const Point & p3,
            const Point & p4,
            const int order )
```

6.39.1.3 CRectangleHBasis() [3/6]

```
CRectangleHBasis::CRectangleHBasis (
            const Point & p1,
            const Point & p2,
            const Point & p3,
            const Point & p4,
            const int px,
            const int py )
```

6.39.1.4 CRectangleHBasis() [4/6]

```
CRectangleHBasis::CRectangleHBasis (
            const Point * p,
            const int order )
```

6.39.1.5 CRectangleHBasis() [5/6]

```
CRectangleHBasis::CRectangleHBasis (
            const Point * p,
            const int px,
            const int py )
```

6.39.1.6 CRectangleHBasis() [6/6]

```
CRectangleHBasis::CRectangleHBasis (
            const CRectangleHBasis & t )
```

6.39.1.7 ∼CRectangleHBasis()

```
corenc::Mesh::CRectangleHBasis::∼CRectangleHBasis ( ) [inline]
```

6.39.2 Member Function Documentation

6.39.2.1 GetGradShapeFunction()

```
const Point CRectangleHBasis::GetGradShapeFunction (
            const int k,
            const Point & p ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.39.2.2 GetMeasure()

```
const double corenc::Mesh::CRectangleHBasis::GetMeasure ( ) const [inline], [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.39.2.3 GetNormal()

```
const Point CRectangleHBasis::GetNormal ( ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.39.2.4 GetNumberOfShapeFunctions()

```
const int CRectangleHBasis::GetNumberOfShapeFunctions ( ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.39.2.5 GetShapeFunction()

```
const double CRectangleHBasis::GetShapeFunction (
             const int k,
             const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.39.2.6 GetValue()

```
const double CRectangleHBasis::GetValue (
             const Point & p ) const
```

6.39.2.7 GetWeight()

```
const double CRectangleHBasis::GetWeight (
             const int node,
             const std::vector< Point > & verts,
             const std::function< const double(const Point &)> & f ) const
```

6.39.2.8 IncreaseOrder()

```
const int CRectangleHBasis::IncreaseOrder ( )
```

6.39.2.9 operator=()

```
CRectangleHBasis & corenc::Mesh::CRectangleHBasis::operator= (
             const CRectangleHBasis & t )  [inline]
```

6.39.2.10 ReverseNormal()

```
void CRectangleHBasis::ReverseNormal ( )  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.39.2.11 SetOrder()

```
const int CRectangleHBasis::SetOrder (
            const int px,
            const int py )
```

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

- CoreNCFEM/FiniteElements/Rectangle.h
- CoreNCFEM/FiniteElements/RectangleHBasis.cpp

6.40 corenc::Mesh::CRegularMesh Class Reference

```
#include <RegularMesh.h>
```

Public Member Functions

- CRegularMesh ()
- CRegularMesh (const std::string &file_name)
- CRegularMesh (const CRegularMesh &)
- CRegularMesh (const Point &p1, const Point &p2, const int nx, const int ny)
- CRegularMesh (const Point &p1, const Point &p2, const int nx, const int ny, const int px, const int py)
- CRegularMesh (const double x1, const double y1, const double x2, const double y2, const int nx, const int ny)
- CRegularMesh & operator= (const CRegularMesh &tr)
- CRegularMesh ∗ Clone () const
- const unsigned int GetNumberOfElements () const
- const unsigned int GetNumberOfNodes () const
- const int GetNumberOfINodes () const
- const unsigned int GetNumberOfBoundaries () const
- const int FindElement (const Point &) const
- const Point GetNode (const unsigned int) const
- const CElement2D ∗ GetElement (const unsigned int) const
- const CElement ∗ GetBoundary (const unsigned int) const
- const double getMinSize () const
- const double getSolution (const unsigned int element, const unsigned int node) const
- const int updateSolution (const unsigned int element, const unsigned int node, const double value)
- const std::vector< double > getSolution () const
- const int updateSolution (const std::vector< double > &)
- const int updateSolution (const unsigned int element, const unsigned int node, CSolution ∗value)
- const double getParameter (Parameters, const unsigned int, const Point &) const
- const double getParameter (Parameters, const unsigned int, const int) const
- const int setParameter (Parameters, const double, const unsigned int)
- const int setParameter (const CParameter &, const unsigned int type)
- const int updateSolution (const unsigned int node, const double value)
- const int refine_hx ()
- const int refine_hy ()
- const int refine_h ()
- const int refine_p ()
- const int refine_hp ()
- const int interpolate (const int node) const
- ∼CRegularMesh ()
- auto GetElements () -> decltype(m_elems)
- auto GetBoundary () -> decltype(m_edges)

6.40.1 Constructor & Destructor Documentation

6.40.1.1 CRegularMesh() [1/6]

```
CRegularMesh::CRegularMesh ( )
```

6.40.1.2 CRegularMesh() [2/6]

```
corenc::Mesh::CRegularMesh::CRegularMesh (
            const std::string & file_name )
```

6.40.1.3 CRegularMesh() [3/6]

```
CRegularMesh::CRegularMesh (
            const CRegularMesh & tr )
```

6.40.1.4 CRegularMesh() [4/6]

```
CRegularMesh::CRegularMesh (
            const Point & p1,
            const Point & p2,
            const int nx,
            const int ny )
```

6.40.1.5 CRegularMesh() [5/6]

```
CRegularMesh::CRegularMesh (
            const Point & p1,
            const Point & p2,
            const int nx,
            const int ny,
            const int px,
            const int py )
```

6.40.1.6 CRegularMesh() [6/6]

```
CRegularMesh::CRegularMesh (
            const double x1,
            const double y1,
            const double x2,
            const double y2,
            const int nx,
            const int ny )
```

6.40.1.7 ∼CRegularMesh()

```
CRegularMesh::∼CRegularMesh ( )
```

6.40.2 Member Function Documentation

6.40.2.1 Clone()

```
CRegularMesh * corenc::Mesh::CRegularMesh::Clone ( ) const  [inline]
```

6.40.2.2 FindElement()

```
const int CRegularMesh::FindElement (
            const Point & test ) const
```

6.40.2.3 GetBoundary() [1/2]

```
auto corenc::Mesh::CRegularMesh::GetBoundary ( ) -> decltype(m_edges)  [inline]
```

6.40.2.4 GetBoundary() [2/2]

```
const CElement * CRegularMesh::GetBoundary (
            const unsigned int n ) const
```

6.40.2.5 GetElement()

```
const CElement2D * CRegularMesh::GetElement (
            const unsigned int n ) const
```

6.40.2.6 GetElements()

```
auto corenc::Mesh::CRegularMesh::GetElements ( ) -> decltype(m_elems)   [inline]
```

6.40.2.7 getMinSize()

```
const double corenc::Mesh::CRegularMesh::getMinSize ( ) const  [inline]
```

6.40.2.8 GetNode()

```
const Mesh::Point CRegularMesh::GetNode (
            const unsigned int n ) const
```

6.40.2.9 GetNumberOfBoundaries()

```
const unsigned int CRegularMesh::GetNumberOfBoundaries ( ) const
```

6.40.2.10 GetNumberOfElements()

```
const unsigned int CRegularMesh::GetNumberOfElements ( ) const
```

6.40.2.11 GetNumberOfINodes()

```
const int CRegularMesh::GetNumberOfINodes ( ) const
```

6.40.2.12 GetNumberOfNodes()

```
const unsigned int CRegularMesh::GetNumberOfNodes ( ) const
```

6.40.2.13 getParameter() [1/2]

```
const double CRegularMesh::getParameter (
            Parameters param,
            const unsigned int l,
            const int i ) const
```

6.40.2.14 getParameter() [2/2]

```
const double CRegularMesh::getParameter (
            Parameters param,
            const unsigned int l,
            const Point & p ) const
```

6.40.2.15 getSolution() [1/2]

```
const std::vector< double > CRegularMesh::getSolution ( ) const
```

6.40.2.16 getSolution() [2/2]

```
const double CRegularMesh::getSolution (
            const unsigned int element,
            const unsigned int node ) const
```

6.40.2.17 interpolate()

```
const int CRegularMesh::interpolate (
            const int node ) const
```

6.40.2.18 operator=()

```
CRegularMesh & corenc::Mesh::CRegularMesh::operator= (
             const CRegularMesh & tr )  [inline]
```

6.40.2.19 refine_h()

```
const int CRegularMesh::refine_h ( )
```

6.40.2.20 refine_hp()

```
const int corenc::Mesh::CRegularMesh::refine_hp ( )
```

6.40.2.21 refine_hx()

```
const int corenc::Mesh::CRegularMesh::refine_hx ( )
```

6.40.2.22 refine_hy()

```
const int corenc::Mesh::CRegularMesh::refine_hy ( )
```

6.40.2.23 refine_p()

```
const int CRegularMesh::refine_p ( )
```

6.40.2.24 setParameter() [1/2]

```
const int CRegularMesh::setParameter (
             const CParameter & p,
             const unsigned int type )
```

6.40.2.25 setParameter() [2/2]

```
const int CRegularMesh::setParameter (
            Parameters param,
            const double ,
            const unsigned int )
```

6.40.2.26 updateSolution() [1/4]

```
const int CRegularMesh::updateSolution (
            const std::vector< double > & )
```

6.40.2.27 updateSolution() [2/4]

```
const int CRegularMesh::updateSolution (
            const unsigned int element,
            const unsigned int node,
            const double value )
```

6.40.2.28 updateSolution() [3/4]

```
const int CRegularMesh::updateSolution (
            const unsigned int element,
            const unsigned int node,
            CSolution * value )
```

6.40.2.29 updateSolution() [4/4]

```
const int CRegularMesh::updateSolution (
            const unsigned int node,
            const double value )
```

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

- CoreNCFEM/Grids/RegularMesh.h
- CoreNCFEM/Grids/RegularMesh.cpp

6.41 corenc::Mesh::CRegularMesh3D Class Reference

```
#include <RegularMesh3D.h>
```

Public Member Functions

- CRegularMesh3D ()
- CRegularMesh3D (const std::string &file_name)
- CRegularMesh3D (const CRegularMesh3D &)
- CRegularMesh3D (const Point &p1, const Point &p2, const int nx, const int ny)
- CRegularMesh3D (const Point &p1, const Point &p2, const int nx, const int ny, const int px, const int py)
- CRegularMesh3D (const double x1, const double y1, const double x2, const double y2, const int nx, const int ny)
- CRegularMesh3D & operator= (const CRegularMesh3D &tr)
- CRegularMesh3D ∗ Clone () const
- const unsigned int GetNumberOfElements () const
- const unsigned int GetNumberOfNodes () const
- const int GetNumberOfINodes () const
- const unsigned int GetNumberOfBoundaries () const
- const int FindElement (const Point &) const
- const Point GetNode (const unsigned int) const
- const CElement ∗ GetElement (const unsigned int) const
- const CElement ∗ GetBoundary (const unsigned int) const
- const double getMinSize () const
- const double getSolution (const unsigned int element, const unsigned int node) const
- const int updateSolution (const unsigned int element, const unsigned int node, const double value)
- const std::vector< double > getSolution () const
- const int updateSolution (const std::vector< double > &)
- const int updateSolution (const unsigned int element, const unsigned int node, CSolution ∗value)
- const double getParameter (Parameters, const unsigned int, const Point &) const
- const double getParameter (Parameters, const unsigned int, const int) const
- const int setParameter (Parameters, const double, const unsigned int)
- const int setParameter (const CParameter &, const unsigned int type)
- const int updateSolution (const unsigned int node, const double value)
- const int refine_hx ()
- const int refine_hy ()
- const int refine_h ()
- const int refine_p ()
- const int refine_hp ()
- const int interpolate (const int node) const
- ∼CRegularMesh3D ()
- auto GetElements () -> decltype(m_elems)
- auto GetBoundary () -> decltype(m_edges)

6.41.1 Constructor & Destructor Documentation

6.41.1.1 CRegularMesh3D() [1/6]

```
CRegularMesh3D::CRegularMesh3D ( )
```

6.41.1.2 CRegularMesh3D() [2/6]

```
corenc::Mesh::CRegularMesh3D::CRegularMesh3D (
             const std::string & file_name )
```

6.41.1.3 CRegularMesh3D() [3/6]

```
CRegularMesh3D::CRegularMesh3D (
             const CRegularMesh3D & tr )
```

6.41.1.4 CRegularMesh3D() [4/6]

```
CRegularMesh3D::CRegularMesh3D (
             const Point & p1,
             const Point & p2,
             const int nx,
             const int ny )
```

6.41.1.5 CRegularMesh3D() [5/6]

```
CRegularMesh3D::CRegularMesh3D (
             const Point & p1,
             const Point & p2,
             const int nx,
             const int ny,
             const int px,
             const int py )
```

6.41.1.6 CRegularMesh3D() [6/6]

```
CRegularMesh3D::CRegularMesh3D (
             const double x1,
             const double y1,
             const double x2,
             const double y2,
             const int nx,
             const int ny )
```

6.41.1.7 ∼CRegularMesh3D()

```
CRegularMesh3D::∼CRegularMesh3D ( )
```

6.41.2 Member Function Documentation

6.41.2.1 Clone()

```
CRegularMesh3D * corenc::Mesh::CRegularMesh3D::Clone ( ) const [inline]
```

6.41.2.2 FindElement()

```
const int CRegularMesh3D::FindElement (
            const Point & test ) const
```

6.41.2.3 GetBoundary() [1/2]

```
auto corenc::Mesh::CRegularMesh3D::GetBoundary ( ) -> decltype(m_edges) [inline]
```

6.41.2.4 GetBoundary() [2/2]

```
const CElement * CRegularMesh3D::GetBoundary (
            const unsigned int n ) const
```

6.41.2.5 GetElement()

```
const CElement * CRegularMesh3D::GetElement (
            const unsigned int n ) const
```

6.41.2.6 GetElements()

```
auto corenc::Mesh::CRegularMesh3D::GetElements ( ) -> decltype(m_elems) [inline]
```

6.41.2.7 getMinSize()

```
const double corenc::Mesh::CRegularMesh3D::getMinSize ( ) const  [inline]
```

6.41.2.8 GetNode()

```
const Mesh::Point CRegularMesh3D::GetNode (
            const unsigned int n ) const
```

6.41.2.9 GetNumberOfBoundaries()

```
const unsigned int CRegularMesh3D::GetNumberOfBoundaries ( ) const
```

6.41.2.10 GetNumberOfElements()

```
const unsigned int CRegularMesh3D::GetNumberOfElements ( ) const
```

6.41.2.11 GetNumberOfINodes()

```
const int CRegularMesh3D::GetNumberOfINodes ( ) const
```

6.41.2.12 GetNumberOfNodes()

```
const unsigned int CRegularMesh3D::GetNumberOfNodes ( ) const
```

6.41.2.13 getParameter() [1/2]

```
const double CRegularMesh3D::getParameter (
            Parameters param,
            const unsigned int l,
            const int i ) const
```

6.41.2.14 getParameter() [2/2]

```
const double CRegularMesh3D::getParameter (
            Parameters param,
            const unsigned int l,
            const Point & p ) const
```

6.41.2.15 getSolution() [1/2]

```
const std::vector< double > CRegularMesh3D::getSolution ( ) const
```

6.41.2.16 getSolution() [2/2]

```
const double CRegularMesh3D::getSolution (
            const unsigned int element,
            const unsigned int node ) const
```

6.41.2.17 interpolate()

```
const int CRegularMesh3D::interpolate (
            const int node ) const
```

6.41.2.18 operator=()

```
CRegularMesh3D & corenc::Mesh::CRegularMesh3D::operator= (
            const CRegularMesh3D & tr )  [inline]
```

6.41.2.19 refine_h()

```
const int CRegularMesh3D::refine_h ( )
```

6.41.2.20 refine_hp()

```
const int corenc::Mesh::CRegularMesh3D::refine_hp ( )
```

6.41.2.21 refine_hx()

```
const int corenc::Mesh::CRegularMesh3D::refine_hx ( )
```

6.41.2.22 refine_hy()

```
const int corenc::Mesh::CRegularMesh3D::refine_hy ( )
```

6.41.2.23 refine_p()

```
const int CRegularMesh3D::refine_p ( )
```

6.41.2.24 setParameter() [1/2]

```
const int CRegularMesh3D::setParameter (
            const CParameter & p,
            const unsigned int type )
```

6.41.2.25 setParameter() [2/2]

```
const int CRegularMesh3D::setParameter (
            Parameters param,
            const double ,
            const unsigned int  )
```

6.41.2.26 updateSolution() [1/4]

```
const int CRegularMesh3D::updateSolution (
            const std::vector< double > &  )
```

6.41.2.27 updateSolution() [2/4]

```
const int CRegularMesh3D::updateSolution (
            const unsigned int element,
            const unsigned int node,
            const double value )
```

6.41.2.28 updateSolution() [3/4]

```
const int CRegularMesh3D::updateSolution (
            const unsigned int element,
            const unsigned int node,
            CSolution * value )
```

6.41.2.29 updateSolution() [4/4]

```
const int CRegularMesh3D::updateSolution (
            const unsigned int node,
            const double value )
```

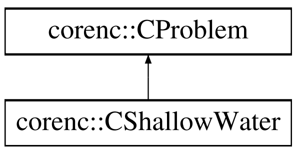
The documentation for this class was generated from the following files:

- CoreNCFEM/Grids/RegularMesh3D.h
- CoreNCFEM/Grids/RegularMesh3D.cpp

6.42 corenc::CShallowWater Class Reference

```
#include <ShallowWater.h>
```

Inheritance diagram for corenc::CShallowWater:



Public Member Functions

- CShallowWater ()
- ∼CShallowWater ()
- Terms getTerm (const unsigned int) const
- const unsigned int getNumberOfTerms () const
- const int setTerm (const unsigned int, const Terms &)
- const int addTerm (const Terms &)
- const int removeTerm (const Terms &)
- const int load_parameters (const std::string &file_name)
- const double get_parameter (const Terms &, const int element_type, const Mesh::Point &) const
- const double get_parameter (const Terms &, const int element_number, const int element_type, const Mesh::Point &) const
- const double get_boundary_parameter (const int type, const int element_type, const Mesh::Point &) const
- const double get_boundary_parameter (const int type, const int element_number, const int element_type, const Mesh::Point &) const
- const int get_number_of_boundaries () const

- const double get_solution (const int sys_number, const int element_type, const int element_number, const Mesh::Point &) const
- const int get_boundary_type (const int number) const
- const int add_parameter (const Terms &, const int element_type, const Mesh::parameter< double > &value)
- const int set_parameter (const Terms &, const int element_type, const Mesh::parameter< double > &value)
- const int set_boundary_parameter (const int type, const int element_type, const boundary &value)
- const int add_boundary_parameter (const int type, const int element_type, const Mesh::parameter< double > &value)
- const int add_boundary_parameter (const int element_type, const Mesh::parameter< double > &value, const Mesh::parameter< double > &value2)

6.42.1 Constructor & Destructor Documentation

6.42.1.1 CShallowWater()

```
CShallowWater::CShallowWater ( )
```

6.42.1.2 ∼CShallowWater()

```
CShallowWater::∼CShallowWater ( )
```

6.42.2 Member Function Documentation

6.42.2.1 add_boundary_parameter() [1/2]

```
const int CShallowWater::add_boundary_parameter (
            const int element_type,
            const Mesh::parameter< double > & value,
            const Mesh::parameter< double > & value2 )
```

6.42.2.2 add_boundary_parameter() [2/2]

```
const int CShallowWater::add_boundary_parameter (
            const int type,
            const int element_type,
            const Mesh::parameter< double > & value )
```

6.42.2.3 add_parameter()

```
const int CShallowWater::add_parameter (
            const Terms & term,
            const int element_type,
            const Mesh::parameter< double > & value )
```

6.42.2.4 addTerm()

```
const int CShallowWater::addTerm (
            const Terms & term )  [virtual]
```

Implements corenc::CProblem.

6.42.2.5 get_boundary_parameter() [1/2]

```
const double CShallowWater::get_boundary_parameter (
            const int type,
            const int element_number,
            const int element_type,
            const Mesh::Point & p ) const
```

6.42.2.6 get_boundary_parameter() [2/2]

```
const double CShallowWater::get_boundary_parameter (
            const int type,
            const int element_type,
            const Mesh::Point & p ) const
```

6.42.2.7 get_boundary_type()

```
const int CShallowWater::get_boundary_type (
            const int number ) const
```

6.42.2.8 get_number_of_boundaries()

```
const int CShallowWater::get_number_of_boundaries ( ) const
```

6.42.2.9 get_parameter() [1/2]

```
const double CShallowWater::get_parameter (
            const Terms & term,
            const int element_number,
            const int element_type,
            const Mesh::Point & p ) const
```

6.42.2.10 get_parameter() [2/2]

```
const double CShallowWater::get_parameter (
            const Terms & term,
            const int element_type,
            const Mesh::Point & p ) const
```

6.42.2.11 get_solution()

```
const double CShallowWater::get_solution (
            const int sys_number,
            const int element_type,
            const int element_number,
            const Mesh::Point & ) const
```

6.42.2.12 getNumberOfTerms()

```
const unsigned int CShallowWater::getNumberOfTerms ( ) const [virtual]
```

Implements corenc::CProblem.

6.42.2.13 getTerm()

```
Terms CShallowWater::getTerm (
            const unsigned int i ) const [virtual]
```

Implements corenc::CProblem.

6.42.2.14 load_parameters()

```
const int CShallowWater::load_parameters (
            const std::string & file_name )  [virtual]
```

Implements corenc::CProblem.

6.42.2.15 removeTerm()

```
const int CShallowWater::removeTerm (
            const Terms & term )
```

6.42.2.16 set_boundary_parameter()

```
const int CShallowWater::set_boundary_parameter (
            const int type,
            const int element_type,
            const boundary & value )
```

6.42.2.17 set_parameter()

```
const int CShallowWater::set_parameter (
            const Terms & term,
            const int element_type,
            const Mesh::parameter< double > & value )
```

6.42.2.18 setTerm()

```
const int CShallowWater::setTerm (
            const unsigned int i,
            const Terms & term )  [virtual]
```

Implements corenc::CProblem.

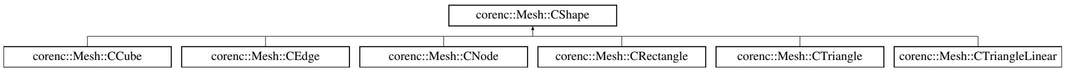
The documentation for this class was generated from the following files:

- Problems/ShallowWater.h
- Problems/ShallowWater.cpp

6.43 corenc::Mesh::CShape Class Reference

```
#include <Shape.h>
```

Inheritance diagram for corenc::Mesh::CShape:



Public Member Functions

- CShape ()
- CShape (const int ∗)
- virtual ∼CShape ()
- virtual const int GetNumberOfNodes () const
- virtual const int GetNumberOfEdges () const
- virtual const int GetNumberOfFacets () const
- virtual const int GetNode (const int) const
- virtual const int GetNode (const NODES &) const
- virtual const int GetEdge (const int) const
- virtual const int GetFacet (const int) const
- virtual const double Integrate (const scalar_func &, const std::vector< Point > &) const =0
- virtual const Point Integrate (const vector_func &, const std::vector< Point > &) const =0
- virtual const std::vector< double > Integrate (const std::function< const std::vector< double >(const Point &)> &, const std::vector< Point > &) const =0
- virtual void SetNode (const int, const int)=0
- virtual void SetEdge (const int, const int)
- virtual void SetFacet (const int, const int)

6.43.1 Constructor & Destructor Documentation

6.43.1.1 CShape() [1/2]

```
corenc::Mesh::CShape::CShape ( )   [inline]
```

6.43.1.2 CShape() [2/2]

```
corenc::Mesh::CShape::CShape (
            const int ∗  )   [inline]
```

6.43.1.3 ∼CShape()

```
virtual corenc::Mesh::CShape::~CShape ( ) [inline], [virtual]
```

6.43.2 Member Function Documentation

6.43.2.1 GetEdge()

```
virtual const int corenc::Mesh::CShape::GetEdge (
            const int ) const [inline], [virtual]
```

Reimplemented in corenc::Mesh::CCube, corenc::Mesh::CRectangle, corenc::Mesh::CTriangle, and corenc::Mesh::CTriangleLinear.

6.43.2.2 GetFacet()

```
virtual const int corenc::Mesh::CShape::GetFacet (
            const int ) const [inline], [virtual]
```

Reimplemented in corenc::Mesh::CCube, corenc::Mesh::CRectangle, corenc::Mesh::CTriangle, and corenc::Mesh::CTriangleLinear.

6.43.2.3 GetNode() [1/2]

```
virtual const int corenc::Mesh::CShape::GetNode (
            const int ) const [inline], [virtual]
```

Reimplemented in corenc::Mesh::CCube, corenc::Mesh::CEdge, corenc::Mesh::CNode, corenc::Mesh::CRectangle, corenc::Mesh::CTriangle, and corenc::Mesh::CTriangleLinear.

6.43.2.4 GetNode() [2/2]

```
virtual const int corenc::Mesh::CShape::GetNode (
            const NODES & ) const [inline], [virtual]
```

Reimplemented in corenc::Mesh::CCube, corenc::Mesh::CEdge, corenc::Mesh::CNode, corenc::Mesh::CRectangle, corenc::Mesh::CTriangle, and corenc::Mesh::CTriangleLinear.

6.43.2.5 GetNumberOfEdges()

```
virtual const int corenc::Mesh::CShape::GetNumberOfEdges ( ) const  [inline], [virtual]
```

Reimplemented in corenc::Mesh::CCube, corenc::Mesh::CRectangle, corenc::Mesh::CTriangle, and corenc::Mesh::CTriangleLinear.

6.43.2.6 GetNumberOfFacets()

```
virtual const int corenc::Mesh::CShape::GetNumberOfFacets ( ) const  [inline], [virtual]
```

Reimplemented in corenc::Mesh::CCube, corenc::Mesh::CRectangle, corenc::Mesh::CTriangle, and corenc::Mesh::CTriangleLinear.

6.43.2.7 GetNumberOfNodes()

```
virtual const int corenc::Mesh::CShape::GetNumberOfNodes ( ) const  [inline], [virtual]
```

Reimplemented in corenc::Mesh::CCube, corenc::Mesh::CEdge, corenc::Mesh::CNode, corenc::Mesh::CRectangle, corenc::Mesh::CTriangle, and corenc::Mesh::CTriangleLinear.

6.43.2.8 Integrate() [1/3]

```
virtual const double corenc::Mesh::CShape::Integrate (
            const scalar_func & ,
            const std::vector< Point > &  ) const  [pure virtual]
```

6.43.2.9 Integrate() [2/3]

```
virtual const std::vector< double > corenc::Mesh::CShape::Integrate (
            const std::function< const std::vector< double >(const Point &)> & ,
            const std::vector< Point > &  ) const  [pure virtual]
```

Implemented in corenc::Mesh::CCube, corenc::Mesh::CEdge, corenc::Mesh::CNode, corenc::Mesh::CRectangle, corenc::Mesh::CTriangle, and corenc::Mesh::CTriangleLinear.

6.43.2.10 Integrate() [3/3]

```
virtual const Point corenc::Mesh::CShape::Integrate (
            const vector_func & ,
            const std::vector< Point > &  ) const  [pure virtual]
```

6.43.2.11 SetEdge()

```
virtual void corenc::Mesh::CShape::SetEdge (
            const int ,
            const int )  [inline], [virtual]
```

Reimplemented in corenc::Mesh::CCube, corenc::Mesh::CRectangle, corenc::Mesh::CTriangle, and corenc::Mesh::CTriangleLinear.

6.43.2.12 SetFacet()

```
virtual void corenc::Mesh::CShape::SetFacet (
            const int ,
            const int )  [inline], [virtual]
```

Reimplemented in corenc::Mesh::CCube, corenc::Mesh::CRectangle, corenc::Mesh::CTriangle, and corenc::Mesh::CTriangleLinear.

6.43.2.13 SetNode()

```
virtual void corenc::Mesh::CShape::SetNode (
            const int ,
            const int )  [pure virtual]
```

Implemented in corenc::Mesh::CCube, corenc::Mesh::CEdge, corenc::Mesh::CNode, corenc::Mesh::CRectangle, corenc::Mesh::CTriangle, and corenc::Mesh::CTriangleLinear.

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

- CoreNCFEM/FiniteElements/Shape.h

6.44 corenc::Mesh::CShapeFunction< Type > Class Template Reference

```
#include <ShapeFunction.h>
```

Public Member Functions

- CShapeFunction ()
- CShapeFunction (const Point ∗)
- virtual ∼CShapeFunction ()
- virtual const int GetNumberOfShapeFunctions () const =0
- virtual const double GetShapeFunction (const int, const Point &) const =0
- virtual const Point GetGradShapeFunction (const int, const Point &) const =0
- virtual const Point GetNormal () const =0
- virtual void ReverseNormal ()=0
- virtual const double GetMeasure () const =0

6.44.1 Constructor & Destructor Documentation

6.44.1.1 CShapeFunction() [1/2]

```
template<class Type >
corenc::Mesh::CShapeFunction< Type >::CShapeFunction ( )  [inline]
```

6.44.1.2 CShapeFunction() [2/2]

```
template<class Type >
corenc::Mesh::CShapeFunction< Type >::CShapeFunction (
            const Point * )  [inline]
```

6.44.1.3 ∼CShapeFunction()

```
template<class Type >
virtual corenc::Mesh::CShapeFunction< Type >::∼CShapeFunction ( )  [inline], [virtual]
```

6.44.2 Member Function Documentation

6.44.2.1 GetGradShapeFunction()

```
template<class Type >
virtual const Point corenc::Mesh::CShapeFunction< Type >::GetGradShapeFunction (
            const int ,
            const Point & ) const  [pure virtual]
```

Implemented in corenc::Mesh::CCubeBasis, corenc::Mesh::CEdgeLinearBasis, corenc::Mesh::CEdgeConstantBasis, corenc::Mesh::CEdgeMultiBasis, corenc::Mesh::CEdgeHermiteBasis, corenc::Mesh::CEdge2ndBasis, corenc::Mesh::CNodeBasis, corenc::Mesh::CRectangleBasis, corenc::Mesh::CRectangleHBasis, corenc::Mesh::CRectangleBasis2x, corenc::Mesh::CRectangleB corenc::Mesh::CRectangleBasis2, corenc::Mesh::CRectangleConstantBasis, corenc::Mesh::CTriangleBasis, corenc::Mesh::CTriangleLagrangeBasis, corenc::Mesh::CTriangleLinearBasis, and corenc::Mesh::CTriangleBasis.

6.44.2.2 GetMeasure()

```
template<class Type >
virtual const double corenc::Mesh::CShapeFunction< Type >::GetMeasure ( ) const  [pure virtual]
```

Implemented in corenc::Mesh::CCubeBasis, corenc::Mesh::CEdgeLinearBasis, corenc::Mesh::CEdgeConstantBasis, corenc::Mesh::CEdgeMultiBasis, corenc::Mesh::CEdgeHermiteBasis, corenc::Mesh::CEdge2ndBasis, corenc::Mesh::CNodeBasis, corenc::Mesh::CRectangleBasis, corenc::Mesh::CRectangleHBasis, corenc::Mesh::CRectangleBasis2x, corenc::Mesh::CRectangleBa corenc::Mesh::CRectangleBasis2, corenc::Mesh::CRectangleConstantBasis, corenc::Mesh::CTriangleBasis, corenc::Mesh::CTriangleLagrangeBasis, and corenc::Mesh::CTriangleLinearBasis.

6.44.2.3 GetNormal()

```
template<class Type >
virtual const Point corenc::Mesh::CShapeFunction< Type >::GetNormal ( ) const  [pure virtual]
```

Implemented in corenc::Mesh::CCubeBasis, corenc::Mesh::CEdgeLinearBasis, corenc::Mesh::CEdgeConstantBasis, corenc::Mesh::CEdgeMultiBasis, corenc::Mesh::CEdgeHermiteBasis, corenc::Mesh::CEdge2ndBasis, corenc::Mesh::CNodeBasis, corenc::Mesh::CRectangleBasis, corenc::Mesh::CRectangleHBasis, corenc::Mesh::CRectangleBasis2x, corenc::Mesh::CRectangleBa corenc::Mesh::CRectangleBasis2, corenc::Mesh::CRectangleConstantBasis, corenc::Mesh::CTriangleBasis, corenc::Mesh::CTriangleLagrangeBasis, corenc::Mesh::CTriangleLinearBasis, and corenc::Mesh::CTriangleBasis.

6.44.2.4 GetNumberOfShapeFunctions()

```
template<class Type >
virtual const int corenc::Mesh::CShapeFunction< Type >::GetNumberOfShapeFunctions ( ) const
[pure virtual]
```

Implemented in corenc::Mesh::CCubeBasis, corenc::Mesh::CEdgeLinearBasis, corenc::Mesh::CEdgeConstantBasis, corenc::Mesh::CEdgeMultiBasis, corenc::Mesh::CEdgeHermiteBasis, corenc::Mesh::CEdge2ndBasis, corenc::Mesh::CNodeBasis, corenc::Mesh::CRectangleBasis, corenc::Mesh::CRectangleHBasis, corenc::Mesh::CRectangleBasis2x, corenc::Mesh::CRectangleBa corenc::Mesh::CRectangleBasis2, corenc::Mesh::CRectangleConstantBasis, corenc::Mesh::CTriangleBasis, corenc::Mesh::CTriangleLagrangeBasis, corenc::Mesh::CTriangleLinearBasis, and corenc::Mesh::CTriangleBasis.

6.44.2.5 GetShapeFunction()

```
template<class Type >
virtual const double corenc::Mesh::CShapeFunction< Type >::GetShapeFunction (
            const int ,
            const Point &  ) const  [pure virtual]
```

Implemented in corenc::Mesh::CCubeBasis, corenc::Mesh::CEdgeLinearBasis, corenc::Mesh::CEdgeConstantBasis, corenc::Mesh::CEdgeMultiBasis, corenc::Mesh::CEdgeHermiteBasis, corenc::Mesh::CEdge2ndBasis, corenc::Mesh::CNodeBasis, corenc::Mesh::CRectangleBasis, corenc::Mesh::CRectangleHBasis, corenc::Mesh::CRectangleBasis2x, corenc::Mesh::CRectangleBa corenc::Mesh::CRectangleBasis2, corenc::Mesh::CRectangleConstantBasis, corenc::Mesh::CTriangleBasis, corenc::Mesh::CTriangleLagrangeBasis, corenc::Mesh::CTriangleLinearBasis, and corenc::Mesh::CTriangleBasis.

6.44.2.6 ReverseNormal()

```
template<class Type >
virtual void corenc::Mesh::CShapeFunction< Type >::ReverseNormal ( ) [pure virtual]
```

Implemented in corenc::Mesh::CCubeBasis, corenc::Mesh::CEdgeLinearBasis, corenc::Mesh::CEdgeConstantBasis, corenc::Mesh::CEdgeMultiBasis, corenc::Mesh::CEdgeHermiteBasis, corenc::Mesh::CEdge2ndBasis, corenc::Mesh::CNodeBasis, corenc::Mesh::CRectangleBasis, corenc::Mesh::CRectangleHBasis, corenc::Mesh::CRectangleBasis2x, corenc::Mesh::CRectangleB corenc::Mesh::CRectangleBasis2, corenc::Mesh::CRectangleConstantBasis, corenc::Mesh::CTriangleBasis, corenc::Mesh::CTriangleLagrangeBasis, corenc::Mesh::CTriangleLinearBasis, and corenc::Mesh::CTriangleBasis.

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

- CoreNCFEM/FiniteElements/ShapeFunction.h

6.45 Methods::CSMethod Class Reference

```
#include <CSMethod.h>
```

Public Member Functions

- CSMethod ()
- virtual ~CSMethod ()

6.45.1 Constructor & Destructor Documentation

6.45.1.1 CSMethod()

```
Methods::CSMethod::CSMethod ( ) [inline]
```

6.45.1.2 ~CSMethod()

```
virtual Methods::CSMethod::~CSMethod ( ) [inline], [virtual]
```

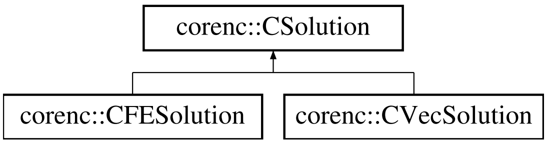
The documentation for this class was generated from the following file:

- CoreNCFEM/Methods/CSMethod.h

6.46 corenc::CSolution Class Reference

```
#include <FESolution.h>
```

Inheritance diagram for corenc::CSolution:



Public Member Functions

- CSolution ()
- virtual ∼CSolution ()

6.46.1 Constructor & Destructor Documentation

6.46.1.1 CSolution()

```
corenc::CSolution::CSolution ( )  [inline]
```

6.46.1.2 ∼CSolution()

```
virtual corenc::CSolution::~CSolution ( )  [inline], [virtual]
```

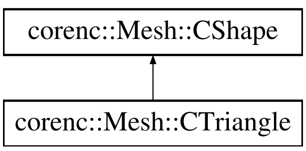
The documentation for this class was generated from the following file:

- CoreNCFEM/FESolution.h

6.47 corenc::Mesh::CTriangle Class Reference

```
#include <Triangle.h>
```

Inheritance diagram for corenc::Mesh::CTriangle:



Public Member Functions

- CTriangle ()
- CTriangle (const int n1, const int n2, const int n3, const int order)
- CTriangle (const int n1, const int n2, const int n3, const int e1, const int e2, const int e3, const int order)
- CTriangle (const int *, const int order)
- CTriangle (const int *, const int *, const int order)
- CTriangle (const CTriangle &)
- CTriangle & operator= (const CTriangle &t)
- const bool operator== (const CTriangle &t)
- std::istream & operator>> (std::istream &is)
- ∼CTriangle ()
- const int GetNode (const int) const
- const int GetNode (const NODES &) const
- const int GetEdge (const int) const
- const int GetFacet (const int) const
- const int GetNumberOfNodes () const
- const int GetNumberOfEdges () const
- const int GetNumberOfFacets () const
- const double Integrate (const std::function< const double(const Point &)> &, const std::vector< Point > &v) const
- const Point Integrate (const std::function< const Point(const Point &)> &, const std::vector< Point > &v) const
- const std::vector< double > Integrate (const std::function< const std::vector< double >(const Point &)> &, const std::vector< Point > &) const
- void SetNode (const int k, const int node)
- const int IncreaseOrder ()
- void SetEdge (const int k, const int edge)
- void SetFacet (const int k, const int facet)

6.47.1 Constructor & Destructor Documentation

6.47.1.1 CTriangle() [1/6]

```
CTriangle::CTriangle ( )
```

6.47.1.2 CTriangle() [2/6]

```
CTriangle::CTriangle (
            const int n1,
            const int n2,
            const int n3,
            const int order )
```

6.47.1.3 CTriangle() [3/6]

```
CTriangle::CTriangle (
            const int n1,
            const int n2,
            const int n3,
            const int e1,
            const int e2,
            const int e3,
            const int order )
```

6.47.1.4 CTriangle() [4/6]

```
CTriangle::CTriangle (
            const int * nodes,
            const int order )
```

6.47.1.5 CTriangle() [5/6]

```
CTriangle::CTriangle (
            const int * nodes,
            const int * edges,
            const int order )
```

6.47.1.6 CTriangle() [6/6]

```
CTriangle::CTriangle (
            const CTriangle & t )
```

6.47.1.7 ∼CTriangle()

```
corenc::Mesh::CTriangle::∼CTriangle ( ) [inline]
```

6.47.2 Member Function Documentation

6.47.2.1 GetEdge()

```
const int CTriangle::GetEdge (
            const int n ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.47.2.2 GetFacet()

```
const int CTriangle::GetFacet (
            const int  ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.47.2.3 GetNode() [1/2]

```
const int CTriangle::GetNode (
            const int n ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.47.2.4 GetNode() [2/2]

```
const int CTriangle::GetNode (
            const NODES & node ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.47.2.5 GetNumberOfEdges()

```
const int CTriangle::GetNumberOfEdges ( ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.47.2.6 GetNumberOfFacets()

```
const int CTriangle::GetNumberOfFacets ( ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.47.2.7 GetNumberOfNodes()

```
const int CTriangle::GetNumberOfNodes ( ) const [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.47.2.8 IncreaseOrder()

```
const int CTriangle::IncreaseOrder ( )
```

6.47.2.9 Integrate() [1/3]

```
const double CTriangle::Integrate (
            const std::function< const double(const Point &)> & f,
            const std::vector< Point > & v ) const
```

6.47.2.10 Integrate() [2/3]

```
const Point CTriangle::Integrate (
            const std::function< const Point(const Point &)> & f,
            const std::vector< Point > & v ) const
```

6.47.2.11 Integrate() [3/3]

```
const vector< double > CTriangle::Integrate (
            const std::function< const std::vector< double >(const Point &)> & f,
            const std::vector< Point > & v ) const [virtual]
```

Implements corenc::Mesh::CShape.

6.47.2.12 operator=()

```
CTriangle & corenc::Mesh::CTriangle::operator= (
            const CTriangle & t ) [inline]
```

6.47.2.13 operator==()

```
const bool corenc::Mesh::CTriangle::operator== (
            const CTriangle & t )  [inline]
```

6.47.2.14 operator>>()

```
std::istream & corenc::Mesh::CTriangle::operator>> (
            std::istream & is )  [inline]
```

6.47.2.15 SetEdge()

```
void CTriangle::SetEdge (
            const int k,
            const int edge )  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.47.2.16 SetFacet()

```
void CTriangle::SetFacet (
            const int k,
            const int facet )  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.47.2.17 SetNode()

```
void CTriangle::SetNode (
            const int k,
            const int node )  [virtual]
```

Implements corenc::Mesh::CShape.

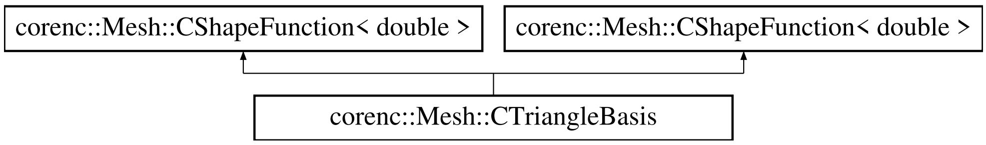
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/Triangle.h
- CoreNCFEM/FiniteElements/Triangle.cpp

6.48 corenc::Mesh::CTriangleBasis Class Reference

```
#include <Triangle.h>
```

Inheritance diagram for corenc::Mesh::CTriangleBasis:



Public Member Functions

- CTriangleBasis ()
- CTriangleBasis (const Point &, const Point &, const Point &, const int order)
- CTriangleBasis (const Point ∗, const int order)
- CTriangleBasis (const CTriangleBasis &)
- CTriangleBasis & operator= (const CTriangleBasis &t)
- ∼CTriangleBasis ()
- const int GetNumberOfShapeFunctions () const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetValue (const Point &) const
- const int IncreaseOrder ()
- const double GetMeasure () const
- const double GetWeight (const int, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const
- CTriangleBasis ()
- CTriangleBasis (const Point &, const Point &, const Point &, const int order)
- CTriangleBasis (const Point ∗, const int order)
- CTriangleBasis (const CTriangleBasis &)
- CTriangleBasis & operator= (const CTriangleBasis &t)
- ∼CTriangleBasis ()
- const int GetNumberOfShapeFunctions () const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetValue (const Point &) const

6.48.1 Constructor & Destructor Documentation

6.48.1.1 CTriangleBasis() [1/8]

```
CTriangleBasis::CTriangleBasis ( )
```

6.48.1.2 CTriangleBasis() [2/8]

```
CTriangleBasis::CTriangleBasis (
            const Point & p1,
            const Point & p2,
            const Point & p3,
            const int order )
```

6.48.1.3 CTriangleBasis() [3/8]

```
CTriangleBasis::CTriangleBasis (
            const Point * p,
            const int order )
```

6.48.1.4 CTriangleBasis() [4/8]

```
CTriangleBasis::CTriangleBasis (
            const CTriangleBasis & t )
```

6.48.1.5 ∼CTriangleBasis() [1/2]

```
corenc::Mesh::CTriangleBasis::∼CTriangleBasis ( )  [inline]
```

6.48.1.6 CTriangleBasis() [5/8]

```
corenc::Mesh::CTriangleBasis::CTriangleBasis ( )
```

6.48.1.7 CTriangleBasis() [6/8]

```
corenc::Mesh::CTriangleBasis::CTriangleBasis (
            const Point & ,
            const Point & ,
            const Point & ,
            const int order )
```

6.48.1.8 CTriangleBasis() [7/8]

```
corenc::Mesh::CTriangleBasis::CTriangleBasis (
            const Point * ,
            const int order )
```

6.48.1.9 CTriangleBasis() [8/8]

```
corenc::Mesh::CTriangleBasis::CTriangleBasis (
            const CTriangleBasis & )
```

6.48.1.10 ∼CTriangleBasis() [2/2]

```
corenc::Mesh::CTriangleBasis::∼CTriangleBasis ( ) [inline]
```

6.48.2 Member Function Documentation

6.48.2.1 GetGradShapeFunction() [1/2]

```
const Point CTriangleBasis::GetGradShapeFunction (
            const int k,
            const Point & p ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.48.2.2 GetGradShapeFunction() [2/2]

```
const Point corenc::Mesh::CTriangleBasis::GetGradShapeFunction (
            const int ,
            const Point & ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.48.2.3 GetMeasure()

```
const double corenc::Mesh::CTriangleBasis::GetMeasure ( ) const [inline], [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.48.2.4 GetNormal() [1/2]

```
const Point CTriangleBasis::GetNormal ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.48.2.5 GetNormal() [2/2]

```
const Point corenc::Mesh::CTriangleBasis::GetNormal ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.48.2.6 GetNumberOfShapeFunctions() [1/2]

```
const int CTriangleBasis::GetNumberOfShapeFunctions ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.48.2.7 GetNumberOfShapeFunctions() [2/2]

```
const int corenc::Mesh::CTriangleBasis::GetNumberOfShapeFunctions ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.48.2.8 GetShapeFunction() [1/2]

```
const double CTriangleBasis::GetShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.48.2.9 GetShapeFunction() [2/2]

```
const double corenc::Mesh::CTriangleBasis::GetShapeFunction (
            const int ,
            const Point &  ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.48.2.10 GetValue() [1/2]

```
const double CTriangleBasis::GetValue (
            const Point & p ) const
```

6.48.2.11 GetValue() [2/2]

```
const double corenc::Mesh::CTriangleBasis::GetValue (
            const Point & ) const
```

6.48.2.12 GetWeight()

```
const double CTriangleBasis::GetWeight (
            const int ,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const
```

6.48.2.13 IncreaseOrder()

```
const int CTriangleBasis::IncreaseOrder ( )
```

6.48.2.14 operator=() [1/2]

```
CTriangleBasis & corenc::Mesh::CTriangleBasis::operator= (
            const CTriangleBasis & t ) [inline]
```

6.48.2.15 operator=() [2/2]

```
CTriangleBasis & corenc::Mesh::CTriangleBasis::operator= (
            const CTriangleBasis & t ) [inline]
```

6.48.2.16 ReverseNormal() [1/2]

```
void CTriangleBasis::ReverseNormal ( ) [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.48.2.17 ReverseNormal() [2/2]

```
void corenc::Mesh::CTriangleBasis::ReverseNormal ( ) [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

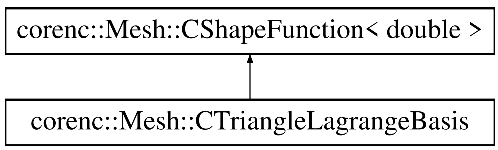
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/Triangle.h
- CoreNCFEM/FiniteElements/TriangleLinear.h
- CoreNCFEM/FiniteElements/Triangle.cpp

6.49 corenc::Mesh::CTriangleLagrangeBasis Class Reference

```
#include <Triangle.h>
```

Inheritance diagram for corenc::Mesh::CTriangleLagrangeBasis:



Public Member Functions

- CTriangleLagrangeBasis ()
- CTriangleLagrangeBasis (const Point &, const Point &, const Point &, const int order)
- CTriangleLagrangeBasis (const Point *, const int order)
- CTriangleLagrangeBasis (const CTriangleLagrangeBasis &)
- CTriangleLagrangeBasis & operator= (const CTriangleLagrangeBasis &t)
- ∼CTriangleLagrangeBasis ()
- const int GetNumberOfShapeFunctions () const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetValue (const Point &) const
- const int IncreaseOrder ()
- const double GetAlpha (const int i, const int j) const
- const double GetMeasure () const
- const double GetWeight (const int, const std::vector< Point > &verts, const std::function< const double(const Point &)> &f) const

6.49.1 Constructor & Destructor Documentation

6.49.1.1 CTriangleLagrangeBasis() [1/4]

```
CTriangleLagrangeBasis::CTriangleLagrangeBasis ( )
```

6.49.1.2 CTriangleLagrangeBasis() [2/4]

```
CTriangleLagrangeBasis::CTriangleLagrangeBasis (
            const Point & p1,
            const Point & p2,
            const Point & p3,
            const int order )
```

6.49.1.3 CTriangleLagrangeBasis() [3/4]

```
CTriangleLagrangeBasis::CTriangleLagrangeBasis (
            const Point * p,
            const int order )
```

6.49.1.4 CTriangleLagrangeBasis() [4/4]

```
CTriangleLagrangeBasis::CTriangleLagrangeBasis (
            const CTriangleLagrangeBasis & t )
```

6.49.1.5 ∼CTriangleLagrangeBasis()

```
corenc::Mesh::CTriangleLagrangeBasis::∼CTriangleLagrangeBasis ( ) [inline]
```

6.49.2 Member Function Documentation

6.49.2.1 GetAlpha()

```
const double corenc::Mesh::CTriangleLagrangeBasis::GetAlpha (
            const int i,
            const int j ) const [inline]
```

6.49.2.2 GetGradShapeFunction()

```
const Point CTriangleLagrangeBasis::GetGradShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.49.2.3 GetMeasure()

```
const double corenc::Mesh::CTriangleLagrangeBasis::GetMeasure ( ) const  [inline], [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.49.2.4 GetNormal()

```
const Point CTriangleLagrangeBasis::GetNormal ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.49.2.5 GetNumberOfShapeFunctions()

```
const int CTriangleLagrangeBasis::GetNumberOfShapeFunctions ( ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.49.2.6 GetShapeFunction()

```
const double CTriangleLagrangeBasis::GetShapeFunction (
            const int k,
            const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.49.2.7 GetValue()

```
const double CTriangleLagrangeBasis::GetValue (
            const Point & p ) const
```

6.49.2.8 GetWeight()

```
const double CTriangleLagrangeBasis::GetWeight (
            const int ,
            const std::vector< Point > & verts,
            const std::function< const double(const Point &)> & f ) const
```

6.49.2.9 IncreaseOrder()

```
const int CTriangleLagrangeBasis::IncreaseOrder ( )
```

6.49.2.10 operator=()

```
CTriangleLagrangeBasis & corenc::Mesh::CTriangleLagrangeBasis::operator= (
            const CTriangleLagrangeBasis & t ) [inline]
```

6.49.2.11 ReverseNormal()

```
void CTriangleLagrangeBasis::ReverseNormal ( ) [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

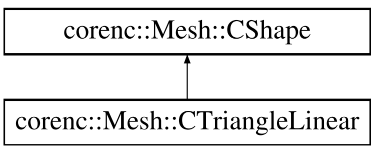
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/Triangle.h
- CoreNCFEM/FiniteElements/TriangleLagrange.cpp

6.50 corenc::Mesh::CTriangleLinear Class Reference

```
#include <TriangleLinear.h>
```

Inheritance diagram for corenc::Mesh::CTriangleLinear:



Public Member Functions

- CTriangleLinear ()
- CTriangleLinear (const int n1, const int n2, const int n3)
- CTriangleLinear (const int n1, const int n2, const int n3, const int e1, const int e2, const int e3)
- CTriangleLinear (const int *)
- CTriangleLinear (const int *, const int *)
- CTriangleLinear (const CTriangleLinear &)
- CTriangleLinear & operator= (const CTriangleLinear &t)
- const bool operator== (const CTriangleLinear &t)
- std::istream & operator>> (std::istream &is)
- ~CTriangleLinear ()
- const int GetNode (const int) const
- const int GetNode (const NODES &) const
- const int GetEdge (const int) const
- const int GetFacet (const int) const
- const int GetNumberOfNodes () const
- const int GetNumberOfEdges () const
- const int GetNumberOfFacets () const
- const double Integrate (const std::function< const double(const Point &)> &, const std::vector< Point > &v) const
- const Point Integrate (const std::function< const Point(const Point &)> &, const std::vector< Point > &v) const
- const std::vector< double > Integrate (const std::function< const std::vector< double >(const Point &)> &, const std::vector< Point > &) const
- void SetNode (const int k, const int node)
- const int IncreaseOrder ()
- void SetEdge (const int k, const int edge)
- void SetFacet (const int k, const int facet)

6.50.1 Constructor & Destructor Documentation

6.50.1.1 CTriangleLinear() [1/6]

```
CTriangleLinear::CTriangleLinear ( )
```

6.50.1.2 CTriangleLinear() [2/6]

```
CTriangleLinear::CTriangleLinear (
            const int n1,
            const int n2,
            const int n3 )
```

6.50.1.3 CTriangleLinear() [3/6]

```
CTriangleLinear::CTriangleLinear (
            const int n1,
            const int n2,
            const int n3,
            const int e1,
            const int e2,
            const int e3 )
```

6.50.1.4 CTriangleLinear() [4/6]

```
CTriangleLinear::CTriangleLinear (
            const int * nodes )
```

6.50.1.5 CTriangleLinear() [5/6]

```
CTriangleLinear::CTriangleLinear (
            const int * nodes,
            const int * edges )
```

6.50.1.6 CTriangleLinear() [6/6]

```
CTriangleLinear::CTriangleLinear (
            const CTriangleLinear & t )
```

6.50.1.7 ∼CTriangleLinear()

```
corenc::Mesh::CTriangleLinear::∼CTriangleLinear ( )  [inline]
```

6.50.2 Member Function Documentation

6.50.2.1 GetEdge()

```
const int CTriangleLinear::GetEdge (
            const int n ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.50.2.2 GetFacet()

```
const int CTriangleLinear::GetFacet (
            const int  ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.50.2.3 GetNode() [1/2]

```
const int CTriangleLinear::GetNode (
            const int n ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.50.2.4 GetNode() [2/2]

```
const int CTriangleLinear::GetNode (
            const NODES & node ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.50.2.5 GetNumberOfEdges()

```
const int CTriangleLinear::GetNumberOfEdges ( ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.50.2.6 GetNumberOfFacets()

```
const int CTriangleLinear::GetNumberOfFacets ( ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.50.2.7 GetNumberOfNodes()

```
const int CTriangleLinear::GetNumberOfNodes ( ) const  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.50.2.8 IncreaseOrder()

```
const int corenc::Mesh::CTriangleLinear::IncreaseOrder ( )  [inline]
```

6.50.2.9 Integrate() [1/3]

```
const double CTriangleLinear::Integrate (
            const std::function< const double(const Point &)> & f,
            const std::vector< Point > & v ) const
```

6.50.2.10 Integrate() [2/3]

```
const Point CTriangleLinear::Integrate (
            const std::function< const Point(const Point &)> & f,
            const std::vector< Point > & v ) const
```

6.50.2.11 Integrate() [3/3]

```
const vector< double > CTriangleLinear::Integrate (
            const std::function< const std::vector< double >(const Point &)> & f,
            const std::vector< Point > & v ) const  [virtual]
```

Implements corenc::Mesh::CShape.

6.50.2.12 operator=()

```
CTriangleLinear & corenc::Mesh::CTriangleLinear::operator= (
            const CTriangleLinear & t )  [inline]
```

6.50.2.13 operator==()

```
const bool corenc::Mesh::CTriangleLinear::operator== (
            const CTriangleLinear & t )  [inline]
```

6.50.2.14 operator>>()

```
std::istream & corenc::Mesh::CTriangleLinear::operator>> (
            std::istream & is )  [inline]
```

6.50.2.15 SetEdge()

```
void CTriangleLinear::SetEdge (
            const int k,
            const int edge )  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.50.2.16 SetFacet()

```
void CTriangleLinear::SetFacet (
            const int k,
            const int facet )  [virtual]
```

Reimplemented from corenc::Mesh::CShape.

6.50.2.17 SetNode()

```
void CTriangleLinear::SetNode (
            const int k,
            const int node )  [virtual]
```

Implements corenc::Mesh::CShape.

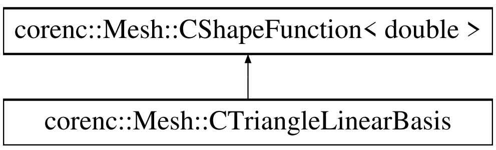
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/TriangleLinear.h
- CoreNCFEM/FiniteElements/TriangleLinear.cpp

6.51 corenc::Mesh::CTriangleLinearBasis Class Reference

```
#include <TriangleLinear.h>
```

Inheritance diagram for corenc::Mesh::CTriangleLinearBasis:



Public Member Functions

- CTriangleLinearBasis ()
- CTriangleLinearBasis (const Point &, const Point &, const Point &)
- CTriangleLinearBasis (const Point ∗)
- CTriangleLinearBasis (const CTriangleLinearBasis &)
- CTriangleLinearBasis & operator= (const CTriangleLinearBasis &t)
- ∼CTriangleLinearBasis ()
- const int GetNumberOfShapeFunctions () const
- const double GetShapeFunction (const int, const Point &) const
- const Point GetGradShapeFunction (const int, const Point &) const
- const Point GetNormal () const
- void ReverseNormal ()
- const double GetValue (const Point &) const
- const int IncreaseOrder ()
- const double GetMeasure () const

6.51.1 Constructor & Destructor Documentation

6.51.1.1 CTriangleLinearBasis() [1/4]

```
CTriangleLinearBasis::CTriangleLinearBasis ( )
```

6.51.1.2 CTriangleLinearBasis() [2/4]

```
CTriangleLinearBasis::CTriangleLinearBasis (
            const Point & p1,
            const Point & p2,
            const Point & p3 )
```

6.51.1.3 CTriangleLinearBasis() [3/4]

```
CTriangleLinearBasis::CTriangleLinearBasis (
            const Point * p )
```

6.51.1.4 CTriangleLinearBasis() [4/4]

```
CTriangleLinearBasis::CTriangleLinearBasis (
            const CTriangleLinearBasis & t )
```

6.51.1.5 ∼CTriangleLinearBasis()

```
corenc::Mesh::CTriangleLinearBasis::~CTriangleLinearBasis ( ) [inline]
```

6.51.2 Member Function Documentation

6.51.2.1 GetGradShapeFunction()

```
const Point CTriangleLinearBasis::GetGradShapeFunction (
            const int k,
            const Point & ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.51.2.2 GetMeasure()

```
const double corenc::Mesh::CTriangleLinearBasis::GetMeasure ( ) const [inline], [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.51.2.3 GetNormal()

```
const Point CTriangleLinearBasis::GetNormal ( ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.51.2.4 GetNumberOfShapeFunctions()

```
const int CTriangleLinearBasis::GetNumberOfShapeFunctions ( ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.51.2.5 GetShapeFunction()

```
const double CTriangleLinearBasis::GetShapeFunction (
            const int k,
            const Point & p ) const [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

6.51.2.6 GetValue()

```
const double CTriangleLinearBasis::GetValue (
            const Point & p ) const
```

6.51.2.7 IncreaseOrder()

```
const int corenc::Mesh::CTriangleLinearBasis::IncreaseOrder ( )  [inline]
```

6.51.2.8 operator=()

```
CTriangleLinearBasis & corenc::Mesh::CTriangleLinearBasis::operator= (
            const CTriangleLinearBasis & t )  [inline]
```

6.51.2.9 ReverseNormal()

```
void CTriangleLinearBasis::ReverseNormal ( )  [virtual]
```

Implements corenc::Mesh::CShapeFunction< double >.

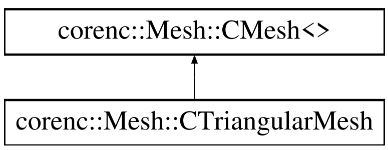
The documentation for this class was generated from the following files:

- CoreNCFEM/FiniteElements/TriangleLinear.h
- CoreNCFEM/FiniteElements/TriangleLinear.cpp

6.52 corenc::Mesh::CTriangularMesh Class Reference

```
#include <TriangularMesh.h>
```

Inheritance diagram for corenc::Mesh::CTriangularMesh:



Public Member Functions

- CTriangularMesh ()
- CTriangularMesh (const std::string &file_name)
- CTriangularMesh (const CTriangularMesh &)
- CTriangularMesh (const Point &p1, const Point &p2, const int nx, const int ny)
- CTriangularMesh & operator= (const CTriangularMesh &tr)
- CTriangularMesh ∗ Clone () const
- const unsigned int GetNumberOfElements () const
- const unsigned int GetNumberOfNodes () const
- const unsigned int GetNumberOfBoundaries () const
- const int FindElement (const Point &) const
- const Point GetNode (const unsigned int) const
- const CElement ∗ GetElement (const unsigned int) const
- const CElement ∗ GetBoundary (const unsigned int) const
- const double getMinSize () const
- const double getSolution (const unsigned int element, const unsigned int node) const
- const int updateSolution (const unsigned int element, const unsigned int node, const double value)
- const std::vector< double > getSolution () const
- const int updateSolution (const std::vector< double > &)
- const int updateSolution (const unsigned int element, const unsigned int node, CSolution ∗value)
- const double getParameter (Parameters, const unsigned int, const Point &) const
- const double getParameter (Parameters, const unsigned int, const int) const
- const int setParameter (Parameters, const double, const unsigned int)
- const int setParameter (const CParameter &, const unsigned int type)
- const int updateSolution (const unsigned int node, const double value)
- const int refine_h ()
- const int refine_p ()
- const int refine_hp ()
- const int set4thOrder ()
- const int set2ndOrder ()
- const int set3rdOrder ()
- const int interpolate (const int node) const
- const int GetNumberOfINodes () const
- ∼CTriangularMesh ()
- auto GetElements () -> decltype(m_elems)
- auto GetBoundary () -> decltype(m_edges)

6.52.1 Constructor & Destructor Documentation

6.52.1.1 CTriangularMesh() [1/4]

```
CTriangularMesh::CTriangularMesh ( )
```

6.52.1.2 CTriangularMesh() [2/4]

```
corenc::Mesh::CTriangularMesh::CTriangularMesh (
            const std::string & file_name )
```

6.52.1.3 CTriangularMesh() [3/4]

```
CTriangularMesh::CTriangularMesh (
            const CTriangularMesh & tr )
```

6.52.1.4 CTriangularMesh() [4/4]

```
CTriangularMesh::CTriangularMesh (
            const Point & p1,
            const Point & p2,
            const int nx,
            const int ny )
```

6.52.1.5 ∼CTriangularMesh()

```
CTriangularMesh::~CTriangularMesh ( )
```

6.52.2 Member Function Documentation

6.52.2.1 Clone()

```
CTriangularMesh * corenc::Mesh::CTriangularMesh::Clone ( ) const  [inline]
```

6.52.2.2 FindElement()

```
const int CTriangularMesh::FindElement (
            const Point & test ) const  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.52.2.3 GetBoundary() [1/2]

```
auto corenc::Mesh::CTriangularMesh::GetBoundary ( ) -> decltype(m_edges)   [inline]
```

6.52.2.4 GetBoundary() [2/2]

```
const CElement * CTriangularMesh::GetBoundary (
             const unsigned int n ) const  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.52.2.5 GetElement()

```
const CElement * CTriangularMesh::GetElement (
             const unsigned int n ) const  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.52.2.6 GetElements()

```
auto corenc::Mesh::CTriangularMesh::GetElements ( ) -> decltype(m_elems)   [inline]
```

6.52.2.7 getMinSize()

```
const double corenc::Mesh::CTriangularMesh::getMinSize ( ) const  [inline], [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.52.2.8 GetNode()

```
const Mesh::Point CTriangularMesh::GetNode (
             const unsigned int n ) const  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.52.2.9 GetNumberOfBoundaries()

```
const unsigned int CTriangularMesh::GetNumberOfBoundaries ( ) const  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.52.2.10 GetNumberOfElements()

```
const unsigned int CTriangularMesh::GetNumberOfElements ( ) const  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.52.2.11 GetNumberOfINodes()

```
const int CTriangularMesh::GetNumberOfINodes ( ) const
```

6.52.2.12 GetNumberOfNodes()

```
const unsigned int CTriangularMesh::GetNumberOfNodes ( ) const  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.52.2.13 getParameter() [1/2]

```
const double CTriangularMesh::getParameter (
             Parameters param,
             const unsigned int l,
             const int i ) const  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.52.2.14 getParameter() [2/2]

```
const double CTriangularMesh::getParameter (
             Parameters param,
             const unsigned int l,
             const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.52.2.15 getSolution() [1/2]

```
const std::vector< double > CTriangularMesh::getSolution ( ) const [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.52.2.16 getSolution() [2/2]

```
const double CTriangularMesh::getSolution (
            const unsigned int element,
            const unsigned int node ) const [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.52.2.17 interpolate()

```
const int CTriangularMesh::interpolate (
            const int node ) const
```

6.52.2.18 operator=()

```
CTriangularMesh & corenc::Mesh::CTriangularMesh::operator= (
            const CTriangularMesh & tr ) [inline]
```

6.52.2.19 refine_h()

```
const int CTriangularMesh::refine_h ( )
```

6.52.2.20 refine_hp()

```
const int corenc::Mesh::CTriangularMesh::refine_hp ( )
```

6.52.2.21 refine_p()

```
const int CTriangularMesh::refine_p ( )
```

6.52.2.22 set2ndOrder()

```
const int CTriangularMesh::set2ndOrder ( )
```

6.52.2.23 set3rdOrder()

```
const int CTriangularMesh::set3rdOrder ( )
```

6.52.2.24 set4thOrder()

```
const int CTriangularMesh::set4thOrder ( )
```

6.52.2.25 setParameter() [1/2]

```
const int CTriangularMesh::setParameter (
            const CParameter & p,
            const unsigned int type )
```

6.52.2.26 setParameter() [2/2]

```
const int CTriangularMesh::setParameter (
            Parameters param,
            const double ,
            const unsigned int )   [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.52.2.27 updateSolution() [1/4]

```
const int CTriangularMesh::updateSolution (
            const std::vector< double > & )   [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.52.2.28 updateSolution() [2/4]

```
const int CTriangularMesh::updateSolution (
            const unsigned int element,
            const unsigned int node,
            const double value ) [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.52.2.29 updateSolution() [3/4]

```
const int CTriangularMesh::updateSolution (
            const unsigned int element,
            const unsigned int node,
            CSolution * value ) [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.52.2.30 updateSolution() [4/4]

```
const int CTriangularMesh::updateSolution (
            const unsigned int node,
            const double value ) [virtual]
```

Implements corenc::Mesh::CMesh<>.

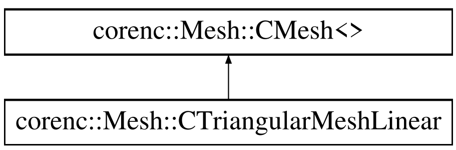
The documentation for this class was generated from the following files:

- CoreNCFEM/Grids/TriangularMesh.h
- CoreNCFEM/Grids/TriangularMesh.cpp

6.53 corenc::Mesh::CTriangularMeshLinear Class Reference

```
#include <TriangularMeshLinear.h>
```

Inheritance diagram for corenc::Mesh::CTriangularMeshLinear:



Public Member Functions

- CTriangularMeshLinear ()
- CTriangularMeshLinear (const std::string &file_name)
- CTriangularMeshLinear (const CTriangularMeshLinear &)
- const unsigned int GetNumberOfElements () const
- const unsigned int GetNumberOfNodes () const
- const unsigned int GetNumberOfBoundaries () const
- const int FindElement (const Point &) const
- const Point GetNode (const unsigned int) const
- const CElement ∗ GetElement (const unsigned int) const
- const CElement ∗ GetBoundary (const unsigned int) const
- const double getMinSize () const
- const double getSolution (const unsigned int element, const unsigned int node) const
- const int updateSolution (const unsigned int element, const unsigned int node, const double value)
- const std::vector< double > getSolution () const
- const int updateSolution (const std::vector< double > &)
- const int updateSolution (const unsigned int element, const unsigned int node, CSolution ∗value)
- const double getParameter (Parameters, const unsigned int, const Point &) const
- const double getParameter (Parameters, const unsigned int, const int) const
- const int setParameter (Parameters, const double, const unsigned int)
- const int setParameter (const CParameter &, const unsigned int type)
- const int updateSolution (const unsigned int node, const double value)
- const int refine_h ()
- ∼CTriangularMeshLinear ()
- auto GetElements () -> decltype(m_elems)
- auto GetBoundary () -> decltype(m_edges)

6.53.1 Constructor & Destructor Documentation

6.53.1.1 CTriangularMeshLinear() [1/3]

```
CTriangularMeshLinear::CTriangularMeshLinear ( )
```

6.53.1.2 CTriangularMeshLinear() [2/3]

```
corenc::Mesh::CTriangularMeshLinear::CTriangularMeshLinear (
            const std::string & file_name )
```

6.53.1.3 CTriangularMeshLinear() [3/3]

```
corenc::Mesh::CTriangularMeshLinear::CTriangularMeshLinear (
            const CTriangularMeshLinear & )
```

6.53.1.4 ∼CTriangularMeshLinear()

```
CTriangularMeshLinear::∼CTriangularMeshLinear ( )
```

6.53.2 Member Function Documentation

6.53.2.1 FindElement()

```
const int CTriangularMeshLinear::FindElement (
            const Point & ) const [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.53.2.2 GetBoundary() [1/2]

```
auto corenc::Mesh::CTriangularMeshLinear::GetBoundary ( ) -> decltype(m_edges) [inline]
```

6.53.2.3 GetBoundary() [2/2]

```
const CElement * CTriangularMeshLinear::GetBoundary (
            const unsigned int n ) const [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.53.2.4 GetElement()

```
const CElement * CTriangularMeshLinear::GetElement (
            const unsigned int n ) const [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.53.2.5 GetElements()

```
auto corenc::Mesh::CTriangularMeshLinear::GetElements ( ) -> decltype(m_elems) [inline]
```

6.53.2.6 getMinSize()

```
const double corenc::Mesh::CTriangularMeshLinear::getMinSize ( ) const  [inline], [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.53.2.7 GetNode()

```
const Mesh::Point CTriangularMeshLinear::GetNode (
            const unsigned int n ) const  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.53.2.8 GetNumberOfBoundaries()

```
const unsigned int CTriangularMeshLinear::GetNumberOfBoundaries ( ) const  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.53.2.9 GetNumberOfElements()

```
const unsigned int CTriangularMeshLinear::GetNumberOfElements ( ) const  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.53.2.10 GetNumberOfNodes()

```
const unsigned int CTriangularMeshLinear::GetNumberOfNodes ( ) const  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.53.2.11 getParameter() [1/2]

```
const double CTriangularMeshLinear::getParameter (
            Parameters param,
            const unsigned int l,
            const int i ) const  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.53.2.12 getParameter() [2/2]

```
const double CTriangularMeshLinear::getParameter (
             Parameters param,
             const unsigned int l,
             const Point & p ) const  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.53.2.13 getSolution() [1/2]

```
const std::vector< double > CTriangularMeshLinear::getSolution ( ) const  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.53.2.14 getSolution() [2/2]

```
const double CTriangularMeshLinear::getSolution (
             const unsigned int element,
             const unsigned int node ) const  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.53.2.15 refine_h()

```
const int CTriangularMeshLinear::refine_h ( )
```

6.53.2.16 setParameter() [1/2]

```
const int CTriangularMeshLinear::setParameter (
             const CParameter & p,
             const unsigned int type )
```

6.53.2.17 setParameter() [2/2]

```
const int CTriangularMeshLinear::setParameter (
             Parameters param,
             const double ,
             const unsigned int  )  [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.53.2.18 updateSolution() [1/4]

```
const int CTriangularMeshLinear::updateSolution (
            const std::vector< double > & ) [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.53.2.19 updateSolution() [2/4]

```
const int CTriangularMeshLinear::updateSolution (
            const unsigned int element,
            const unsigned int node,
            const double value ) [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.53.2.20 updateSolution() [3/4]

```
const int CTriangularMeshLinear::updateSolution (
            const unsigned int element,
            const unsigned int node,
            CSolution * value ) [virtual]
```

Implements corenc::Mesh::CMesh<>.

6.53.2.21 updateSolution() [4/4]

```
const int CTriangularMeshLinear::updateSolution (
            const unsigned int node,
            const double value ) [virtual]
```

Implements corenc::Mesh::CMesh<>.

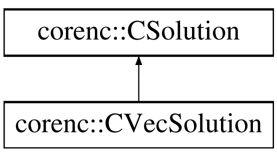
The documentation for this class was generated from the following files:

- CoreNCFEM/Grids/TriangularMeshLinear.h
- CoreNCFEM/Grids/TriangularMeshLinear.cpp

6.54 corenc::CVecSolution Class Reference

`#include <FESolution.h>`

Inheritance diagram for corenc::CVecSolution:



Public Member Functions

- CVecSolution ()
- ∼CVecSolution ()

Public Attributes

- std::vector< double > m_w

6.54.1 Constructor & Destructor Documentation

6.54.1.1 CVecSolution()

```
corenc::CVecSolution::CVecSolution ( ) [inline]
```

6.54.1.2 ∼CVecSolution()

```
corenc::CVecSolution::∼CVecSolution ( ) [inline]
```

6.54.2 Member Data Documentation

6.54.2.1 m_w

```
std::vector<double> corenc::CVecSolution::m_w
```

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

- CoreNCFEM/FESolution.h

6.55 corenc::solvers::dg_shallow_water< Mesh > Class Template Reference

```
#include <dg_solver_shallow_water.h>
```

Public Member Functions

- dg_shallow_water ()
- ∼dg_shallow_water ()
- const int solve (const double t0, const double t1, const Mesh &mesh, vector_solution &sol, const std↩ ::function< const std::vector< double >(const std::vector< double > &)> &, const std::function< const std::vector< double >(const std::vector< double > &)> &, const std::function< const std::vector< double >(const std::vector< double > &)> &) const
- const int solve (const double t0, const double t1, const Mesh &mesh, vector_solution &sol, std::vector< double > &bath, std::vector< double > &ze, std::vector< double > &dzx, std::vector< double > &dzy, std::vector< double > &dbx, std::vector< double > &dby, const std::function< const std::vector< double >(const std::vector< double > &, const int)> &, const std::function< const std::vector< double >(const std::vector< double > &, const int)> &, const std::function< const std::vector< double >(const std::vector< double > &, const int)> &, const bool WRITE_FILE) const

6.55.1 Constructor & Destructor Documentation

6.55.1.1 dg_shallow_water()

```
template<class Mesh >
corenc::solvers::dg_shallow_water< Mesh >::dg_shallow_water
```

6.55.1.2 ∼dg_shallow_water()

```
template<class Mesh >
corenc::solvers::dg_shallow_water< Mesh >::∼dg_shallow_water
```

6.55.2 Member Function Documentation

6.55.2.1 solve() [1/2]

```
template<class Mesh >
const int corenc::solvers::dg_shallow_water< Mesh >::solve (
            const double t0,
            const double t1,
            const Mesh & mesh,
            vector_solution & sol,
            const std::function< const std::vector< double >(const std::vector< double >
&)> & R,
            const std::function< const std::vector< double >(const std::vector< double >
&)> & G,
            const std::function< const std::vector< double >(const std::vector< double >
&)> & F ) const
```

6.55.2.2 solve() [2/2]

```
template<class Mesh >
const int corenc::solvers::dg_shallow_water< Mesh >::solve (
            const double t0,
            const double t1,
            const Mesh & mesh,
            vector_solution & sol,
            std::vector< double > & bath,
            std::vector< double > & ze,
            std::vector< double > & dzx,
            std::vector< double > & dzy,
            std::vector< double > & dbx,
            std::vector< double > & dby,
            const std::function< const std::vector< double >(const std::vector< double > &,
const int)> & R,
            const std::function< const std::vector< double >(const std::vector< double > &,
const int)> & G,
            const std::function< const std::vector< double >(const std::vector< double > &,
const int)> & F,
            const bool WRITE_FILE ) const
```

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

- Solvers/dg_solver_shallow_water.h

6.56 corenc::solvers::dg_solver< _Problem, _Mesh, _Result > Class Template Reference

```
#include <dg_solver.h>
```

Public Member Functions

- dg_solver ()
- ∼dg_solver ()
- const int elliptic_solver (_Problem ∗, _Mesh ∗, _Result ∗)
- const double get_value (const _Mesh &, const _Result &, const Mesh::Point &p) const
- const double get_value (const _Method ∗, const _Mesh &, const _Result &, const Mesh::Point &p) const
- const double get_value (const _Mesh &, const _Result &, const Mesh::Point &p, const int i) const
- const Mesh::Point get_gradvalue (const _Mesh &, const _Result &, const Mesh::Point &p) const
- const Mesh::Point get_gradvalue (const _Mesh &, const _Result &, const Mesh::Point &p, const int i) const

6.56.1 Constructor & Destructor Documentation

6.56.1.1 dg_solver()

```
template<class _Problem , class _Mesh , class _Result >
corenc::solvers::dg_solver< _Problem, _Mesh, _Result >::dg_solver ( ) [inline]
```

6.56.1.2 ∼dg_solver()

```
template<class _Problem , class _Mesh , class _Result >
corenc::solvers::dg_solver< _Problem, _Mesh, _Result >::∼dg_solver ( )  [inline]
```

6.56.2 Member Function Documentation

6.56.2.1 elliptic_solver()

```
template<class _Problem , class _Mesh , class _Result >
const int corenc::solvers::dg_solver< _Problem, _Mesh, _Result >::elliptic_solver (
            _Problem * problem,
            _Mesh * mesh,
            _Result * result )
```

6.56.2.2 get_gradvalue() [1/2]

```
template<class _Problem , class _Mesh , class _Result >
const Mesh::Point corenc::solvers::dg_solver< _Problem, _Mesh, _Result >::get_gradvalue (
            const _Mesh & mesh,
            const _Result & res,
            const Mesh::Point & p ) const
```

6.56.2.3 get_gradvalue() [2/2]

```
template<class _Problem , class _Mesh , class _Result >
const Mesh::Point corenc::solvers::dg_solver< _Problem, _Mesh, _Result >::get_gradvalue (
            const _Mesh & mesh,
            const _Result & res,
            const Mesh::Point & p,
            const int i ) const
```

6.56.2.4 get_value() [1/3]

```
template<class _Problem , class _Mesh , class _Result >
const double corenc::solvers::dg_solver< _Problem, _Mesh, _Result >::get_value (
            const _Mesh & mesh,
            const _Result & res,
            const Mesh::Point & p ) const
```

6.56.2.5 get_value() [2/3]

```
template<class _Problem , class _Mesh , class _Result >
const double corenc::solvers::dg_solver< _Problem, _Mesh, _Result >::get_value (
            const _Mesh & mesh,
            const _Result & res,
            const Mesh::Point & p,
            const int i ) const
```

6.56.2.6 get_value() [3/3]

```
template<class _Problem , class _Mesh , class _Result >
const double corenc::solvers::dg_solver< _Problem, _Mesh, _Result >::get_value (
            const _Method * method2,
            const _Mesh & mesh,
            const _Result & res,
            const Mesh::Point & p ) const
```

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

- Solvers/dg_solver.h

6.57 corenc::solvers::dg_solver_shallow_water Class Reference

```
#include <dg_solver_shallow_water.h>
```

Public Member Functions

- dg_solver_shallow_water ()
- ∼dg_solver_shallow_water ()
- const int solve () const
- const int solve (const double t0, const double t1, const size_t nx, const size_t ny, const double x0, const double x1, const double y0, const double y1, const double g, const double H, const std::function< const std::vector< double >(const std::vector< double > &)> &, const std::function< const std::vector< double >(const std::vector< double > &)> &, const std::function< const std::vector< double >(const std::vector< double > &)> &) const

6.57.1 Constructor & Destructor Documentation

6.57.1.1 dg_solver_shallow_water()

```
dg_solver_shallow_water::dg_solver_shallow_water ( )
```

6.57.1.2 ∼dg_solver_shallow_water()

```
dg_solver_shallow_water::~dg_solver_shallow_water ( )
```

6.57.2 Member Function Documentation

6.57.2.1 solve() [1/2]

```
const int dg_solver_shallow_water::solve ( ) const
```

6.57.2.2 solve() [2/2]

```
const int corenc::solvers::dg_solver_shallow_water::solve (
            const double t0,
            const double t1,
            const size_t nx,
            const size_t ny,
            const double x0,
            const double x1,
            const double y0,
            const double y1,
            const double g,
            const double H,
            const std::function< const std::vector< double >(const std::vector< double >
&)> & ,
            const std::function< const std::vector< double >(const std::vector< double >
&)> & ,
            const std::function< const std::vector< double >(const std::vector< double >
&)> &  ) const
```

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

- Solvers/dg_solver_shallow_water.h
- Solvers/dg_solver_shallow_water.cpp

6.58 corenc::method::DGMethod< Problem, Grid, Matrix > Class Template Reference

```
#include <DGMethod.h>
```

Public Member Functions

- DGMethod ()
- DGMethod (Problem ∗p, Grid ∗g, Matrix ∗m, std::vector< double > ∗rhs)
- DGMethod (Problem ∗p, Grid ∗g, Matrix ∗m, Matrix ∗rm, std::vector< double > ∗rhs)
- DGMethod (const std::shared_ptr< Grid > &grid)
- DGMethod (Grid ∗grid)
- DGMethod (const DGMethod &meth)
- void Discretization ()
- const double GetValue (const Mesh::Point &) const
- const double GetValue (const Mesh::Point &, const std::vector< double > &vec) const
- const double GetValue (const Mesh::Point &, const std::vector< double > &vec, const int num) const
- const double GetEffective (const std::vector< double > &vec) const
- void ProjectSolution (std::vector< double > &, std::function< const double(const Mesh::Point &, const std↩ ::vector< double > &, const int)> GetValue, std::vector< double > &sol)
- void ProjectSolution (std::vector< double > &, std::function< const double(const Mesh::Point &, const std↩ ::vector< double > &)> GetValue, std::vector< double > &sol, const int)
- void LoadSolution (const std::vector< double > &vec)
- const std::vector< double > SetSolution (const int sol, const int liq, const double, const double, const double)
- void GetSolution (std::vector< double > &vec)
- void Rediscretization (const std::shared_ptr< Grid > &)
- void Rediscretization ()
- void SetTimeStep (const double &step)
- Matrix ∗ GetGlobalMatrix () const
- Grid ∗ GetMesh ()
- const std::vector< double > GetRightVector () const
- void OutDatFormat (const Mesh::Point &min, const Mesh::Point &max, const std::string &file_name, const std::vector< double > &vec) const
- void OutMeshFormat (const std::string &file_name, const std::vector< double > &vec)
- void OutMeshTimeFormat (const std::string &file_name, const std::vector< double > &vec)
- ∼DGMethod ()

Static Public Member Functions

- static const double GetSolution (const Grid &g, const std::vector< double > &weights, const Mesh::Point &p)
- static const double GetSolution (const Grid &g, const std::vector< double > &weights, const Mesh::Point &p, const int nfem)
- static const Mesh::Point GetGradSolution (const Grid &g, const std::vector< double > &weights, const Mesh::Point &p)
- static const Mesh::Point GetGradSolution (const Grid &g, const std::vector< double > &weights, const Mesh::Point &p, const int n)

6.58.1 Constructor & Destructor Documentation

6.58.1.1 DGMethod() [1/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::DGMethod< Problem, Grid, Matrix >::DGMethod ( )  [inline]
```

6.58.1.2 DGMethod() [2/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::DGMethod< Problem, Grid, Matrix >::DGMethod (
            Problem * p,
            Grid * g,
            Matrix * m,
            std::vector< double > * rhs )  [inline]
```

6.58.1.3 DGMethod() [3/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::DGMethod< Problem, Grid, Matrix >::DGMethod (
            Problem * p,
            Grid * g,
            Matrix * m,
            Matrix * rm,
            std::vector< double > * rhs )  [inline]
```

6.58.1.4 DGMethod() [4/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::DGMethod< Problem, Grid, Matrix >::DGMethod (
            const std::shared_ptr< Grid > & grid )  [inline]
```

6.58.1.5 DGMethod() [5/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::DGMethod< Problem, Grid, Matrix >::DGMethod (
            Grid * grid )  [inline]
```

6.58.1.6 DGMethod() [6/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::DGMethod< Problem, Grid, Matrix >::DGMethod (
            const DGMethod< Problem, Grid, Matrix > & meth )  [inline]
```

6.58.1.7 ∼DGMethod()

```
template<class Problem , class Grid , class Matrix >
corenc::method::DGMethod< Problem, Grid, Matrix >::~DGMethod
```

6.58.2 Member Function Documentation

6.58.2.1 Discretization()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethod< Problem, Grid, Matrix >::Discretization
```

6.58.2.2 GetEffective()

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::DGMethod< Problem, Grid, Matrix >::GetEffective (
            const std::vector< double > & vec ) const
```

6.58.2.3 GetGlobalMatrix()

```
template<class Problem , class Grid , class Matrix >
Matrix * corenc::method::DGMethod< Problem, Grid, Matrix >::GetGlobalMatrix
```

6.58.2.4 GetGradSolution() [1/2]

```
template<class Problem , class Grid , class Matrix >
const Mesh::Point corenc::method::DGMethod< Problem, Grid, Matrix >::GetGradSolution (
            const Grid & g,
            const std::vector< double > & weights,
            const Mesh::Point & p ) [static]
```

6.58.2.5 GetGradSolution() [2/2]

```
template<class Problem , class Grid , class Matrix >
const Mesh::Point corenc::method::DGMethod< Problem, Grid, Matrix >::GetGradSolution (
            const Grid & g,
            const std::vector< double > & weights,
            const Mesh::Point & p,
            const int n ) [static]
```

6.58.2.6 GetMesh()

```
template<class Problem , class Grid , class Matrix >
Grid * corenc::method::DGMethod< Problem, Grid, Matrix >::GetMesh ( )  [inline]
```

6.58.2.7 GetRightVector()

```
template<class Problem , class Grid , class Matrix >
const std::vector< double > corenc::method::DGMethod< Problem, Grid, Matrix >::GetRightVector
```

6.58.2.8 GetSolution() [1/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::DGMethod< Problem, Grid, Matrix >::GetSolution (
            const Grid & g,
            const std::vector< double > & weights,
            const Mesh::Point & p )  [static]
```

6.58.2.9 GetSolution() [2/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::DGMethod< Problem, Grid, Matrix >::GetSolution (
            const Grid & g,
            const std::vector< double > & weights,
            const Mesh::Point & p,
            const int nfem )  [static]
```

6.58.2.10 GetSolution() [3/3]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethod< Problem, Grid, Matrix >::GetSolution (
            std::vector< double > & vec )
```

6.58.2.11 GetValue() [1/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::DGMethod< Problem, Grid, Matrix >::GetValue (
            const Mesh::Point & p ) const
```

6.58.2.12 GetValue() [2/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::DGMethod< Problem, Grid, Matrix >::GetValue (
            const Mesh::Point & p,
            const std::vector< double > & vec ) const
```

6.58.2.13 GetValue() [3/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::DGMethod< Problem, Grid, Matrix >::GetValue (
            const Mesh::Point & p,
            const std::vector< double > & vec,
            const int num ) const
```

6.58.2.14 LoadSolution()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethod< Problem, Grid, Matrix >::LoadSolution (
            const std::vector< double > & vec )
```

6.58.2.15 OutDatFormat()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethod< Problem, Grid, Matrix >::OutDatFormat (
            const Mesh::Point & min,
            const Mesh::Point & max,
            const std::string & file_name,
            const std::vector< double > & vec ) const
```

6.58.2.16 OutMeshFormat()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethod< Problem, Grid, Matrix >::OutMeshFormat (
            const std::string & file_name,
            const std::vector< double > & vec )
```

6.58.2.17 OutMeshTimeFormat()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethod< Problem, Grid, Matrix >::OutMeshTimeFormat (
            const std::string & file_name,
            const std::vector< double > & vec )
```

6.58.2.18 ProjectSolution() [1/2]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethod< Problem, Grid, Matrix >::ProjectSolution (
            std::vector< double > & sol,
            std::function< const double(const Mesh::Point &, const std::vector< double > &)>
GetValue,
            std::vector< double > & sol,
            const int  )
```

6.58.2.19 ProjectSolution() [2/2]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethod< Problem, Grid, Matrix >::ProjectSolution (
            std::vector< double > & sol,
            std::function< const double(const Mesh::Point &, const std::vector< double > &,
const int)> GetValue,
            std::vector< double > & sol )
```

6.58.2.20 Rediscretization() [1/2]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethod< Problem, Grid, Matrix >::Rediscretization
```

6.58.2.21 Rediscretization() [2/2]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethod< Problem, Grid, Matrix >::Rediscretization (
            const std::shared_ptr< Grid > & grid )
```

6.58.2.22 SetSolution()

```
template<class Problem , class Grid , class Matrix >
const std::vector< double > corenc::method::DGMethod< Problem, Grid, Matrix >::SetSolution (
            const int sol,
            const int liq,
            const double s,
            const double l,
            const double m )
```

6.58.2.23 SetTimeStep()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethod< Problem, Grid, Matrix >::SetTimeStep (
            const double & step ) [inline]
```

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

- CoreNCFEM/Methods/DGMethod.h

6.59 corenc::method::DGMethodZero< Problem, Grid, Matrix > Class Template Reference

```
#include <DGMethodZero.h>
```

Public Member Functions

- DGMethodZero ()
- DGMethodZero (Problem ∗p, Grid ∗g, Matrix ∗m, std::vector< double > ∗rhs)
- DGMethodZero (Problem ∗p, Grid ∗g, Matrix ∗m, Matrix ∗rm, std::vector< double > ∗rhs)
- DGMethodZero (const std::shared_ptr< Grid > &grid)
- DGMethodZero (Grid ∗grid)
- DGMethodZero (const DGMethodZero &meth)
- void Discretization ()
- const double GetValue (const Mesh::Point &) const
- const double GetValue (const Mesh::Point &, const std::vector< double > &vec) const
- const double GetValue (const Mesh::Point &, const std::vector< double > &vec, const int num) const
- const double GetEffective (const std::vector< double > &vec) const
- void ProjectSolution (std::vector< double > &, std::function< const double(const Mesh::Point &, const std::vector< double > &, const int)> GetValue, std::vector< double > &sol)
- void ProjectSolution (std::vector< double > &, std::function< const double(const Mesh::Point &, const std::vector< double > &)> GetValue, std::vector< double > &sol, const int)
- void LoadSolution (const std::vector< double > &vec)
- const std::vector< double > SetSolution (const int sol, const int liq, const double, const double, const double)
- void GetSolution (std::vector< double > &vec)
- void Rediscretization (const std::shared_ptr< Grid > &)
- void Rediscretization ()
- void SetTimeStep (const double &step)

- Matrix * GetGlobalMatrix () const
- Grid * GetMesh ()
- const std::vector< double > GetRightVector () const
- void OutDatFormat (const Mesh::Point &min, const Mesh::Point &max, const std::string &file_name, const std::vector< double > &vec) const
- void OutMeshFormat (const std::string &file_name, const std::vector< double > &vec)
- void OutMeshTimeFormat (const std::string &file_name, const std::vector< double > &vec)
- ~DGMethodZero ()

Static Public Member Functions

- static const double GetSolution (const Grid &g, const std::vector< double > &weights, const Mesh::Point &p)
- static const double GetSolution (const Grid &g, const std::vector< double > &weights, const Mesh::Point &p, const int nfem)
- static const Mesh::Point GetGradSolution (const Grid &g, const std::vector< double > &weights, const Mesh::Point &p)
- static const Mesh::Point GetGradSolution (const Grid &g, const std::vector< double > &weights, const Mesh::Point &p, const int n)

6.59.1 Constructor & Destructor Documentation

6.59.1.1 DGMethodZero() [1/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::DGMethodZero< Problem, Grid, Matrix >::DGMethodZero ( )  [inline]
```

6.59.1.2 DGMethodZero() [2/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::DGMethodZero< Problem, Grid, Matrix >::DGMethodZero (
            Problem * p,
            Grid * g,
            Matrix * m,
            std::vector< double > * rhs )  [inline]
```

6.59.1.3 DGMethodZero() [3/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::DGMethodZero< Problem, Grid, Matrix >::DGMethodZero (
            Problem * p,
            Grid * g,
            Matrix * m,
            Matrix * rm,
            std::vector< double > * rhs )  [inline]
```

6.59.1.4 DGMethodZero() [4/6]

template<class Problem , class Grid , class Matrix >
corenc::method::DGMethodZero< Problem, Grid, Matrix >::DGMethodZero (
    const std::shared_ptr< Grid > & *grid*) [inline]

6.59.1.5 DGMethodZero() [5/6]

template<class Problem , class Grid , class Matrix >
corenc::method::DGMethodZero< Problem, Grid, Matrix >::DGMethodZero (
    Grid * *grid*) [inline]

6.59.1.6 DGMethodZero() [6/6]

template<class Problem , class Grid , class Matrix >
corenc::method::DGMethodZero< Problem, Grid, Matrix >::DGMethodZero (
    const DGMethodZero< Problem, Grid, Matrix > & *meth*) [inline]

6.59.1.7 ∼DGMethodZero()

template<class Problem , class Grid , class Matrix >
corenc::method::DGMethodZero< Problem, Grid, Matrix >::∼DGMethodZero

6.59.2 Member Function Documentation

6.59.2.1 Discretization()

template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethodZero< Problem, Grid, Matrix >::Discretization

6.59.2.2 GetEffective()

template<class Problem , class Grid , class Matrix >
const double corenc::method::DGMethodZero< Problem, Grid, Matrix >::GetEffective (
    const std::vector< double > & *vec*) const

6.59.2.3 GetGlobalMatrix()

```
template<class Problem , class Grid , class Matrix >
Matrix * corenc::method::DGMethodZero< Problem, Grid, Matrix >::GetGlobalMatrix
```

6.59.2.4 GetGradSolution() [1/2]

```
template<class Problem , class Grid , class Matrix >
const Mesh::Point corenc::method::DGMethodZero< Problem, Grid, Matrix >::GetGradSolution (
            const Grid & g,
            const std::vector< double > & weights,
            const Mesh::Point & p ) [static]
```

6.59.2.5 GetGradSolution() [2/2]

```
template<class Problem , class Grid , class Matrix >
const Mesh::Point corenc::method::DGMethodZero< Problem, Grid, Matrix >::GetGradSolution (
            const Grid & g,
            const std::vector< double > & weights,
            const Mesh::Point & p,
            const int n ) [static]
```

6.59.2.6 GetMesh()

```
template<class Problem , class Grid , class Matrix >
Grid * corenc::method::DGMethodZero< Problem, Grid, Matrix >::GetMesh ( ) [inline]
```

6.59.2.7 GetRightVector()

```
template<class Problem , class Grid , class Matrix >
const std::vector< double > corenc::method::DGMethodZero< Problem, Grid, Matrix >::GetRight←
Vector
```

6.59.2.8 GetSolution() [1/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::DGMethodZero< Problem, Grid, Matrix >::GetSolution (
            const Grid & g,
            const std::vector< double > & weights,
            const Mesh::Point & p ) [static]
```

6.59.2.9 GetSolution() [2/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::DGMethodZero< Problem, Grid, Matrix >::GetSolution (
            const Grid & g,
            const std::vector< double > & weights,
            const Mesh::Point & p,
            const int nfem ) [static]
```

6.59.2.10 GetSolution() [3/3]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethodZero< Problem, Grid, Matrix >::GetSolution (
            std::vector< double > & vec )
```

6.59.2.11 GetValue() [1/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::DGMethodZero< Problem, Grid, Matrix >::GetValue (
            const Mesh::Point & p ) const
```

6.59.2.12 GetValue() [2/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::DGMethodZero< Problem, Grid, Matrix >::GetValue (
            const Mesh::Point & p,
            const std::vector< double > & vec ) const
```

6.59.2.13 GetValue() [3/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::DGMethodZero< Problem, Grid, Matrix >::GetValue (
            const Mesh::Point & p,
            const std::vector< double > & vec,
            const int num ) const
```

6.59.2.14 LoadSolution()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethodZero< Problem, Grid, Matrix >::LoadSolution (
            const std::vector< double > & vec )
```

6.59.2.15 OutDatFormat()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethodZero< Problem, Grid, Matrix >::OutDatFormat (
            const Mesh::Point & min,
            const Mesh::Point & max,
            const std::string & file_name,
            const std::vector< double > & vec ) const
```

6.59.2.16 OutMeshFormat()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethodZero< Problem, Grid, Matrix >::OutMeshFormat (
            const std::string & file_name,
            const std::vector< double > & vec )
```

6.59.2.17 OutMeshTimeFormat()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethodZero< Problem, Grid, Matrix >::OutMeshTimeFormat (
            const std::string & file_name,
            const std::vector< double > & vec )
```

6.59.2.18 ProjectSolution() [1/2]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethodZero< Problem, Grid, Matrix >::ProjectSolution (
            std::vector< double > & sol,
            std::function< const double(const Mesh::Point &, const std::vector< double > &)>
GetValue,
            std::vector< double > & sol,
            const int )
```

6.59.2.19 ProjectSolution() [2/2]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethodZero< Problem, Grid, Matrix >::ProjectSolution (
            std::vector< double > & sol,
            std::function< const double(const Mesh::Point &, const std::vector< double > &,
const int)> GetValue,
            std::vector< double > & sol )
```

6.59.2.20 Rediscretization() [1/2]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethodZero< Problem, Grid, Matrix >::Rediscretization
```

6.59.2.21 Rediscretization() [2/2]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethodZero< Problem, Grid, Matrix >::Rediscretization (
            const std::shared_ptr< Grid > & grid )
```

6.59.2.22 SetSolution()

```
template<class Problem , class Grid , class Matrix >
const std::vector< double > corenc::method::DGMethodZero< Problem, Grid, Matrix >::Set↩
Solution (
            const int sol,
            const int liq,
            const double s,
            const double l,
            const double m )
```

6.59.2.23 SetTimeStep()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::DGMethodZero< Problem, Grid, Matrix >::SetTimeStep (
            const double & step )  [inline]
```

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

- CoreNCFEM/Methods/DGMethodZero.h

6.60 corenc::method::DGSolution< Grid > Class Template Reference

```
#include <DGSolution.h>
```

Public Member Functions

- DGSolution ()
- DGSolution (const std::vector< double > &w)
- DGSolution (const DGSolution< Grid > &dg)
- DGSolution< Grid > & operator= (const DGSolution< Grid > &dg)
- ∼DGSolution ()
- const double getWeight (const Grid &g, const Mesh::Point &p) const
- const std::vector< double > getWeights () const
- const int updateWeight (const unsigned int i, const double val)

6.60.1 Constructor & Destructor Documentation

6.60.1.1 DGSolution() [1/3]

```
template<class Grid >
corenc::method::DGSolution< Grid >::DGSolution ( )  [inline]
```

6.60.1.2 DGSolution() [2/3]

```
template<class Grid >
corenc::method::DGSolution< Grid >::DGSolution (
            const std::vector< double > & w )  [inline]
```

6.60.1.3 DGSolution() [3/3]

```
template<class Grid >
corenc::method::DGSolution< Grid >::DGSolution (
            const DGSolution< Grid > & dg )  [inline]
```

6.60.1.4 ∼DGSolution()

```
template<class Grid >
corenc::method::DGSolution< Grid >::∼DGSolution ( )  [inline]
```

6.60.2 Member Function Documentation

6.60.2.1 getWeight()

```
template<class Grid >
const double corenc::method::DGSolution< Grid >::getWeight (
            const Grid & g,
            const Mesh::Point & p ) const [inline]
```

6.60.2.2 getWeights()

```
template<class Grid >
const std::vector< double > corenc::method::DGSolution< Grid >::getWeights ( ) const [inline]
```

6.60.2.3 operator=()

```
template<class Grid >
DGSolution< Grid > & corenc::method::DGSolution< Grid >::operator= (
            const DGSolution< Grid > & dg ) [inline]
```

6.60.2.4 updateWeight()

```
template<class Grid >
const int corenc::method::DGSolution< Grid >::updateWeight (
            const unsigned int i,
            const double val ) [inline]
```

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

- CoreNCFEM/Methods/DGSolution.h

6.61 corenc::solvers::eigen_solver< Matrix, Solver > Class Template Reference

```
#include <eigen_solver.h>
```

Public Member Functions

- eigen_solver ()
- ~eigen_solver ()
- void rayleigh (Matrix *A, Matrix *B, Solver *esl, std::complex< double > *mu0, double *x0, const int n) const

6.61.1 Constructor & Destructor Documentation

6.61.1.1 eigen_solver()

```
template<class Matrix , class Solver >
corenc::solvers::eigen_solver< Matrix, Solver >::eigen_solver ( ) [inline]
```

6.61.1.2 ~eigen_solver()

```
template<class Matrix , class Solver >
corenc::solvers::eigen_solver< Matrix, Solver >::~eigen_solver ( ) [inline]
```

6.61.2 Member Function Documentation

6.61.2.1 rayleigh()

```
template<class Matrix , class Solver >
void corenc::solvers::eigen_solver< Matrix, Solver >::rayleigh (
            Matrix * A,
            Matrix * B,
            Solver * esl,
            std::complex< double > * mu0,
            double * x0,
            const int n ) const [inline]
```

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

- Solvers/eigen_solver.h

6.62 Algebra::ESolver Class Reference

```
#include <MatrixSkyline.h>
```

Public Member Functions

- ESolver (const MatrixSkyline &matrix, const std::vector< double > &rightvector)
- ESolver ()
- ESolver (Solvers kek)
- void Reload (const MatrixSkyline &matrix, const std::vector< double > &right)
- void Solve (Solvers)
- const std::vector< double > Solve (MatrixSkyline &, const std::vector< double > &rhs, std::vector< double > &sol, std::vector< double > &residual, const int iter, const double eps)
- const std::vector< double > Solve (MatrixDiag &, const std::vector< double > &rhs, std::vector< double > &sol, std::vector< double > &residual, const int iter, const double eps)
- double BiCGStab (const int _maxiter)
- double GMRES (const int _maxiter)
- void GMRES (MatrixSkyline &, const std::vector< double > &rhs, std::vector< double > &sol, std::vector< double > &residual, const int iter, const double eps)
- void BiCGStab (MatrixSkyline &, const std::vector< double > &rhs, std::vector< double > &sol, std::vector< double > &residual, const int iter, const double eps)
- void Gauss (MatrixSkyline &, const std::vector< double > &rhs, std::vector< double > &sol, std::vector< double > &residual, const int iter, const double eps)
- void Gauss (Matrix &, const std::vector< double > &rhs, std::vector< double > &sol)
- void Gauss (const Matrix &, double ∗in_out)
- void Gauss (const Matrix &, double ∗in, double ∗out)
- void Gauss (const Matrix &, const double ∗in, double ∗out)
- void Pardiso (MatrixSkyline &, const std::vector< double > &rhs, std::vector< double > &sol)
- void BiCGStabPrecond ()
- const std::vector< double > GetSolution () const
- void GetSolution (std::vector< double > &sol) const
- void MatrixprodVector (double ∗res, std::vector< double > &x, MatrixSkyline &m)
- void MatrixprodVector (double ∗res, double ∗x, MatrixSkyline &m)
- void MatrixprodVector (double ∗res, double ∗x, const Matrix &m)
- void MatrixprodVector (double ∗res, const double ∗x, const Matrix &m)
- ∼ESolver ()
- auto GetSolution () -> decltype(m_solution)

6.62.1 Constructor & Destructor Documentation

6.62.1.1 ESolver() [1/3]

```
Algebra::ESolver::ESolver (
            const MatrixSkyline & matrix,
            const std::vector< double > & rightvector )   [inline]
```

6.62.1.2 ESolver() [2/3]

```
Algebra::ESolver::ESolver ( )   [inline]
```

6.62.1.3 ESolver() [3/3]

```
Algebra::ESolver::ESolver (
            Solvers kek )  [inline]
```

6.62.1.4 ∼ESolver()

```
ESolver::∼ESolver ( )
```

6.62.2 Member Function Documentation

6.62.2.1 BiCGStab() [1/2]

```
double ESolver::BiCGStab (
            const int _maxiter )
```

6.62.2.2 BiCGStab() [2/2]

```
void ESolver::BiCGStab (
            MatrixSkyline & matrix,
            const std::vector< double > & rhs,
            std::vector< double > & sol,
            std::vector< double > & residual,
            const int iter,
            const double eps )
```

6.62.2.3 BiCGStabPrecond()

```
void ESolver::BiCGStabPrecond ( )
```

6.62.2.4 Gauss() [1/5]

```
void ESolver::Gauss (
            const Matrix & matrix,
            const double * in,
            double * out )
```

6.62.2.5 Gauss() [2/5]

```
void ESolver::Gauss (
            const Matrix & matrix,
            double * in,
            double * out )
```

6.62.2.6 Gauss() [3/5]

```
void ESolver::Gauss (
            const Matrix & matrix,
            double * in_out )
```

6.62.2.7 Gauss() [4/5]

```
void ESolver::Gauss (
            Matrix & matrix,
            const std::vector< double > & rhs,
            std::vector< double > & sol )
```

6.62.2.8 Gauss() [5/5]

```
void ESolver::Gauss (
            MatrixSkyline & matrix,
            const std::vector< double > & rhs,
            std::vector< double > & sol,
            std::vector< double > & residual,
            const int iter,
            const double eps )
```

6.62.2.9 GetSolution() [1/3]

```
auto Algebra::ESolver::GetSolution ( ) -> decltype(m_solution)   [inline]
```

6.62.2.10 GetSolution() [2/3]

```
const std::vector< double > Algebra::ESolver::GetSolution ( ) const   [inline]
```

6.62.2.11 GetSolution() [3/3]

```
void Algebra::ESolver::GetSolution (
             std::vector< double > & sol ) const
```

6.62.2.12 GMRES() [1/2]

```
double ESolver::GMRES (
             const int _maxiter )
```

6.62.2.13 GMRES() [2/2]

```
void ESolver::GMRES (
             MatrixSkyline & matrix,
             const std::vector< double > & rhs,
             std::vector< double > & sol,
             std::vector< double > & residual,
             const int iter,
             const double eps )
```

6.62.2.14 MatrixprodVector() [1/4]

```
void ESolver::MatrixprodVector (
             double * res,
             const double * x,
             const Matrix & m )
```

6.62.2.15 MatrixprodVector() [2/4]

```
void ESolver::MatrixprodVector (
             double * res,
             double * x,
             const Matrix & m )
```

6.62.2.16 MatrixprodVector() [3/4]

```
void ESolver::MatrixprodVector (
             double * res,
             double * x,
             MatrixSkyline & m )
```

6.62.2.17 MatrixprodVector() [4/4]

```
void Algebra::ESolver::MatrixprodVector (
            double * res,
            std::vector< double > & x,
            MatrixSkyline & m )
```

6.62.2.18 Pardiso()

```
void ESolver::Pardiso (
            MatrixSkyline & matrix,
            const std::vector< double > & rhs,
            std::vector< double > & sol )
```

6.62.2.19 Reload()

```
void ESolver::Reload (
            const MatrixSkyline & matrix,
            const std::vector< double > & right )
```

6.62.2.20 Solve() [1/3]

```
const std::vector< double > ESolver::Solve (
            MatrixDiag & matrix,
            const std::vector< double > & rhs,
            std::vector< double > & sol,
            std::vector< double > & residual,
            const int iter,
            const double eps )
```

6.62.2.21 Solve() [2/3]

```
const std::vector< double > ESolver::Solve (
            MatrixSkyline & matrix,
            const std::vector< double > & rhs,
            std::vector< double > & sol,
            std::vector< double > & residual,
            const int iter,
            const double eps )
```

6.62.2.22 Solve() [3/3]

```
void ESolver::Solve (
            Solvers solver )
```

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

- CoreNCA/MatrixSkyline.h
- CoreNCA/MatrixSkyline.cpp

6.63 corenc::method::FEAnalysis< Method1, Method2, Mesh1, Mesh2 > Class Template Reference

```
#include <FEAnalysis.h>
```

Public Member Functions

- FEAnalysis ()
- ~FEAnalysis ()
- const double L2Norm (const Method1 &method1, const Method2 &method2, const Mesh1 &mesh1, const Mesh2 &mesh2, const std::vector< double > &w1, const std::vector< double > &w2) const

6.63.1 Constructor & Destructor Documentation

6.63.1.1 FEAnalysis()

```
template<class Method1 , class Method2 , class Mesh1 , class Mesh2 >
corenc::method::FEAnalysis< Method1, Method2, Mesh1, Mesh2 >::FEAnalysis ( )  [inline]
```

6.63.1.2 ~FEAnalysis()

```
template<class Method1 , class Method2 , class Mesh1 , class Mesh2 >
corenc::method::FEAnalysis< Method1, Method2, Mesh1, Mesh2 >::~FEAnalysis ( )  [inline]
```

6.63.2 Member Function Documentation

6.63.2.1 L2Norm()

```
template<class Method1 , class Method2 , class Mesh1 , class Mesh2 >
const double corenc::method::FEAnalysis< Method1, Method2, Mesh1, Mesh2 >::L2Norm (
            const Method1 & method1,
            const Method2 & method2,
            const Mesh1 & mesh1,
            const Mesh2 & mesh2,
            const std::vector< double > & w1,
            const std::vector< double > & w2 ) const
```

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

- CoreNCFEM/Methods/FEAnalysis.h

6.64 corenc::solvers::fem_solver< _Problem, _Mesh, _Result > Class Template Reference

`#include <fem_solver.h>`

Public Member Functions

- fem_solver ()
- ~fem_solver ()
- const int elliptic_solver (_Problem *, _Mesh *, _Result *)
- const int elliptic_solver_gauss (_Problem *, _Mesh *, _Result *)
- const double get_value (const _Mesh &, const _Result &, const Mesh::Point &p) const
- const double get_value (const _Method2 *, const _Mesh &, const _Result &, const Mesh::Point &p) const
- const double get_value (const _Method *, const _Mesh &, const _Result &, const Mesh::Point &p) const
- const double get_value (const _Mesh &, const _Result &, const Mesh::Point &p, const int i) const
- const Mesh::Point get_gradvalue (const _Mesh &, const _Result &, const Mesh::Point &p) const
- const Mesh::Point get_gradvalue (const _Mesh &, const _Result &, const Mesh::Point &p, const int i) const

6.64.1 Constructor & Destructor Documentation

6.64.1.1 fem_solver()

```
template<class _Problem , class _Mesh , class _Result >
corenc::solvers::fem_solver< _Problem, _Mesh, _Result >::fem_solver ( )  [inline]
```

6.64.1.2 ~fem_solver()

```
template<class _Problem , class _Mesh , class _Result >
corenc::solvers::fem_solver< _Problem, _Mesh, _Result >::~fem_solver ( )  [inline]
```

6.64.2 Member Function Documentation

6.64.2.1 elliptic_solver()

```
template<class _Problem , class _Mesh , class _Result >
const int corenc::solvers::fem_solver< _Problem, _Mesh, _Result >::elliptic_solver (
            _Problem * problem,
            _Mesh * mesh,
            _Result * result )
```

6.64.2.2 elliptic_solver_gauss()

```
template<class _Problem , class _Mesh , class _Result >
const int corenc::solvers::fem_solver< _Problem, _Mesh, _Result >::elliptic_solver_gauss (
            _Problem * problem,
            _Mesh * mesh,
            _Result * result )
```

6.64.2.3 get_gradvalue() [1/2]

```
template<class _Problem , class _Mesh , class _Result >
const Mesh::Point corenc::solvers::fem_solver< _Problem, _Mesh, _Result >::get_gradvalue (
            const _Mesh & mesh,
            const _Result & res,
            const Mesh::Point & p ) const
```

6.64.2.4 get_gradvalue() [2/2]

```
template<class _Problem , class _Mesh , class _Result >
const Mesh::Point corenc::solvers::fem_solver< _Problem, _Mesh, _Result >::get_gradvalue (
            const _Mesh & mesh,
            const _Result & res,
            const Mesh::Point & p,
            const int i ) const
```

6.64.2.5 get_value() [1/4]

```
template<class _Problem , class _Mesh , class _Result >
const double corenc::solvers::fem_solver< _Problem, _Mesh, _Result >::get_value (
            const _Mesh & mesh,
            const _Result & res,
            const Mesh::Point & p ) const
```

6.64.2.6 get_value() [2/4]

```
template<class _Problem , class _Mesh , class _Result >
const double corenc::solvers::fem_solver< _Problem, _Mesh, _Result >::get_value (
            const _Mesh & mesh,
            const _Result & res,
            const Mesh::Point & p,
            const int i ) const
```

6.64.2.7 get_value() [3/4]

```
template<class _Problem , class _Mesh , class _Result >
const double corenc::solvers::fem_solver< _Problem, _Mesh, _Result >::get_value (
            const _Method * ,
            const _Mesh & ,
            const _Result & ,
            const Mesh::Point & p ) const
```

6.64.2.8 get_value() [4/4]

```
template<class _Problem , class _Mesh , class _Result >
const double corenc::solvers::fem_solver< _Problem, _Mesh, _Result >::get_value (
            const _Method2 * method2,
            const _Mesh & mesh,
            const _Result & res,
            const Mesh::Point & p ) const
```

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

- Solvers/fem_solver.h

6.65 corenc::solvers::fem_solver_lib< _Problem, _Mesh, _Result > Class Template Reference

```
#include <fem_solver_lib.h>
```

Public Member Functions

- fem_solver_lib ()
- ∼fem_solver_lib ()
- const int elliptic_solver (_Problem ∗, _Mesh ∗, _Result ∗)
- const int elliptic_solver_gauss (_Problem ∗, _Mesh ∗, _Result ∗)
- const double get_value (const _Mesh &, const _Result &, const Mesh::Point &p) const
- const double get_value (const _Method2 ∗, const _Mesh &, const _Result &, const Mesh::Point &p) const
- const double get_value (const _Method ∗, const _Mesh &, const _Result &, const Mesh::Point &p) const
- const double get_value (const _Mesh &, const _Result &, const Mesh::Point &p, const int i) const
- const Mesh::Point get_gradvalue (const _Mesh &, const _Result &, const Mesh::Point &p) const
- const Mesh::Point get_gradvalue (const _Mesh &, const _Result &, const Mesh::Point &p, const int i) const

6.65.1 Constructor & Destructor Documentation

6.65.1.1 fem_solver_lib()

```
template<class _Problem , class _Mesh , class _Result >
corenc::solvers::fem_solver_lib< _Problem, _Mesh, _Result >::fem_solver_lib ( )  [inline]
```

6.65.1.2 ∼fem_solver_lib()

```
template<class _Problem , class _Mesh , class _Result >
corenc::solvers::fem_solver_lib< _Problem, _Mesh, _Result >::∼fem_solver_lib ( )  [inline]
```

6.65.2 Member Function Documentation

6.65.2.1 elliptic_solver()

```
template<class _Problem , class _Mesh , class _Result >
const int corenc::solvers::fem_solver_lib< _Problem, _Mesh, _Result >::elliptic_solver (
            _Problem ∗ problem,
            _Mesh ∗ mesh,
            _Result ∗ result )
```

6.65.2.2 elliptic_solver_gauss()

```
template<class _Problem , class _Mesh , class _Result >
const int corenc::solvers::fem_solver_lib< _Problem, _Mesh, _Result >::elliptic_solver_gauss (
            _Problem * problem,
            _Mesh * mesh,
            _Result * result )
```

6.65.2.3 get_gradvalue() [1/2]

```
template<class _Problem , class _Mesh , class _Result >
const Mesh::Point corenc::solvers::fem_solver_lib< _Problem, _Mesh, _Result >::get_gradvalue (
            const _Mesh & mesh,
            const _Result & res,
            const Mesh::Point & p ) const
```

6.65.2.4 get_gradvalue() [2/2]

```
template<class _Problem , class _Mesh , class _Result >
const Mesh::Point corenc::solvers::fem_solver_lib< _Problem, _Mesh, _Result >::get_gradvalue (
            const _Mesh & mesh,
            const _Result & res,
            const Mesh::Point & p,
            const int i ) const
```

6.65.2.5 get_value() [1/4]

```
template<class _Problem , class _Mesh , class _Result >
const double corenc::solvers::fem_solver_lib< _Problem, _Mesh, _Result >::get_value (
            const _Mesh & mesh,
            const _Result & res,
            const Mesh::Point & p ) const
```

6.65.2.6 get_value() [2/4]

```
template<class _Problem , class _Mesh , class _Result >
const double corenc::solvers::fem_solver_lib< _Problem, _Mesh, _Result >::get_value (
            const _Mesh & mesh,
            const _Result & res,
            const Mesh::Point & p,
            const int i ) const
```

6.65.2.7 get_value() [3/4]

```
template<class _Problem , class _Mesh , class _Result >
const double corenc::solvers::fem_solver_lib< _Problem, _Mesh, _Result >::get_value (
            const _Method * ,
            const _Mesh & ,
            const _Result & ,
            const Mesh::Point & p ) const
```

6.65.2.8 get_value() [4/4]

```
template<class _Problem , class _Mesh , class _Result >
const double corenc::solvers::fem_solver_lib< _Problem, _Mesh, _Result >::get_value (
            const _Method2 * method2,
            const _Mesh & mesh,
            const _Result & res,
            const Mesh::Point & p ) const
```

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

- Solvers/fem_solver_lib.h

6.66 corenc::method::FEMethod< Problem, Grid, Matrix > Class Template Reference

```
#include <FEMethod.h>
```

Public Member Functions

- FEMethod ()
- FEMethod (Problem *p, Grid *g, Matrix *m, std::vector< double > *rhs)
- FEMethod (Problem *p, Grid *g, Matrix *m, Matrix *rm, std::vector< double > *rhs)
- FEMethod (const std::shared_ptr< Grid > &grid)
- FEMethod (Grid *grid)
- FEMethod (const FEMethod &meth)
- FEMethod & operator= (const FEMethod &fem)
- void Discretization ()
- const double GetValue (const Mesh::Point &) const
- const double GetValue (const Mesh::Point &, const std::vector< double > &vec) const
- const double GetValue (const Mesh::Point &, const std::vector< double > &vec, const int num) const
- const double GetEffective (const std::vector< double > &vec) const
- void ProjectSolution (std::vector< double > &, std::function< const double(const Mesh::Point &, const std::vector< double > &, const int)> GetValue, std::vector< double > &sol)
- void ProjectSolution (std::vector< double > &, std::function< const double(const Mesh::Point &, const std::vector< double > &)> GetValue, std::vector< double > &sol, const int)
- void LoadSolution (const std::vector< double > &vec)
- const std::vector< double > SetSolution (const int sol, const int liq, const double, const double, const double)
- void GetSolution (std::vector< double > &vec)

- void Rediscretization (const std::shared_ptr< Grid > &)
- void Rediscretization ()
- void SetTimeStep (const double &step)
- Matrix * GetGlobalMatrix () const
- Grid * GetMesh ()
- const std::vector< double > GetRightVector () const
- void OutDatFormat (const Mesh::Point &min, const Mesh::Point &max, const std::string &file_name, const std::vector< double > &vec) const
- void OutMeshFormat (const std::string &file_name, const std::vector< double > &vec)
- void OutMeshTimeFormat (const std::string &file_name, const std::vector< double > &vec)
- ~FEMethod ()

Static Public Member Functions

- static const double GetSolution (const Grid &g, const std::vector< double > &weights, const Mesh::Point &p)
- static const double GetSolution (const Grid &g, const std::vector< double > &weights, const Mesh::Point &p, const int nfem)
- static const Mesh::Point GetGradSolution (const Grid &g, const std::vector< double > &weights, const Mesh::Point &p)
- static const Mesh::Point GetGradSolution (const Grid &g, const std::vector< double > &weights, const Mesh::Point &p, const int n)

6.66.1 Constructor & Destructor Documentation

6.66.1.1 FEMethod() [1/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::FEMethod< Problem, Grid, Matrix >::FEMethod ( )  [inline]
```

6.66.1.2 FEMethod() [2/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::FEMethod< Problem, Grid, Matrix >::FEMethod (
            Problem * p,
            Grid * g,
            Matrix * m,
            std::vector< double > * rhs )  [inline]
```

6.66.1.3 FEMethod() [3/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::FEMethod< Problem, Grid, Matrix >::FEMethod (
            Problem * p,
            Grid * g,
            Matrix * m,
            Matrix * rm,
            std::vector< double > * rhs )  [inline]
```

6.66.1.4 FEMethod() [4/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::FEMethod< Problem, Grid, Matrix >::FEMethod (
            const std::shared_ptr< Grid > & grid )  [inline]
```

6.66.1.5 FEMethod() [5/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::FEMethod< Problem, Grid, Matrix >::FEMethod (
            Grid * grid )  [inline]
```

6.66.1.6 FEMethod() [6/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::FEMethod< Problem, Grid, Matrix >::FEMethod (
            const FEMethod< Problem, Grid, Matrix > & meth )  [inline]
```

6.66.1.7 ∼FEMethod()

```
template<class Problem , class Grid , class Matrix >
corenc::method::FEMethod< Problem, Grid, Matrix >::∼FEMethod
```

6.66.2 Member Function Documentation

6.66.2.1 Discretization()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethod< Problem, Grid, Matrix >::Discretization
```

6.66.2.2 GetEffective()

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::FEMethod< Problem, Grid, Matrix >::GetEffective (
             const std::vector< double > & vec ) const
```

6.66.2.3 GetGlobalMatrix()

```
template<class Problem , class Grid , class Matrix >
Matrix * corenc::method::FEMethod< Problem, Grid, Matrix >::GetGlobalMatrix
```

6.66.2.4 GetGradSolution() [1/2]

```
template<class Problem , class Grid , class Matrix >
const Mesh::Point corenc::method::FEMethod< Problem, Grid, Matrix >::GetGradSolution (
             const Grid & g,
             const std::vector< double > & weights,
             const Mesh::Point & p )  [static]
```

6.66.2.5 GetGradSolution() [2/2]

```
template<class Problem , class Grid , class Matrix >
const Mesh::Point corenc::method::FEMethod< Problem, Grid, Matrix >::GetGradSolution (
             const Grid & g,
             const std::vector< double > & weights,
             const Mesh::Point & p,
             const int n )  [static]
```

6.66.2.6 GetMesh()

```
template<class Problem , class Grid , class Matrix >
Grid * corenc::method::FEMethod< Problem, Grid, Matrix >::GetMesh ( )  [inline]
```

6.66.2.7 GetRightVector()

```
template<class Problem , class Grid , class Matrix >
const std::vector< double > corenc::method::FEMethod< Problem, Grid, Matrix >::GetRightVector
```

6.66.2.8 GetSolution() [1/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::FEMethod< Problem, Grid, Matrix >::GetSolution (
            const Grid & g,
            const std::vector< double > & weights,
            const Mesh::Point & p )  [static]
```

6.66.2.9 GetSolution() [2/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::FEMethod< Problem, Grid, Matrix >::GetSolution (
            const Grid & g,
            const std::vector< double > & weights,
            const Mesh::Point & p,
            const int nfem )  [static]
```

6.66.2.10 GetSolution() [3/3]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethod< Problem, Grid, Matrix >::GetSolution (
            std::vector< double > & vec )
```

6.66.2.11 GetValue() [1/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::FEMethod< Problem, Grid, Matrix >::GetValue (
            const Mesh::Point & p ) const
```

6.66.2.12 GetValue() [2/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::FEMethod< Problem, Grid, Matrix >::GetValue (
            const Mesh::Point & p,
            const std::vector< double > & vec ) const
```

6.66.2.13 GetValue() [3/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::FEMethod< Problem, Grid, Matrix >::GetValue (
            const Mesh::Point & p,
            const std::vector< double > & vec,
            const int num ) const
```

6.66.2.14 LoadSolution()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethod< Problem, Grid, Matrix >::LoadSolution (
            const std::vector< double > & vec )
```

6.66.2.15 operator=()

```
template<class Problem , class Grid , class Matrix >
FEMethod & corenc::method::FEMethod< Problem, Grid, Matrix >::operator= (
            const FEMethod< Problem, Grid, Matrix > & fem )  [inline]
```

6.66.2.16 OutDatFormat()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethod< Problem, Grid, Matrix >::OutDatFormat (
            const Mesh::Point & min,
            const Mesh::Point & max,
            const std::string & file_name,
            const std::vector< double > & vec ) const
```

6.66.2.17 OutMeshFormat()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethod< Problem, Grid, Matrix >::OutMeshFormat (
            const std::string & file_name,
            const std::vector< double > & vec )
```

6.66.2.18 OutMeshTimeFormat()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethod< Problem, Grid, Matrix >::OutMeshTimeFormat (
            const std::string & file_name,
            const std::vector< double > & vec )
```

6.66.2.19 ProjectSolution() [1/2]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethod< Problem, Grid, Matrix >::ProjectSolution (
            std::vector< double > & sol,
            std::function< const double(const Mesh::Point &, const std::vector< double > &)>
GetValue,
            std::vector< double > & sol,
            const int )
```

6.66.2.20 ProjectSolution() [2/2]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethod< Problem, Grid, Matrix >::ProjectSolution (
            std::vector< double > & sol,
            std::function< const double(const Mesh::Point &, const std::vector< double > &,
const int)> GetValue,
            std::vector< double > & sol )
```

6.66.2.21 Rediscretization() [1/2]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethod< Problem, Grid, Matrix >::Rediscretization
```

6.66.2.22 Rediscretization() [2/2]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethod< Problem, Grid, Matrix >::Rediscretization (
            const std::shared_ptr< Grid > & grid )
```

6.66.2.23 SetSolution()

```
template<class Problem , class Grid , class Matrix >
const std::vector< double > corenc::method::FEMethod< Problem, Grid, Matrix >::SetSolution (
            const int sol,
            const int liq,
            const double s,
            const double l,
            const double m )
```

6.66.2.24 SetTimeStep()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethod< Problem, Grid, Matrix >::SetTimeStep (
            const double & step ) [inline]
```

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

- CoreNCFEM/Methods/FEMethod.h

6.67 corenc::method::FEMethodZero< Problem, Grid, Matrix > Class Template Reference

```
#include <FEMethodZero.h>
```

Public Member Functions

- FEMethodZero ()
- FEMethodZero (Problem *p, Grid *g, Matrix *m, std::vector< double > *rhs)
- FEMethodZero (Problem *p, Grid *g, Matrix *m, Matrix *rm, std::vector< double > *rhs)
- FEMethodZero (const std::shared_ptr< Grid > &grid)
- FEMethodZero (Grid *grid)
- FEMethodZero (const FEMethodZero &meth)
- void Discretization ()
- const double GetValue (const Mesh::Point &) const
- const double GetValue (const Mesh::Point &, const std::vector< double > &vec) const
- const double GetValue (const Mesh::Point &, const std::vector< double > &vec, const int num) const
- const double GetEffective (const std::vector< double > &vec) const
- void ProjectSolution (std::vector< double > &, std::function< const double(const Mesh::Point &, const std::vector< double > &, const int)> GetValue, std::vector< double > &sol)
- void ProjectSolution (std::vector< double > &, std::function< const double(const Mesh::Point &, const std::vector< double > &)> GetValue, std::vector< double > &sol, const int)
- void LoadSolution (const std::vector< double > &vec)
- const std::vector< double > SetSolution (const int sol, const int liq, const double, const double, const double)
- void GetSolution (std::vector< double > &vec)
- void Rediscretization (const std::shared_ptr< Grid > &)
- void Rediscretization ()
- void SetTimeStep (const double &step)

- Matrix * GetGlobalMatrix () const
- Grid * GetMesh ()
- const std::vector< double > GetRightVector () const
- void OutDatFormat (const Mesh::Point &min, const Mesh::Point &max, const std::string &file_name, const std::vector< double > &vec) const
- void OutMeshFormat (const std::string &file_name, const std::vector< double > &vec)
- void OutMeshTimeFormat (const std::string &file_name, const std::vector< double > &vec)
- ~FEMethodZero ()

Static Public Member Functions

- static const double GetSolution (const Grid &g, const std::vector< double > &weights, const Mesh::Point &p)
- static const double GetSolution (const Grid &g, const std::vector< double > &weights, const Mesh::Point &p, const int nfem)
- static const Mesh::Point GetGradSolution (const Grid &g, const std::vector< double > &weights, const Mesh::Point &p)
- static const Mesh::Point GetGradSolution (const Grid &g, const std::vector< double > &weights, const Mesh::Point &p, const int n)

6.67.1 Constructor & Destructor Documentation

6.67.1.1 FEMethodZero() [1/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::FEMethodZero< Problem, Grid, Matrix >::FEMethodZero ( )  [inline]
```

6.67.1.2 FEMethodZero() [2/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::FEMethodZero< Problem, Grid, Matrix >::FEMethodZero (
            Problem * p,
            Grid * g,
            Matrix * m,
            std::vector< double > * rhs )  [inline]
```

6.67.1.3 FEMethodZero() [3/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::FEMethodZero< Problem, Grid, Matrix >::FEMethodZero (
            Problem * p,
            Grid * g,
            Matrix * m,
            Matrix * rm,
            std::vector< double > * rhs )  [inline]
```

6.67.1.4 FEMethodZero() [4/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::FEMethodZero< Problem, Grid, Matrix >::FEMethodZero (
            const std::shared_ptr< Grid > & grid ) [inline]
```

6.67.1.5 FEMethodZero() [5/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::FEMethodZero< Problem, Grid, Matrix >::FEMethodZero (
            Grid * grid ) [inline]
```

6.67.1.6 FEMethodZero() [6/6]

```
template<class Problem , class Grid , class Matrix >
corenc::method::FEMethodZero< Problem, Grid, Matrix >::FEMethodZero (
            const FEMethodZero< Problem, Grid, Matrix > & meth ) [inline]
```

6.67.1.7 ∼FEMethodZero()

```
template<class Problem , class Grid , class Matrix >
corenc::method::FEMethodZero< Problem, Grid, Matrix >::~FEMethodZero
```

6.67.2 Member Function Documentation

6.67.2.1 Discretization()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethodZero< Problem, Grid, Matrix >::Discretization
```

6.67.2.2 GetEffective()

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::FEMethodZero< Problem, Grid, Matrix >::GetEffective (
            const std::vector< double > & vec ) const
```

6.67.2.3 GetGlobalMatrix()

```
template<class Problem , class Grid , class Matrix >
Matrix * corenc::method::FEMethodZero< Problem, Grid, Matrix >::GetGlobalMatrix
```

6.67.2.4 GetGradSolution() [1/2]

```
template<class Problem , class Grid , class Matrix >
const Mesh::Point corenc::method::FEMethodZero< Problem, Grid, Matrix >::GetGradSolution (
            const Grid & g,
            const std::vector< double > & weights,
            const Mesh::Point & p )  [static]
```

6.67.2.5 GetGradSolution() [2/2]

```
template<class Problem , class Grid , class Matrix >
const Mesh::Point corenc::method::FEMethodZero< Problem, Grid, Matrix >::GetGradSolution (
            const Grid & g,
            const std::vector< double > & weights,
            const Mesh::Point & p,
            const int n )  [static]
```

6.67.2.6 GetMesh()

```
template<class Problem , class Grid , class Matrix >
Grid * corenc::method::FEMethodZero< Problem, Grid, Matrix >::GetMesh ( )  [inline]
```

6.67.2.7 GetRightVector()

```
template<class Problem , class Grid , class Matrix >
const std::vector< double > corenc::method::FEMethodZero< Problem, Grid, Matrix >::GetRight←
Vector
```

6.67.2.8 GetSolution() [1/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::FEMethodZero< Problem, Grid, Matrix >::GetSolution (
            const Grid & g,
            const std::vector< double > & weights,
            const Mesh::Point & p )  [static]
```

6.67.2.9 GetSolution() [2/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::FEMethodZero< Problem, Grid, Matrix >::GetSolution (
            const Grid & g,
            const std::vector< double > & weights,
            const Mesh::Point & p,
            const int nfem ) [static]
```

6.67.2.10 GetSolution() [3/3]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethodZero< Problem, Grid, Matrix >::GetSolution (
            std::vector< double > & vec )
```

6.67.2.11 GetValue() [1/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::FEMethodZero< Problem, Grid, Matrix >::GetValue (
            const Mesh::Point & p ) const
```

6.67.2.12 GetValue() [2/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::FEMethodZero< Problem, Grid, Matrix >::GetValue (
            const Mesh::Point & p,
            const std::vector< double > & vec ) const
```

6.67.2.13 GetValue() [3/3]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::FEMethodZero< Problem, Grid, Matrix >::GetValue (
            const Mesh::Point & p,
            const std::vector< double > & vec,
            const int num ) const
```

6.67.2.14 LoadSolution()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethodZero< Problem, Grid, Matrix >::LoadSolution (
            const std::vector< double > & vec )
```

6.67.2.15 OutDatFormat()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethodZero< Problem, Grid, Matrix >::OutDatFormat (
            const Mesh::Point & min,
            const Mesh::Point & max,
            const std::string & file_name,
            const std::vector< double > & vec ) const
```

6.67.2.16 OutMeshFormat()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethodZero< Problem, Grid, Matrix >::OutMeshFormat (
            const std::string & file_name,
            const std::vector< double > & vec )
```

6.67.2.17 OutMeshTimeFormat()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethodZero< Problem, Grid, Matrix >::OutMeshTimeFormat (
            const std::string & file_name,
            const std::vector< double > & vec )
```

6.67.2.18 ProjectSolution() [1/2]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethodZero< Problem, Grid, Matrix >::ProjectSolution (
            std::vector< double > & sol,
            std::function< const double(const Mesh::Point &, const std::vector< double > &)>
GetValue,
            std::vector< double > & sol,
            const int )
```

6.67.2.19 ProjectSolution() [2/2]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethodZero< Problem, Grid, Matrix >::ProjectSolution (
            std::vector< double > & sol,
            std::function< const double(const Mesh::Point &, const std::vector< double > &,
const int)> GetValue,
            std::vector< double > & sol )
```

6.67.2.20 Rediscretization() [1/2]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethodZero< Problem, Grid, Matrix >::Rediscretization
```

6.67.2.21 Rediscretization() [2/2]

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethodZero< Problem, Grid, Matrix >::Rediscretization (
            const std::shared_ptr< Grid > & grid )
```

6.67.2.22 SetSolution()

```
template<class Problem , class Grid , class Matrix >
const std::vector< double > corenc::method::FEMethodZero< Problem, Grid, Matrix >::Set←
Solution (
            const int sol,
            const int liq,
            const double s,
            const double l,
            const double m )
```

6.67.2.23 SetTimeStep()

```
template<class Problem , class Grid , class Matrix >
void corenc::method::FEMethodZero< Problem, Grid, Matrix >::SetTimeStep (
            const double & step )  [inline]
```

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

- CoreNCFEM/Methods/FEMethodZero.h

6.68 corenc::method::FVMethod1d Class Reference

`#include <FVMethod.h>`

Public Member Functions

- FVMethod1d ()
- ∼FVMethod1d ()

Static Public Member Functions

- static const int Solve (Mesh::CMesh< CFESolution > *mesh, const std::function< const double(const double)> &flux_func, const FVFlux &flux_type, std::vector< double > &new_solution, const double time_step)
- static const double GetSolution (const Mesh::CMesh1D &g, const Mesh::Point &p)

6.68.1 Constructor & Destructor Documentation

6.68.1.1 FVMethod1d()

```
FVMethod1d::FVMethod1d ( )
```

6.68.1.2 ∼FVMethod1d()

```
FVMethod1d::~FVMethod1d ( )
```

6.68.2 Member Function Documentation

6.68.2.1 GetSolution()

```
const double FVMethod1d::GetSolution (
            const Mesh::CMesh1D & g,
            const Mesh::Point & p )  [static]
```

6.68.2.2 Solve()

```
const int FVMethod1d::Solve (
            Mesh::CMesh< CFESolution > * mesh,
            const std::function< const double(const double)> & flux_func,
            const FVFlux & flux_type,
            std::vector< double > & new_solution,
            const double time_step )  [static]
```

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

- CoreNCFEM/Methods/FVMethod.h
- CoreNCFEM/Methods/FVMethod.cpp

6.69 corenc::Mesh::Gauss1dim Struct Reference

```
#include <Point.h>
```

Static Public Attributes

- static const int m_order = 13
- static const double m_a [ ]
- static const double m_sqrt35 = sqrt(3./5.)
- static const double m_w [ ]

6.69.1 Member Data Documentation

6.69.1.1 m_a

```
const double Gauss1dim::m_a  [static]
```

Initial value:

```
=
{
    0,
    -0.2304583159551348,
    0.2304583159551348,
    -0.4484927510364469,
    0.4484927510364469,
    -0.6423493394403402,
    0.6423493394403402,
    -0.8015780907333099,
    0.8015780907333099,
    -0.9175983992229779,
    0.9175983992229779,
    -0.9841830547185881,
    0.9841830547185881
}
```

6.69.1.2 m_order

```
const int Gauss1dim::m_order = 13  [static]
```

6.69.1.3 m_sqrt35

```
const double Gauss1dim::m_sqrt35 = sqrt(3./5.)  [static]
```

6.69.1.4 m_w

```
const double Gauss1dim::m_w  [static]
```

Initial value:

```
=
{
    0.2325515532308739,
    0.2262831802628972,
    0.2262831802628972,
    0.2078160475368885,
    0.2078160475368885,
    0.1781459807619457,
    0.1781459807619457,
    0.1388735102197872,
    0.1388735102197872,
    0.0921214998377285,
    0.0921214998377285,
    0.0404840047653159,
    0.0404840047653159
}
```

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

- CoreNCFEM/Point.h
- CoreNCFEM/Point.cpp

6.70 corenc::Mesh::Gauss1dimN< N > Struct Template Reference

```
#include <Point.h>
```

Static Public Attributes

- static const int m_order
- static const double m_a []
- static const double m_w []

6.70.1 Member Data Documentation

6.70.1.1 m_a

```
template<int N>
const double corenc::Mesh::Gauss1dimN< N >::m_a[ ] [static]
```

6.70.1.2 m_order

```
template<int N>
const int corenc::Mesh::Gauss1dimN< N >::m_order [static]
```

6.70.1.3 m_w

```
template<int N>
const double corenc::Mesh::Gauss1dimN< N >::m_w[ ] [static]
```

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

- CoreNCFEM/Point.h

6.71 corenc::GaussianKernel Struct Reference

```
#include <GaussianField.h>
```

Public Member Functions

- const double gpexp (const Mesh::Point &a) const
- const double gpstep (const Mesh::Point &a) const
- GaussianKernel (const int _n, const std::vector< Mesh::Point > &centers)
- const double get_gp (const std::vector< double > &a, const Mesh::Point &p) const

Public Attributes

- int N
- std::vector< Mesh::Point > _centrs

6.71.1 Constructor & Destructor Documentation

6.71.1.1 GaussianKernel()

```
corenc::GaussianKernel::GaussianKernel (
            const int _n,
            const std::vector< Mesh::Point > & centers ) [inline]
```

6.71.2 Member Function Documentation

6.71.2.1 get_gp()

```
const double corenc::GaussianKernel::get_gp (
            const std::vector< double > & a,
            const Mesh::Point & p ) const [inline]
```

6.71.2.2 gpexp()

```
const double corenc::GaussianKernel::gpexp (
            const Mesh::Point & a ) const [inline]
```

6.71.2.3 gpstep()

```
const double corenc::GaussianKernel::gpstep (
            const Mesh::Point & a ) const [inline]
```

6.71.3 Member Data Documentation

6.71.3.1 _centrs

```
std::vector<Mesh::Point> corenc::GaussianKernel::_centrs
```

6.71.3.2 N

```
int corenc::GaussianKernel::N
```

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

- CoreNCFEM/GaussianField.h

6.72 corenc::GaussianProcess Struct Reference

```
#include <GaussianField.h>
```

Public Member Functions

- GaussianProcess (const double L, const size_t num)
- const double He (const int i, const double x) const
- const double phi (const int i, const double x) const

Public Attributes

- double sigma2
- double l
- double a
- double b
- double c
- double A
- double B
- size_t K = 1
- std::vector< double > lambda

6.72.1 Constructor & Destructor Documentation

6.72.1.1 GaussianProcess()

```
corenc::GaussianProcess::GaussianProcess (
            const double L,
            const size_t num )  [inline]
```

6.72.2 Member Function Documentation

6.72.2.1 He()

```
const double corenc::GaussianProcess::He (
            const int i,
            const double x ) const  [inline]
```

6.72.2.2 phi()

```
const double corenc::GaussianProcess::phi (
            const int i,
            const double x ) const  [inline]
```

6.72.3 Member Data Documentation

6.72.3.1 a

```
double corenc::GaussianProcess::a
```

6.72.3.2 A

```
double corenc::GaussianProcess::A
```

6.72.3.3 b

```
double corenc::GaussianProcess::b
```

6.72.3.4 B

```
double corenc::GaussianProcess::B
```

6.72.3.5 c

```
double corenc::GaussianProcess::c
```

6.72.3.6 K

```
size_t corenc::GaussianProcess::K = 1
```

6.72.3.7 l

```
double corenc::GaussianProcess::l
```

6.72.3.8 lambda

```
std::vector<double> corenc::GaussianProcess::lambda
```

6.72.3.9 sigma2

```
double corenc::GaussianProcess::sigma2
```

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

- CoreNCFEM/GaussianField.h

6.73 corenc::Mesh::GaussRectangular Struct Reference

```
#include <Point.h>
```

Static Public Attributes

- static const double m_ra [ ] = { -m_c, m_c, 0, 0, -m_a, m_a, -m_a, m_a, -m_b, m_b, -m_b, m_b }
- static const double m_rb [ ] = { 0, 0, -m_c, m_c, -m_a, -m_a, m_a, m_a, -m_b, -m_b, m_b, m_b }
- static const double m_rw [ ] = { m_wc, m_wc, m_wc, m_wc, m_wa, m_wa, m_wa, m_wa, m_wb, m_wb, m_wb, m_wb }
- static const double m_a = sqrt((114. - 3. ∗ sqrt(583.)) / 287)
- static const double m_b = sqrt((114. + 3. ∗ sqrt(583.)) / 287)
- static const double m_c = sqrt(6. / 7)
- static const double m_wa = 307. / 810 + 923. / (270.∗sqrt(583.))
- static const double m_wb = 307. / 810 - 923. / (270.∗sqrt(583.))
- static const double m_wc = 98./405

6.73.1 Member Data Documentation

6.73.1.1 m_a

```
const double GaussRectangular::m_a = sqrt((114. - 3. * sqrt(583.)) / 287) [static]
```

6.73.1.2 m_b

```
const double GaussRectangular::m_b = sqrt((114. + 3. * sqrt(583.)) / 287) [static]
```

6.73.1.3 m_c

```
const double GaussRectangular::m_c = sqrt(6. / 7) [static]
```

6.73.1.4 m_ra

```
const double GaussRectangular::m_ra = { -m_c, m_c, 0, 0, -m_a, m_a, -m_a, m_a, -m_b, m_b,
-m_b, m_b } [static]
```

6.73.1.5 m_rb

```
const double GaussRectangular::m_rb = { 0, 0, -m_c, m_c, -m_a, -m_a, m_a, m_a, -m_b, -m_b,
m_b, m_b } [static]
```

6.73.1.6 m_rw

```
const double GaussRectangular::m_rw = { m_wc, m_wc, m_wc, m_wc, m_wa, m_wa, m_wa, m_wa, m_wb,
m_wb, m_wb, m_wb } [static]
```

6.73.1.7 m_wa

```
const double GaussRectangular::m_wa = 307. / 810 + 923. / (270.*sqrt(583.)) [static]
```

6.73.1.8 m_wb

```
const double GaussRectangular::m_wb = 307. / 810 - 923. / (270.*sqrt(583.)) [static]
```

6.73.1.9 m_wc

```
const double GaussRectangular::m_wc = 98./405  [static]
```

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

- CoreNCFEM/Point.h
- CoreNCFEM/Point.cpp

6.74 corenc::Mesh::GaussRectangularCubic Struct Reference

```
#include <Point.h>
```

Static Public Attributes

- static const double m_ra [ ]
- static const double m_rb [ ]
- static const double m_rc [ ]
- static const double m_rw [ ]
- static const double m_a = sqrt(6. / 7)
- static const double m_b = sqrt((960 - 33. ∗ sqrt(238.)) / 2726)
- static const double m_c = sqrt((960 + 33. ∗ sqrt(238.)) / 2726)
- static const double m_w1 = 1078. / 3645
- static const double m_w2 = 343. / 3645
- static const double m_w3 = 43. / 135 + 829. ∗ sqrt(238.) / 136323
- static const double m_w4 = 43. / 135 - 829. ∗ sqrt(238.) / 136323
- static const int m_s { 34 }

6.74.1 Member Data Documentation

6.74.1.1 m_a

```
const double GaussRectangularCubic::m_a = sqrt(6.  / 7)  [static]
```

6.74.1.2 m_b

```
const double GaussRectangularCubic::m_b = sqrt((960 - 33.  * sqrt(238.))  / 2726)  [static]
```

6.74.1.3 m_c

```
const double GaussRectangularCubic::m_c = sqrt((960 + 33. * sqrt(238.)) / 2726) [static]
```

6.74.1.4 m_ra

```
const double GaussRectangularCubic::m_ra [static]
```

6.74.1.5 m_rb

```
const double GaussRectangularCubic::m_rb [static]
```

6.74.1.6 m_rc

```
const double GaussRectangularCubic::m_rc [static]
```

6.74.1.7 m_rw

```
const double GaussRectangularCubic::m_rw [static]
```

6.74.1.8 m_s

```
const int corenc::Mesh::GaussRectangularCubic::m_s { 34 } [static]
```

6.74.1.9 m_w1

```
const double GaussRectangularCubic::m_w1 = 1078. / 3645 [static]
```

6.74.1.10 m_w2

```
const double GaussRectangularCubic::m_w2 = 343. / 3645 [static]
```

6.74.1.11 m_w3

```
const double GaussRectangularCubic::m_w3 = 43. / 135 + 829. * sqrt(238.) / 136323 [static]
```

6.74.1.12 m_w4

```
const double GaussRectangularCubic::m_w4 = 43. / 135 - 829. * sqrt(238.) / 136323 [static]
```

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

- CoreNCFEM/Point.h
- CoreNCFEM/Point.cpp

6.75 corenc::Mesh::GaussTetrahedron Struct Reference

```
#include <Point.h>
```

Static Public Attributes

- static const double m_la [ ] = { 1. / 4, 11. / 14, 5. / 70, 5. / 70, 5. / 70, m_psq, m_msq, m_msq, m_msq, m_psq, m_psq }
- static const double m_lb [ ] = { 1. / 4, 5. / 70, 11. / 14, 5. / 70, 5. / 70, m_msq, m_psq, m_msq, m_psq, m_msq, m_psq }
- static const double m_lc [ ] = { 1. / 4, 5. / 70, 5. / 70, 11. / 14, 5. / 70, m_msq, m_msq, m_psq, m_psq, m_psq, m_msq }
- static const double m_ld [ ] = { 1. / 4, 1. / 6, 1. / 6, 1. / 6, 1. / 3 }
- static const double m_w [ ]
- static const double m_psq = (1 + sqrt(5. / 14)) / 4
- static const double m_msq = (1 - sqrt(5. / 14)) / 4

6.75.1 Member Data Documentation

6.75.1.1 m_la

```
const double GaussTetrahedron::m_la = { 1. / 4, 11. / 14, 5. / 70, 5. / 70, 5. / 70,
m_psq, m_msq, m_msq, m_msq, m_psq, m_psq } [static]
```

6.75.1.2 m_lb

```
const double GaussTetrahedron::m_lb = { 1. / 4, 5. / 70, 11. / 14, 5. / 70, 5. / 70,
m_msq, m_psq, m_msq, m_psq, m_msq, m_psq } [static]
```

6.75.1.3 m_lc

```
const double GaussTetrahedron::m_lc = { 1. / 4, 5. / 70, 5. / 70, 11. / 14, 5. / 70,
m_msq, m_msq, m_psq, m_psq, m_psq, m_msq } [static]
```

6.75.1.4 m_ld

```
const double GaussTetrahedron::m_ld = { 1. / 4, 1. / 6, 1. / 6, 1. / 6, 1. / 3 } [static]
```

6.75.1.5 m_msq

```
const double GaussTetrahedron::m_msq = (1 - sqrt(5. / 14)) / 4 [static]
```

6.75.1.6 m_psq

```
const double GaussTetrahedron::m_psq = (1 + sqrt(5. / 14)) / 4 [static]
```

6.75.1.7 m_w

```
const double GaussTetrahedron::m_w [static]
```

Initial value:

```
= { -74. / 5625, 343. / 45000, 343. / 45000, 343. / 45000, 343. / 45000,
    56. / 2250, 56. / 2250, 56. / 2250, 56. / 2250, 56. / 2250, 56. / 2250 }
```

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

- CoreNCFEM/Point.h
- CoreNCFEM/Point.cpp

6.76 corenc::Mesh::GaussTriangle Struct Reference

```
#include <Point.h>
```

Static Public Attributes

- static const double m_tra [ ]
- static const double m_trb [ ]
- static const double m_sqrt15 = sqrt(15.)
- static const double m_trw [ ]
- static const int m_order = 7

6.76.1 Member Data Documentation

6.76.1.1 m_order

```
const int GaussTriangle::m_order = 7  [static]
```

6.76.1.2 m_sqrt15

```
const double GaussTriangle::m_sqrt15 = sqrt(15.)  [static]
```

6.76.1.3 m_tra

```
const double GaussTriangle::m_tra  [static]
```

Initial value:

```
=
{
    1. / 3,
    (6 + m_sqrt15) / 21,
    (6 + m_sqrt15) / 21,
    (9 - 2 * m_sqrt15) / 21,
    (9 + 2 * m_sqrt15) / 21,
    (6 - m_sqrt15) / 21,
    (6 - m_sqrt15) / 21,
}
```

6.76.1.4 m_trb

```
const double GaussTriangle::m_trb  [static]
```

Initial value:

```
=
{
    1. / 3,
    (9. - 2.*m_sqrt15) / 21,
    (6. + m_sqrt15) / 21,
    (6. + m_sqrt15) / 21,
    (6. - m_sqrt15) / 21,
    (9. + 2. * m_sqrt15) / 21,
    (6. - m_sqrt15) / 21
}
```

6.76.1.5 m_trw

```
const double GaussTriangle::m_trw  [static]
```

Initial value:

```
=
{
    9. / 80,
    (155. + m_sqrt15) / 2400,
    (155. + m_sqrt15) / 2400,
    (155. + m_sqrt15) / 2400,
    (155. - m_sqrt15) / 2400,
    (155. - m_sqrt15) / 2400,
    (155. - m_sqrt15) / 2400
}
```

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

- CoreNCFEM/Point.h
- CoreNCFEM/Point.cpp

6.77 Algebra::Matrix Class Reference

```
#include <Matrix.h>
```

Public Member Functions

- Matrix (const unsigned int &size, const std::vector< std::set< unsigned int > > &nonzero)
- Matrix ()
- ∼Matrix ()
- void NullRow (const int row)
- double & operator() (const int i, const int j)
- const int GetSize () const
- void NullMatrix ()
- Matrix & operator= (const Matrix &)
- Matrix (const Matrix &matrix)
- void Create (const unsigned int &size, const std::vector< std::set< unsigned int > > &nonzero)
- void Create (const unsigned int &size)
- const double GetElement (const int i, const int j)
- void AddElement (const unsigned int i, const unsigned int j, const double a)

6.77.1 Detailed Description

The Dense Matrix Class

6.77.2 Constructor & Destructor Documentation

6.77.2.1 Matrix() [1/3]

```
Algebra::Matrix::Matrix (
            const unsigned int & size,
            const std::vector< std::set< unsigned int > > & nonzero )
```

6.77.2.2 Matrix() [2/3]

```
Algebra::Matrix::Matrix ( )  [inline]
```

6.77.2.3 ∼Matrix()

```
Algebra::Matrix::∼Matrix ( )
```

6.77.2.4 Matrix() [3/3]

```
Algebra::Matrix::Matrix (
            const Matrix & matrix )
```

6.77.3 Member Function Documentation

6.77.3.1 AddElement()

```
void Algebra::Matrix::AddElement (
            const unsigned int i,
            const unsigned int j,
            const double a )  [inline]
```

6.77.3.2 Create() [1/2]

```
void Algebra::Matrix::Create (
            const unsigned int & size )
```

6.77.3.3 Create() [2/2]

```
void Algebra::Matrix::Create (
            const unsigned int & size,
            const std::vector< std::set< unsigned int > > & nonzero )
```

6.77.3.4 GetElement()

```
const double Algebra::Matrix::GetElement (
            const int i,
            const int j )  [inline]
```

6.77.3.5 GetSize()

```
const int Algebra::Matrix::GetSize ( ) const  [inline]
```

6.77.3.6 NullMatrix()

```
void Algebra::Matrix::NullMatrix ( )
```

6.77.3.7 NullRow()

```
void Algebra::Matrix::NullRow (
            const int row )
```

6.77.3.8 operator()()

```
double & Algebra::Matrix::operator() (
            const int i,
            const int j )  [inline]
```

6.77.3.9 operator=()

```
Algebra::Matrix & Algebra::Matrix::operator= (
            const Matrix & matrix )
```

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

- CoreNCA/Matrix.h
- CoreNCA/Matrix.cpp

6.78 Algebra::MatrixDiag Class Reference

`#include <MatrixDiag.h>`

Public Member Functions

- MatrixDiag (const unsigned int &size, const std::vector< std::set< unsigned int > > &nonzero)
- MatrixDiag ()
- ∼MatrixDiag ()
- void NullRow (const int row)
- double & operator() (const int i, const int j)
- const int GetSize () const
- void NullMatrix ()
- MatrixDiag & operator= (const MatrixDiag &)
- MatrixDiag (const MatrixDiag &matrix)
- void Create (const unsigned int &size, const std::vector< std::set< unsigned int > > &nonzero)
- void AddElement (const unsigned int i, const unsigned int j, const double a)

6.78.1 Detailed Description

The diagonal matrix class

6.78.2 Constructor & Destructor Documentation

6.78.2.1 MatrixDiag() [1/3]

```
Algebra::MatrixDiag::MatrixDiag (
            const unsigned int & size,
            const std::vector< std::set< unsigned int > > & nonzero )
```

6.78.2.2 MatrixDiag() [2/3]

```
Algebra::MatrixDiag::MatrixDiag ( )  [inline]
```

6.78.2.3 ∼MatrixDiag()

```
Algebra::MatrixDiag::∼MatrixDiag ( )
```

6.78.2.4 MatrixDiag() [3/3]

```
Algebra::MatrixDiag::MatrixDiag (
             const MatrixDiag & matrix )
```

6.78.3 Member Function Documentation

6.78.3.1 AddElement()

```
void Algebra::MatrixDiag::AddElement (
             const unsigned int i,
             const unsigned int j,
             const double a )  [inline]
```

6.78.3.2 Create()

```
void Algebra::MatrixDiag::Create (
             const unsigned int & size,
             const std::vector< std::set< unsigned int > > & nonzero )
```

6.78.3.3 GetSize()

```
const int Algebra::MatrixDiag::GetSize ( ) const  [inline]
```

6.78.3.4 NullMatrix()

```
void Algebra::MatrixDiag::NullMatrix ( )
```

6.78.3.5 NullRow()

```
void Algebra::MatrixDiag::NullRow (
            const int row )
```

6.78.3.6 operator()()

```
double & Algebra::MatrixDiag::operator() (
            const int i,
            const int j ) [inline]
```

6.78.3.7 operator=()

```
Algebra::MatrixDiag & Algebra::MatrixDiag::operator= (
            const MatrixDiag & matrix )
```

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

- CoreNCA/MatrixDiag.h
- CoreNCA/MatrixDiag.cpp

6.79 Algebra::MatrixSkyline Class Reference

```
#include <MatrixSkyline.h>
```

Public Member Functions

- MatrixSkyline (const unsigned int &Size, const std::vector< std::set< unsigned int > > &nonzero)
- MatrixSkyline ()
- ∼MatrixSkyline ()
- void NullRow (int row)
- double & operator() (const int i, const int j)
- const double operator() (const int i, const int j) const
- const double GetElement (const int i, const int j) const
- const int GetSize () const
- void NullMatrix ()
- MatrixSkyline & operator= (const MatrixSkyline &)
- MatrixSkyline (const MatrixSkyline &matrix)
- void Create (const unsigned int &Size, const std::vector< std::set< unsigned int > > &nonzero)
- void AddElement (const unsigned int i, const unsigned int j, const double a)

Static Public Member Functions

- static const MatrixSkyline diff_skymatrix (const MatrixSkyline &matrix, const MatrixSkyline &B, const double scal)
- static const MatrixSkyline diff_skymatrix (const MatrixSkyline &matrix, const MatrixSkyline &B, const double a, const double b)
- static const MatrixSkyline transpose_sky (const MatrixSkyline &matrix)

6.79.1 Detailed Description

The sparse (skyline) matrix format

6.79.2 Constructor & Destructor Documentation

6.79.2.1 MatrixSkyline() [1/3]

```
Algebra::MatrixSkyline::MatrixSkyline (
            const unsigned int & Size,
            const std::vector< std::set< unsigned int > > & nonzero )
```

6.79.2.2 MatrixSkyline() [2/3]

```
Algebra::MatrixSkyline::MatrixSkyline ( )  [inline]
```

6.79.2.3 ∼MatrixSkyline()

```
MatrixSkyline::∼MatrixSkyline ( )
```

6.79.2.4 MatrixSkyline() [3/3]

```
MatrixSkyline::MatrixSkyline (
            const MatrixSkyline & matrix )
```

6.79.3 Member Function Documentation

6.79.3.1 AddElement()

```
void Algebra::MatrixSkyline::AddElement (
            const unsigned int i,
            const unsigned int j,
            const double a )  [inline]
```

6.79.3.2 Create()

```
void MatrixSkyline::Create (
            const unsigned int & Size,
            const std::vector< std::set< unsigned int > > & nonzero )
```

6.79.3.3 diff_skymatrix() [1/2]

```
static const MatrixSkyline Algebra::MatrixSkyline::diff_skymatrix (
            const MatrixSkyline & matrix,
            const MatrixSkyline & B,
            const double a,
            const double b )  [inline], [static]
```

6.79.3.4 diff_skymatrix() [2/2]

```
static const MatrixSkyline Algebra::MatrixSkyline::diff_skymatrix (
            const MatrixSkyline & matrix,
            const MatrixSkyline & B,
            const double scal )  [inline], [static]
```

6.79.3.5 GetElement()

```
const double Algebra::MatrixSkyline::GetElement (
            const int i,
            const int j ) const  [inline]
```

6.79.3.6 GetSize()

```
const int Algebra::MatrixSkyline::GetSize ( ) const  [inline]
```

6.79.3.7 NullMatrix()

```
void MatrixSkyline::NullMatrix ( )
```

6.79.3.8 NullRow()

```
void MatrixSkyline::NullRow (
            int row )
```

6.79.3.9 operator()() [1/2]

```
double & Algebra::MatrixSkyline::operator() (
            const int i,
            const int j )  [inline]
```

6.79.3.10 operator()() [2/2]

```
const double Algebra::MatrixSkyline::operator() (
            const int i,
            const int j ) const  [inline]
```

6.79.3.11 operator=()

```
MatrixSkyline & MatrixSkyline::operator= (
            const MatrixSkyline & matrix )
```

6.79.3.12 transpose_sky()

```
static const MatrixSkyline Algebra::MatrixSkyline::transpose_sky (
            const MatrixSkyline & matrix )  [inline], [static]
```

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

- CoreNCA/MatrixSkyline.h
- CoreNCA/MatrixSkyline.cpp

6.80 corenc::multi_vector< T > Class Template Reference

```
#include <multi_vector.h>
```

Public Member Functions

- multi_vector ()
- multi_vector (const size_t block, const size_t dim)
- multi_vector (const size_t dim)
- ~multi_vector ()
- const T get (const size_t i...) const
- const T get (const std::vector< size_t > &i) const
- const int set (const T &element, const std::vector< size_t > &index)
- const int fill_inc ()
- void resize (const size_t block)
- void resize (const size_t block, const size_t dim)
- const size_t size () const
- const size_t totalsize () const

6.80.1 Constructor & Destructor Documentation

6.80.1.1 multi_vector() [1/3]

```
template<class T >
corenc::multi_vector< T >::multi_vector
```

6.80.1.2 multi_vector() [2/3]

```
template<class T >
corenc::multi_vector< T >::multi_vector (
        const size_t block,
        const size_t dim )
```

6.80.1.3 multi_vector() [3/3]

```
template<class T >
corenc::multi_vector< T >::multi_vector (
        const size_t dim )
```

6.80.1.4 ∼multi_vector()

```
template<class T >
corenc::multi_vector< T >::∼multi_vector
```

6.80.2 Member Function Documentation

6.80.2.1 fill_inc()

```
template<class T >
const int corenc::multi_vector< T >::fill_inc
```

6.80.2.2 get() [1/2]

```
template<class T >
const T corenc::multi_vector< T >::get (
            const size_t i... ) const
```

6.80.2.3 get() [2/2]

```
template<class T >
const T corenc::multi_vector< T >::get (
            const std::vector< size_t > & i ) const
```

6.80.2.4 resize() [1/2]

```
template<class T >
void corenc::multi_vector< T >::resize (
            const size_t block )
```

6.80.2.5 resize() [2/2]

```
template<class T >
void corenc::multi_vector< T >::resize (
            const size_t block,
            const size_t dim )
```

6.80.2.6 set()

```
template<class T >
const int corenc::multi_vector< T >::set (
            const T & element,
            const std::vector< size_t > & index )
```

6.80.2.7 size()

```
template<class T >
const size_t corenc::multi_vector< T >::size
```

6.80.2.8 totalsize()

```
template<class T >
const size_t corenc::multi_vector< T >::totalsize
```

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

- CoreNCFEM/multi_vector.h

6.81 corenc::Mesh::parameter< T > Class Template Reference

```
#include <Parameter.h>
```

Public Types

- using cfunc = std::function< const T(const int, const int, const Point &)>
- using cfunc_old = std::function< const T(const int, const Point &)>

Public Member Functions

- parameter ()
- parameter (const cfunc &func)
- parameter (const cfunc_old &func)
- parameter (const double _p)
- parameter (const Mesh::Point _p)
- parameter (const parameter< T > &_p)
- ∼parameter ()
- const T get (const Point &p) const
- const T get (const int number, const Point &p) const
- const T get (const int element, const int node, const Point &p) const
- void set (const cfunc &func)

6.81.1 Member Typedef Documentation

6.81.1.1 cfunc

```
template<class T >
using corenc::Mesh::parameter< T >::cfunc = std::function<const T(const int, const int, const
Point&)>
```

6.81.1.2 cfunc_old

```
template<class T >
using corenc::Mesh::parameter< T >::cfunc_old = std::function<const T(const int, const Point&)>
```

6.81.2 Constructor & Destructor Documentation

6.81.2.1 parameter() [1/6]

```
template<class T >
corenc::Mesh::parameter< T >::parameter ( )  [inline]
```

6.81.2.2 parameter() [2/6]

```
template<class T >
corenc::Mesh::parameter< T >::parameter (
            const cfunc & func )  [inline]
```

6.81.2.3 parameter() [3/6]

```
template<class T >
corenc::Mesh::parameter< T >::parameter (
            const cfunc_old & func )  [inline]
```

6.81.2.4 parameter() [4/6]

```
template<class T >
corenc::Mesh::parameter< T >::parameter (
            const double _p )  [inline]
```

6.81.2.5 parameter() [5/6]

```
template<class T >
corenc::Mesh::parameter< T >::parameter (
            const Mesh::Point _p )  [inline]
```

6.81.2.6 parameter() [6/6]

```
template<class T >
corenc::Mesh::parameter< T >::parameter (
            const parameter< T > & _p )  [inline]
```

6.81.2.7 ~parameter()

```
template<class T >
corenc::Mesh::parameter< T >::~parameter ( )  [inline]
```

6.81.3 Member Function Documentation

6.81.3.1 get() [1/3]

```
template<class T >
const T corenc::Mesh::parameter< T >::get (
            const int element,
            const int node,
            const Point & p ) const  [inline]
```

6.81.3.2 get() [2/3]

```
template<class T >
const T corenc::Mesh::parameter< T >::get (
            const int number,
            const Point & p ) const  [inline]
```

6.81.3.3 get() [3/3]

```
template<class T >
const T corenc::Mesh::parameter< T >::get (
            const Point & p ) const  [inline]
```

6.81.3.4 set()

```
template<class T >
void corenc::Mesh::parameter< T >::set (
            const cfunc & func )  [inline]
```

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

- CoreNCFEM/Parameter.h

6.82 corenc::Mesh::Point Class Reference

```
#include <Point.h>
```

Public Member Functions

- Point ()
- Point (const double _x, const double _y)
- Point (const double _x, const double _y, const double _z)
- Point (const Point &p)
- const double Jacobian () const
- Point & operator= (const Point &p)
- const bool operator== (const Point &p)
- const bool operator< (const Point &p2)
- const Point operator* (const double rhs)
- Point & operator+= (const Point &rhs)
- Point & operator*= (const double rhs)

Public Attributes

- double x
- double y
- double z

Friends

- const bool operator!= (const Point &p1, const Point &p2)
- const double operator* (const Point &lhs, const Point &rhs)
- const Point operator* (const Point &lhs, const double rhs)
- const Point operator* (const double lhs, const Point &rhs)
- const Point operator+ (const Point &lhs, const Point &rhs)
- const Point operator- (const Point &lhs, const Point &rhs)

6.82.1 Constructor & Destructor Documentation

6.82.1.1 Point() [1/4]

```
corenc::Mesh::Point::Point ( )  [inline]
```

6.82.1.2 Point() [2/4]

```
corenc::Mesh::Point::Point (
            const double _x,
            const double _y )  [inline]
```

6.82.1.3 Point() [3/4]

```
corenc::Mesh::Point::Point (
            const double _x,
            const double _y,
            const double _z )  [inline]
```

6.82.1.4 Point() [4/4]

```
corenc::Mesh::Point::Point (
            const Point & p )  [inline]
```

6.82.2 Member Function Documentation

6.82.2.1 Jacobian()

```
const double corenc::Mesh::Point::Jacobian ( ) const  [inline]
```

6.82.2.2 operator*()

```
const Point corenc::Mesh::Point::operator* (
            const double rhs )  [inline]
```

6.82.2.3 operator*=()

```
Point & corenc::Mesh::Point::operator*= (
            const double rhs ) [inline]
```

6.82.2.4 operator+=()

```
Point & corenc::Mesh::Point::operator+= (
            const Point & rhs ) [inline]
```

6.82.2.5 operator<()

```
const bool corenc::Mesh::Point::operator< (
            const Point & p2 ) [inline]
```

6.82.2.6 operator=()

```
Point & corenc::Mesh::Point::operator= (
            const Point & p ) [inline]
```

6.82.2.7 operator==()

```
const bool corenc::Mesh::Point::operator== (
            const Point & p ) [inline]
```

6.82.3 Friends And Related Function Documentation

6.82.3.1 operator"!=

```
const bool operator!= (
            const Point & p1,
            const Point & p2 ) [friend]
```

6.82.3.2 operator* [1/3]

```
const Point operator* (
             const double lhs,
             const Point & rhs )  [friend]
```

6.82.3.3 operator* [2/3]

```
const Point operator* (
             const Point & lhs,
             const double rhs )  [friend]
```

6.82.3.4 operator* [3/3]

```
const double operator* (
             const Point & lhs,
             const Point & rhs )  [friend]
```

6.82.3.5 operator+

```
const Point operator+ (
             const Point & lhs,
             const Point & rhs )  [friend]
```

6.82.3.6 operator-

```
const Point operator- (
             const Point & lhs,
             const Point & rhs )  [friend]
```

6.82.4 Member Data Documentation

6.82.4.1 x

```
double corenc::Mesh::Point::x
```

6.82.4.2 y

```
double corenc::Mesh::Point::y
```

6.82.4.3 z

```
double corenc::Mesh::Point::z
```

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

- CoreNCFEM/Point.h

6.83 corenc::Mesh::point_source< T > Class Template Reference

```
#include <Parameter.h>
```

Public Member Functions

- point_source ()
- point_source (const Mesh::Point &p, const T &val)
- const T get_value () const
- const Mesh::Point get_point () const
- point_source< T > & operator= (const point_source< T > &ps)

6.83.1 Constructor & Destructor Documentation

6.83.1.1 point_source() [1/2]

```
template<class T >
corenc::Mesh::point_source< T >::point_source ( )  [inline]
```

6.83.1.2 point_source() [2/2]

```
template<class T >
corenc::Mesh::point_source< T >::point_source (
            const Mesh::Point & p,
            const T & val )  [inline]
```

6.83.2 Member Function Documentation

6.83.2.1 get_point()

```
template<class T >
const Mesh::Point corenc::Mesh::point_source< T >::get_point ( ) const  [inline]
```

6.83.2.2 get_value()

```
template<class T >
const T corenc::Mesh::point_source< T >::get_value ( ) const  [inline]
```

6.83.2.3 operator=()

```
template<class T >
point_source< T > & corenc::Mesh::point_source< T >::operator= (
            const point_source< T > & ps )  [inline]
```

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

- CoreNCFEM/Parameter.h

6.84 corenc::method::RungeKutta< Problem, Type > Class Template Reference

```
#include <RungeKutta.h>
```

Public Member Functions

- RungeKutta ()
- RungeKutta (const double step, const double final, Problem *problem, const Type *solution)
- const Type discretize (const Type &solution, const std::function< const Type(const double time, const double time_step, const Type &curr_sol, Type *result)> &func)
- const Type explicitEuler (const Type &solution, const std::function< const Type(const double time, const double time_step, const Type &curr_sol, Type *result)> &func)
- void updateTimestep (const double step)
- ~RungeKutta ()

6.84.1 Constructor & Destructor Documentation

6.84.1.1 RungeKutta() [1/2]

```
template<class Problem , class Type >
corenc::method::RungeKutta< Problem, Type >::RungeKutta ( )  [inline]
```

6.84.1.2 RungeKutta() [2/2]

```
template<class Problem , class Type >
corenc::method::RungeKutta< Problem, Type >::RungeKutta (
            const double step,
            const double final,
            Problem * problem,
            const Type * solution )  [inline]
```

6.84.1.3 ∼RungeKutta()

```
template<class Problem , class Type >
corenc::method::RungeKutta< Problem, Type >::∼RungeKutta ( )  [inline]
```

6.84.2 Member Function Documentation

6.84.2.1 discretize()

```
template<class Problem , class Type >
const Type corenc::method::RungeKutta< Problem, Type >::discretize (
            const Type & solution,
            const std::function< const Type(const double time, const double time_step, const
Type &curr_sol, Type *result)> & func )
```

6.84.2.2 explicitEuler()

```
template<class Problem , class Type >
const Type corenc::method::RungeKutta< Problem, Type >::explicitEuler (
            const Type & solution,
            const std::function< const Type(const double time, const double time_step, const
Type &curr_sol, Type *result)> & func )
```

6.84.2.3 updateTimestep()

```
template<class Problem , class Type >
void corenc::method::RungeKutta< Problem, Type >::updateTimestep (
            const double step )  [inline]
```

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

- CoreNCFEM/Methods/RungeKutta.h

6.85 corenc::method::STSolution< Grid > Class Template Reference

```
#include <DGSolution.h>
```

Public Member Functions

- STSolution ()
- STSolution (const Grid &g)
- STSolution (const std::vector< DGSolution< Grid > > &w, const std::vector< double > time, const Grid &g)
- STSolution (const STSolution< Grid > &st)
- STSolution< Grid > & operator= (const STSolution< Grid > &st)
- ~STSolution ()
- const double getWeight (const Mesh::Point &p, const double time) const
- const int updateWeight (const std::vector< double > time, const std::vector< DGSolution< Grid > > w)
- const int addTimeLayer (const double time, const DGSolution< Grid > w)
- const std::vector< DGSolution< Grid > > getWeights () const

6.85.1 Constructor & Destructor Documentation

6.85.1.1 STSolution() [1/4]

```
template<class Grid >
corenc::method::STSolution< Grid >::STSolution ( )  [inline]
```

6.85.1.2 STSolution() [2/4]

```
template<class Grid >
corenc::method::STSolution< Grid >::STSolution (
            const Grid & g )  [inline]
```

6.85.1.3 STSolution() [3/4]

```
template<class Grid >
corenc::method::STSolution< Grid >::STSolution (
            const std::vector< DGSolution< Grid > > & w,
            const std::vector< double > time,
            const Grid & g )  [inline]
```

6.85.1.4 STSolution() [4/4]

```
template<class Grid >
corenc::method::STSolution< Grid >::STSolution (
            const STSolution< Grid > & st )  [inline]
```

6.85.1.5 ∼STSolution()

```
template<class Grid >
corenc::method::STSolution< Grid >::∼STSolution ( )  [inline]
```

6.85.2 Member Function Documentation

6.85.2.1 addTimeLayer()

```
template<class Grid >
const int corenc::method::STSolution< Grid >::addTimeLayer (
            const double time,
            const DGSolution< Grid > w )  [inline]
```

6.85.2.2 getWeight()

```
template<class Grid >
const double corenc::method::STSolution< Grid >::getWeight (
            const Mesh::Point & p,
            const double time ) const  [inline]
```

6.85.2.3 getWeights()

```
template<class Grid >
const std::vector< DGSolution< Grid > > corenc::method::STSolution< Grid >::getWeights ( )
const  [inline]
```

6.85.2.4 operator=()

```
template<class Grid >
STSolution< Grid > & corenc::method::STSolution< Grid >::operator= (
            const STSolution< Grid > & st )  [inline]
```

6.85.2.5 updateWeight()

```
template<class Grid >
const int corenc::method::STSolution< Grid >::updateWeight (
            const std::vector< double > time,
            const std::vector< DGSolution< Grid > > w )  [inline]
```

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

- CoreNCFEM/Methods/DGSolution.h

6.86 corenc::method::system_dg_method< Problem, Grid, Matrix > Class Template Reference

`#include <system_dg_method.h>`

Public Member Functions

- system_dg_method ()
- system_dg_method (Problem ∗p, Grid ∗g, Matrix ∗m, const size_t sys_size, std::vector< double > ∗rhs)
- ∼system_dg_method ()
- const int Assemble ()
- const int changeFlux (const DGFlux flux_type)
- const Matrix ∗ GetGlobalMatrix () const
- const std::vector< double > GetSolution () const
- const double GetSolution (const std::vector< double > &point) const
- const double GetMaxSolution () const
- const double GetMinSolution () const
- const double GetSolution (const std::vector< double > &dg_sol, const Mesh::Point &p)
- const int toDGSolution (const Grid &g, std::vector< double > &dg_result) const
- const int updateWeights (const std::vector< double > &dg_result)
- const int DGtostandart (const std::vector< double > &dg_result)

Static Public Member Functions

- static const double GetSolution (const Grid &g, const std::vector< double > &dg_sol, const Mesh::Point &p)

6.86.1 Constructor & Destructor Documentation

6.86.1.1 system_dg_method() [1/2]

```
template<class Problem , class Grid , class Matrix >
corenc::method::system_dg_method< Problem, Grid, Matrix >::system_dg_method ( )  [inline]
```

6.86.1.2 system_dg_method() [2/2]

```
template<class Problem , class Grid , class Matrix >
corenc::method::system_dg_method< Problem, Grid, Matrix >::system_dg_method (
            Problem * p,
            Grid * g,
            Matrix * m,
            const size_t sys_size,
            std::vector< double > * rhs )  [inline]
```

6.86.1.3 ~system_dg_method()

```
template<class Problem , class Grid , class Matrix >
corenc::method::system_dg_method< Problem, Grid, Matrix >::~system_dg_method ( )  [inline]
```

6.86.2 Member Function Documentation

6.86.2.1 Assemble()

```
template<class Problem , class Grid , class Matrix >
const int corenc::method::system_dg_method< Problem, Grid, Matrix >::Assemble
```

6.86.2.2 changeFlux()

```
template<class Problem , class Grid , class Matrix >
const int corenc::method::system_dg_method< Problem, Grid, Matrix >::changeFlux (
            const DGFlux flux_type ) [inline]
```

6.86.2.3 DGtostandart()

```
template<class Problem , class Grid , class Matrix >
const int corenc::method::system_dg_method< Problem, Grid, Matrix >::DGtostandart (
            const std::vector< double > & dg_result ) [inline]
```

6.86.2.4 GetGlobalMatrix()

```
template<class Problem , class Grid , class Matrix >
const Matrix * corenc::method::system_dg_method< Problem, Grid, Matrix >::GetGlobalMatrix ( )
const [inline]
```

6.86.2.5 GetMaxSolution()

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::system_dg_method< Problem, Grid, Matrix >::GetMaxSolution
```

6.86.2.6 GetMinSolution()

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::system_dg_method< Problem, Grid, Matrix >::GetMinSolution
```

6.86.2.7 GetSolution() [1/4]

```
template<class Problem , class Grid , class Matrix >
const std::vector< double > corenc::method::system_dg_method< Problem, Grid, Matrix >::Get←
Solution ( ) const [inline]
```

6.86.2.8 GetSolution() [2/4]

```
template<class Problem , class Grid , class Matrix >
static const double corenc::method::system_dg_method< Problem, Grid, Matrix >::GetSolution (
            const Grid & g,
            const std::vector< double > & dg_sol,
            const Mesh::Point & p )  [inline], [static]
```

6.86.2.9 GetSolution() [3/4]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::system_dg_method< Problem, Grid, Matrix >::GetSolution (
            const std::vector< double > & dg_sol,
            const Mesh::Point & p )  [inline]
```

6.86.2.10 GetSolution() [4/4]

```
template<class Problem , class Grid , class Matrix >
const double corenc::method::system_dg_method< Problem, Grid, Matrix >::GetSolution (
            const std::vector< double > & point ) const
```

6.86.2.11 toDGSolution()

```
template<class Problem , class Grid , class Matrix >
const int corenc::method::system_dg_method< Problem, Grid, Matrix >::toDGSolution (
            const Grid & g,
            std::vector< double > & dg_result ) const  [inline]
```

6.86.2.12 updateWeights()

```
template<class Problem , class Grid , class Matrix >
const int corenc::method::system_dg_method< Problem, Grid, Matrix >::updateWeights (
            const std::vector< double > & dg_result )  [inline]
```

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

- CoreNCFEM/Methods/system_dg_method.h

6.87 corenc::method::system_dg_method< Grid, bool, bool > Class Template Reference

```
#include <system_dg_method.h>
```

Static Public Member Functions

- static const double GetSolution (const Grid &g, const std::vector< double > &dg_sol, const Mesh::Point &p)

6.87.1 Member Function Documentation

6.87.1.1 GetSolution()

```
template<class Grid >
static const double corenc::method::system_dg_method< Grid, bool, bool >::GetSolution (
            const Grid & g,
            const std::vector< double > & dg_sol,
            const Mesh::Point & p )  [inline], [static]
```

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

- CoreNCFEM/Methods/system_dg_method.h

6.88 corenc::test_case_elliptic_fem Class Reference

```
#include <test_case_elliptic_fem.h>
```

Public Member Functions

- test_case_elliptic_fem ()
- ~test_case_elliptic_fem ()
- const int elliptic_fem_2d_tria () const
- const int elliptic_fem_solver () const
- const int elliptic_fem_square_lin_basis () const
- const int elliptic_fem_hp_fixed (const int h_ref_max, const int p_ref_max) const
- const int elliptic_fem_hp_fixed_triangle (const int h_ref_max, const int p_ref_max) const
- const int elliptic_fem_hp_lagrange_triangle (const int h_ref_max, const int p_ref_max) const
- const int elliptic_fem_hxhy_fixed_triangle (const int hx_max, const int hy_max) const
- const int conv_diff_fem_fixed_triangle (const int h_ref_max, const int p_ref_max) const
- const int global_matrix (const int h_ref_max, const int p_ref_max) const
- const int elliptic_2layer_fem_2d_tria_h () const
- const int elliptic_fem_2d_rect_source () const
- const int elliptic_gaussian_triangle () const
- const int mass_matrix_3rd_order () const
- const int strees_matrix_3rd_order () const
- const int mass_matrix_4th_order () const
- const int stress_matrix_4th_order () const
- const int homotopy_conv_diff_fem (const double step) const

6.88.1 Constructor & Destructor Documentation

6.88.1.1 test_case_elliptic_fem()

```
test_case_elliptic_fem::test_case_elliptic_fem ( )
```

6.88.1.2 ∼test_case_elliptic_fem()

```
test_case_elliptic_fem::∼test_case_elliptic_fem ( )
```

6.88.2 Member Function Documentation

6.88.2.1 conv_diff_fem_fixed_triangle()

```
const int test_case_elliptic_fem::conv_diff_fem_fixed_triangle (
            const int h_ref_max,
            const int p_ref_max ) const
```

6.88.2.2 elliptic_2layer_fem_2d_tria_h()

```
const int test_case_elliptic_fem::elliptic_2layer_fem_2d_tria_h ( ) const
```

6.88.2.3 elliptic_fem_2d_rect_source()

```
const int test_case_elliptic_fem::elliptic_fem_2d_rect_source ( ) const
```

6.88.2.4 elliptic_fem_2d_tria()

```
const int test_case_elliptic_fem::elliptic_fem_2d_tria ( ) const
```

6.88.2.5 elliptic_fem_hp_fixed()

```
const int test_case_elliptic_fem::elliptic_fem_hp_fixed (
            const int h_ref_max,
            const int p_ref_max ) const
```

6.88.2.6 elliptic_fem_hp_fixed_triangle()

```
const int test_case_elliptic_fem::elliptic_fem_hp_fixed_triangle (
            const int h_ref_max,
            const int p_ref_max ) const
```

6.88.2.7 elliptic_fem_hp_lagrange_triangle()

```
const int test_case_elliptic_fem::elliptic_fem_hp_lagrange_triangle (
            const int h_ref_max,
            const int p_ref_max ) const
```

6.88.2.8 elliptic_fem_hxhy_fixed_triangle()

```
const int test_case_elliptic_fem::elliptic_fem_hxhy_fixed_triangle (
            const int hx_max,
            const int hy_max ) const
```

6.88.2.9 elliptic_fem_solver()

```
const int test_case_elliptic_fem::elliptic_fem_solver ( ) const
```

6.88.2.10 elliptic_fem_square_lin_basis()

```
const int test_case_elliptic_fem::elliptic_fem_square_lin_basis ( ) const
```

6.88.2.11 elliptic_gaussian_triangle()

```
const int test_case_elliptic_fem::elliptic_gaussian_triangle ( ) const
```

6.88.2.12 global_matrix()

```
const int test_case_elliptic_fem::global_matrix (
            const int h_ref_max,
            const int p_ref_max ) const
```

6.88.2.13 homotopy_conv_diff_fem()

```
const int test_case_elliptic_fem::homotopy_conv_diff_fem (
            const double step ) const
```

6.88.2.14 mass_matrix_3rd_order()

```
const int test_case_elliptic_fem::mass_matrix_3rd_order ( ) const
```

6.88.2.15 mass_matrix_4th_order()

```
const int test_case_elliptic_fem::mass_matrix_4th_order ( ) const
```

6.88.2.16 strees_matrix_3rd_order()

```
const int test_case_elliptic_fem::strees_matrix_3rd_order ( ) const
```

6.88.2.17 stress_matrix_4th_order()

```
const int test_case_elliptic_fem::stress_matrix_4th_order ( ) const
```

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

- Tests/test_case_elliptic_fem.h
- Tests/test_case_elliptic_fem.cpp

6.89 corenc::tests::test_case_rectanglebasis Class Reference

```
#include <test_case_rectanglebasis.h>
```

Public Member Functions

- test_case_rectanglebasis ()
- ~test_case_rectanglebasis ()
- const int mass_matrix () const
- const int stress_matrix () const

6.89.1 Constructor & Destructor Documentation

6.89.1.1 test_case_rectanglebasis()

```
test_case_rectanglebasis::test_case_rectanglebasis ( )
```

6.89.1.2 ~test_case_rectanglebasis()

```
test_case_rectanglebasis::~test_case_rectanglebasis ( )
```

6.89.2 Member Function Documentation

6.89.2.1 mass_matrix()

```
const int test_case_rectanglebasis::mass_matrix ( ) const
```

6.89.2.2 stress_matrix()

```
const int test_case_rectanglebasis::stress_matrix ( ) const
```

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

- Tests/FiniteElements/test_case_rectanglebasis.h
- Tests/FiniteElements/test_case_rectanglebasis.cpp

6.90 corenc::tests::test_case_regular_mesh Class Reference

```
#include <test_case_regular_mesh.h>
```

Public Member Functions

- test_case_regular_mesh ()
- ∼test_case_regular_mesh ()
- const int construct_mesh () const

6.90.1 Constructor & Destructor Documentation

6.90.1.1 test_case_regular_mesh()

```
test_case_regular_mesh::test_case_regular_mesh ( )
```

6.90.1.2 ∼test_case_regular_mesh()

```
test_case_regular_mesh::∼test_case_regular_mesh ( )
```

6.90.2 Member Function Documentation

6.90.2.1 construct_mesh()

```
const int test_case_regular_mesh::construct_mesh ( ) const
```

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

- Tests/test_case_regular_mesh.h
- Tests/test_case_regular_mesh.cpp

6.91 corenc::test_case_solver Class Reference

```
#include <test_case_solver.h>
```

Public Member Functions

- test_case_solver ()
- ∼test_case_solver ()
- const int gauss_solver () const

6.91.1 Constructor & Destructor Documentation

6.91.1.1 test_case_solver()

```
test_case_solver::test_case_solver ( )
```

6.91.1.2 ∼test_case_solver()

```
test_case_solver::∼test_case_solver ( )
```

6.91.2 Member Function Documentation

6.91.2.1 gauss_solver()

```
const int test_case_solver::gauss_solver ( ) const
```

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

- Tests/test_case_solver.h
- Tests/test_case_solver.cpp

6.92 corenc::tests::test_case_trianglebasis Class Reference

```
#include <test_case_trianglebasis.h>
```

Public Member Functions

- test_case_trianglebasis ()
- ∼test_case_trianglebasis ()
- const int mass_matrix () const
- const int stress_matrix () const

6.92.1 Constructor & Destructor Documentation

6.92.1.1 test_case_trianglebasis()

```
test_case_trianglebasis::test_case_trianglebasis ( )
```

6.92.1.2 ~test_case_trianglebasis()

```
test_case_trianglebasis::~test_case_trianglebasis ( )
```

6.92.2 Member Function Documentation

6.92.2.1 mass_matrix()

```
const int test_case_trianglebasis::mass_matrix ( ) const
```

6.92.2.2 stress_matrix()

```
const int test_case_trianglebasis::stress_matrix ( ) const
```

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

- Tests/FiniteElements/test_case_trianglebasis.h
- Tests/FiniteElements/test_case_trianglebasis.cpp

6.93 corenc::test_cases Class Reference

```
#include <test_cases.h>
```

Public Member Functions

- test_cases ()
- ~test_cases ()
- const int perform () const
- const int perform (const std::function< const int()> &) const
- const int perform (const std::function< const int(std::ostream &)> &, std::ostream &) const

6.93.1 Constructor & Destructor Documentation

6.93.1.1 test_cases()

```
test_cases::test_cases ( )
```

6.93.1.2 ~test_cases()

```
test_cases::~test_cases ( )
```

6.93.2 Member Function Documentation

6.93.2.1 perform() [1/3]

```
const int test_cases::perform ( ) const
```

6.93.2.2 perform() [2/3]

```
const int test_cases::perform (
            const std::function< const int()> & f ) const
```

6.93.2.3 perform() [3/3]

```
const int corenc::test_cases::perform (
            const std::function< const int(std::ostream &)> & ,
            std::ostream & ) const
```

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

- Tests/test_cases.h
- Tests/test_cases.cpp

6.94 corenc::test_conv_diff Class Reference

```
#include <test_conv_diff.h>
```

Public Member Functions

- test_conv_diff ()
- ∼test_conv_diff ()
- void conv_diff_fem (const int h_ref_max, const int p_ref_max=1) const
- void conv_diff_eigen (const int h_ref_max, const int p_ref_max=1) const

6.94.1 Constructor & Destructor Documentation

6.94.1.1 test_conv_diff()

```
corenc::test_conv_diff::test_conv_diff ( )  [inline]
```

6.94.1.2 ∼test_conv_diff()

```
corenc::test_conv_diff::∼test_conv_diff ( )  [inline]
```

6.94.2 Member Function Documentation

6.94.2.1 conv_diff_eigen()

```
void test_conv_diff::conv_diff_eigen (
            const int h_ref_max,
            const int p_ref_max = 1 ) const
```

6.94.2.2 conv_diff_fem()

```
void test_conv_diff::conv_diff_fem (
            const int h_ref_max,
            const int p_ref_max = 1 ) const
```

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

- Tests/test_conv_diff.h
- Tests/test_conv_diff.cpp

6.95 corenc::solvers::vector_solution Struct Reference

`#include <dg_solver_shallow_water.h>`

Public Member Functions

- vector_solution ()
- vector_solution (const int _size)

Public Attributes

- std::vector< double > S [3]

6.95.1 Constructor & Destructor Documentation

6.95.1.1 vector_solution() [1/2]

```
corenc::solvers::vector_solution::vector_solution ( )  [inline]
```

6.95.1.2 vector_solution() [2/2]

```
corenc::solvers::vector_solution::vector_solution (
            const int _size )  [inline]
```

6.95.2 Member Data Documentation

6.95.2.1 S

```
std::vector<double> corenc::solvers::vector_solution::S[3]
```

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

- Solvers/dg_solver_shallow_water.h

Chapter 7

File Documentation

7.1 colors.h File Reference

```
#include <string>
#include <iostream>
```

Namespaces

- namespace corenc
- namespace corenc::color

Variables

- const std::string corenc::color::ESCAPE = "\u001b[0m"
- const std::string corenc::color::BLACK = "\u001b[30m"
- const std::string corenc::color::RED = "\u001b[31m"
- const std::string corenc::color::GREEN = "\u001b[32m"
- const std::string corenc::color::YELLOW = "\u001b[33m"
- const std::string corenc::color::BLUE = "\u001b[34m"
- const std::string corenc::color::MAGENTA = "\u001b[35m"
- const std::string corenc::color::CYAN = "\u001b[36m"
- const std::string corenc::color::WHITE = "\u001b[37m"
- const std::string corenc::color::PURPLE = "\e[1;35m"
- const std::string corenc::color::BBLACK = "\u001b[30;1m"
- const std::string corenc::color::BRED = "\u001b[31;1m"
- const std::string corenc::color::BGREEN = "\u001b[32;1m"
- const std::string corenc::color::BYELLOW = "\u001b[33;1m"
- const std::string corenc::color::BBLUE = "\u001b[34;1m"
- const std::string corenc::color::BMAGENTA = "\u001b[35;1m"
- const std::string corenc::color::BCYAN = "\u001b[36;1m"
- const std::string corenc::color::BWHITE = "\u001b[37;1m"

7.2 colors.h

Go to the documentation of this file.

```
#ifndef CORENC_COLORS_H
#define CORENC_COLORS_H
#include <string>
#include <iostream>
namespace corenc
{
    namespace color
    {
        const std::string ESCAPE = "\u001b[0m";
         // 8-bit colors
         const std::string BLACK = "\u001b[30m";
         const std::string RED = "\u001b[31m";
         const std::string GREEN = "\u001b[32m";
         const std::string YELLOW = "\u001b[33m";
         const std::string BLUE = "\u001b[34m";
         const std::string MAGENTA = "\u001b[35m";
         const std::string CYAN = "\u001b[36m";
         const std::string WHITE = "\u001b[37m";
         const std::string PURPLE = "\e[1;35m";
         // 16-bit colors
         const std::string BBLACK = "\u001b[30;1m";
         const std::string BRED = "\u001b[31;1m";
         const std::string BGREEN = "\u001b[32;1m";
         const std::string BYELLOW = "\u001b[33;1m";
         const std::string BBLUE = "\u001b[34;1m";
         const std::string BMAGENTA = "\u001b[35;1m";
         const std::string BCYAN = "\u001b[36;1m";
         const std::string BWHITE = "\u001b[37;1m";
         static void color_output(const std::string& text, const std::string& col)
         {
             std::cout « col « text « ESCAPE « std::endl;
         }
         static void color_output(std::ostream& os, const std::string& text, const std::string& col)
         {
             os « col « text « ESCAPE « std::endl;
         }
     }
 }
 #endif // CORENC_COLORS_H
```

7.3 CoreNCA/Matrix.cpp File Reference

```
#include "Matrix.h"
```

7.4 CoreNCA/Matrix.h File Reference

```
#include <vector>
#include <set>
```

Classes

- class Algebra::Matrix

Namespaces

- namespace Algebra

7.5 Matrix.h

Go to the documentation of this file.

```
#ifndef CORENC_ALGEBRA_MATRIX_H
#define CORENC_ALGEBRA_MATRIX_H
#include <vector>
#include <set>
namespace Algebra
{
    class ESolver;
     class Matrix
     {
     public:
         Matrix(const unsigned int& size, const std::vector<std::set<unsigned int»& nonzero);
         Matrix() {};
         ~Matrix();
         void                                NullRow(const int row);
         double& operator()(const int i, const int j)
         {
             return (*this).m_elem[i][j];
         }
         const int                           GetSize() const { return m_size; }
         void                                NullMatrix();
         Matrix&                               operator=(const Matrix&);
         Matrix(const Matrix& matrix);
         void                                Create(const unsigned int& size, const std::vector<std::set<unsigned int»& nonzero);
         void                                Create(const unsigned int& size);
         const double                        GetElement(const int i, const int j)
         {
             return m_elem[i][j];
         }
         void                                AddElement(const unsigned int i, const unsigned int j, const double a)
         {
             m_elem[i][j] += a;
             return;
         }
     private:
         std::vector<std::vector<double»        m_elem;
         unsigned int                        m_size{ 0 };
         friend                              ESolver;
     };
 }

 #endif // !CORENC_ALGEBRA_MATRIX_H
```

7.6 CoreNCA/MatrixDiag.cpp File Reference

```
#include "MatrixDiag.h"
```

7.7 CoreNCA/MatrixDiag.h File Reference

```
#include <set>
#include <vector>
```

Classes

- class Algebra::MatrixDiag

Namespaces

- namespace Algebra

7.8 MatrixDiag.h

Go to the documentation of this file.

```
#ifndef CORENC_ALGEBRA_MATRIXDIAG_H
#define CORENC_ALGEBRA_MATRIXDIAG_H
#include <set>
#include <vector>

namespace Algebra
{
    class ESolver;
     class MatrixDiag
     {
     public:
         MatrixDiag(const unsigned int& size, const std::vector<std::set<unsigned int»& nonzero);
         MatrixDiag() {};
         ~MatrixDiag();
         void                                NullRow(const int row);
         double& operator()(const int i, const int j)
         {
             return (*this).m_valDiag[i];
         }
         const int                           GetSize() const { return m_size; }
         void                                NullMatrix();
         MatrixDiag&                           operator=(const MatrixDiag&);
         MatrixDiag(const MatrixDiag& matrix);
         void                                Create(const unsigned int& size, const
     std::vector<std::set<unsigned int»& nonzero);
         void                                AddElement(const unsigned int i, const unsigned int j,
     const double a)
         {
             if (i == j)
             {
                 m_valDiag[i] += a;
                 return;
             }
             return;
         }
     private:
         std::vector<double>                 m_valDiag;
         unsigned int                        m_size{ 0 };
         friend                              ESolver;
     };
 }

 #endif // !CORENC_ALGEBRA_MATRIXDIAG_H
```

7.9 CoreNCA/MatrixSkyline.cpp File Reference

```
#include "MatrixSkyline.h"
#include <iostream>
#include <fstream>
#include <ostream>
#include <ctime>
#include <cmath>
```

Macros

- #define N_MIN 4096
- #define _NOPE_

7.9.1 Macro Definition Documentation

7.9.1.1 _NOPE_

```
#define _NOPE_
```

7.9.1.2 N_MIN

```
#define N_MIN 4096
```

7.10 CoreNCA/MatrixSkyline.h File Reference

```
#include <set>
#include <vector>
#include <memory>
#include "Matrix.h"
#include "MatrixDiag.h"
```

Classes

- class Algebra::MatrixSkyline
- class Algebra::ESolver

Namespaces

- namespace Algebra

Enumerations

- enum class Algebra::Solvers {
  Algebra::BiCGStab , Algebra::GMRES , Algebra::GMRES_BiCGStab , Algebra::Gauss ,
  Algebra::PARDISO }

7.11 MatrixSkyline.h

Go to the documentation of this file.

```cpp
#ifndef CORENC_ALGEBRA_MATRIXSKYLINE_H_
#define CORENC_ALGEBRA_MATRIXSKYLINE_H_
#include <set>
#include <vector>
#include <memory>
#include "Matrix.h"
#include "MatrixDiag.h"
namespace Algebra
{
    class ESolver;
    enum class Solvers
    {
        BiCGStab,
        GMRES,
        GMRES_BiCGStab,
        Gauss,
        PARDISO
    };
    class MatrixSkyline
    {
    public:
        MatrixSkyline(const unsigned int& Size, const std::vector<std::set<unsigned int»& nonzero);
        MatrixSkyline(){};
        ~MatrixSkyline();
        void                NullRow(int row);
        double& operator()(const int i, const int j)
        {
            int ind;
            /*for (ind = m_rowptr[i]; ind < m_rowptr[i + 1]; ++ind)
                if (m_colind[ind] == j)
                    break;
            return (*this).m_valL[ind];*/
            if (i == j)
            {
                return (*this).m_valDiag[i];
            }
            if (i < j)
            {
                for (ind = m_rowptr[j]; ind < m_rowptr[j + 1]; ++ind)
                    if (m_colind[ind] == i)
                        break;
                return (*this).m_valU[ind];
            }
            else
            {
                for (ind = m_rowptr[i]; ind < m_rowptr[i + 1]; ++ind)
                    if (m_colind[ind] == j)
                        break;
                return (*this).m_valL[ind];
            }
            //return (*this)[];
        }
        const double operator()(const int i, const int j) const
        {
            int ind;
            /*for (ind = m_rowptr[i]; ind < m_rowptr[i + 1]; ++ind)
                if (m_colind[ind] == j)
                    break;
            return (*this).m_valL[ind];*/
            if (i == j)
            {
                return (*this).m_valDiag[i];
            }
            if (i < j)
            {
                for (ind = m_rowptr[j]; ind < m_rowptr[j + 1]; ++ind)
                    if (m_colind[ind] == i)
                        break;
                if (ind < m_rowptr[j + 1])
                    return (*this).m_valU[ind];
                return 0;
            }
            else
            {
                for (ind = m_rowptr[i]; ind < m_rowptr[i + 1]; ++ind)
                    if (m_colind[ind] == j)
                        break;
                if (ind < m_rowptr[i + 1])
                    return (*this).m_valL[ind];
                return 0;
            }
            return 0;
```

```
        }
        const double        GetElement(const int i, const int j) const
        {
            int ind;
            /*for (ind = m_rowptr[i]; ind < m_rowptr[i + 1]; ++ind)
                if (m_colind[ind] == j)
                    break;
            return (*this).m_valL[ind];*/
            if (i == j)
            {
                return (*this).m_valDiag[i];
            }
            if (i < j)
            {
                for (ind = m_rowptr[j]; ind < m_rowptr[j + 1]; ++ind)
                    if (m_colind[ind] == i)
                        break;
                if (ind < m_rowptr[j + 1])
                    return (*this).m_valU[ind];
                return 0;
            }
            else
            {
                for (ind = m_rowptr[i]; ind < m_rowptr[i + 1]; ++ind)
                    if (m_colind[ind] == j)
                        break;
                if (ind < m_rowptr[i + 1])
                    return (*this).m_valL[ind];
                return 0;
            }
            return 0;
        }
        const int           GetSize() const { return m_size; }
        void                NullMatrix();
        MatrixSkyline&      operator=(const MatrixSkyline&);
        //MatrixSkyline&        operator-(const MatrixSkyline&);
        //friend const MatrixSkyline operator-(const MatrixSkyline&, const MatrixSkyline&);
        // A - scal * B
        static const MatrixSkyline diff_skymatrix(const MatrixSkyline& matrix, const MatrixSkyline& B, const double scal)
        {
            MatrixSkyline C(matrix);
            if ((B.m_gsize == matrix.m_gsize) && (matrix.m_size == B.m_size))
            {
                for (int i = 0; i < B.m_gsize; ++i)
                {
                    C.m_valL[i] = matrix.m_valL[i] - scal * B.m_valL[i];
                    C.m_valU[i] = matrix.m_valU[i] - scal * B.m_valU[i];
                }
                for (int i = 0; i < B.m_size; ++i)
                {
                    C.m_valDiag[i] = matrix.m_valDiag[i] - scal * B.m_valDiag[i];
                }
            }
            return C;
        }
        static const MatrixSkyline diff_skymatrix(const MatrixSkyline& matrix, const MatrixSkyline& B, const double a, const double b)
        {
            MatrixSkyline C(matrix);
            if ((B.m_gsize == matrix.m_gsize) && (matrix.m_size == B.m_size))
            {
                for (int i = 0; i < B.m_gsize; ++i)
                {
                    C.m_valL[i] = a * matrix.m_valL[i] - b * B.m_valL[i];
                    C.m_valU[i] = a * matrix.m_valU[i] - b * B.m_valU[i];
                }
                for (int i = 0; i < B.m_size; ++i)
                {
                    C.m_valDiag[i] = a * matrix.m_valDiag[i] - b * B.m_valDiag[i];
                }
            }
            return C;
        }
        static const MatrixSkyline transpose_sky(const MatrixSkyline& matrix)
        {
            MatrixSkyline C(matrix);
            C.m_valL = matrix.m_valU;
            C.m_valU = matrix.m_valL;
            return C;
        }
        MatrixSkyline(const MatrixSkyline& matrix);
        void                Create(const unsigned int& Size, const std::vector<std::set<unsigned int>>& nonzero);
        void                AddElement(const unsigned int i, const unsigned int j, const double a)
        {
            int ind;
```

```
            /*for (ind = m_rowptr[i]; ind < m_rowptr[i + 1]; ++ind)
             if (m_colind[ind] == j)
             break;
             return (*this).m_valL[ind];*/
            if (i == j)
            {
                m_valDiag[i] += a;
                return;
            }
            if (i < j)
            {
                for (ind = m_rowptr[j]; ind < m_rowptr[j + 1]; ++ind)
                    if (m_colind[ind] == i)
                        break;
                m_valU[ind] += a;
                return;
            }
            else
            {
                for (ind = m_rowptr[i]; ind < m_rowptr[i + 1]; ++ind)
                    if (m_colind[ind] == j)
                        break;
                m_valL[ind] += a;
                return;
            }
        }

    private:
        //int*                m_rowptr = nullptr;
        std::vector<int>    m_rowptr;
        //int*                m_colind = nullptr;
        std::vector<int>    m_colind;
        //double*                 m_valU = nullptr;
        std::vector<double> m_valU;
        //double*                 m_valL = nullptr;
        std::vector<double> m_valL;
        //double*                 m_valDiag = nullptr;
        std::vector<double> m_valDiag;
        int                 m_size;
        int                 m_gsize;
        friend              ESolver;
        //friend                  Matrix;
    };
    class ESolver
    {
    public:
        ESolver(const MatrixSkyline& matrix, const std::vector<double>& rightvector) :
            m_matrix{ matrix },
            m_rightvector( rightvector ),
            m_maxiter(20000),
            m_eps(1e-10)
        {
            m_solution.resize(matrix.m_size);
        }
        ESolver(){};
        ESolver(Solvers kek) :m_solver(kek){};
        void                    Reload(const MatrixSkyline& matrix, const std::vector<double>& right);
        void                    Solve(Solvers);
        const std::vector<double>       Solve(MatrixSkyline&, const std::vector<double>& rhs, std::vector<double>& sol, std::vector<double>& residual, const int iter, const double eps);
        const std::vector<double>       Solve(MatrixDiag&, const std::vector<double>& rhs, std::vector<double>& sol, std::vector<double>& residual, const int iter, const double eps);
        double                  BiCGStab(const int _maxiter);
        double                  GMRES(const int _maxiter);
        void                    GMRES(MatrixSkyline&, const std::vector<double>& rhs, std::vector<double>& sol, std::vector<double>& residual, const int iter, const double eps);
        void                    BiCGStab(MatrixSkyline&, const std::vector<double>& rhs, std::vector<double>& sol, std::vector<double>& residual, const int iter, const double eps);
        void                    Gauss(MatrixSkyline&, const std::vector<double>& rhs, std::vector<double>& sol, std::vector<double>& residual, const int iter, const double eps);
        void                    Gauss(Matrix&, const std::vector<double>& rhs, std::vector<double>& sol);
        void                    Gauss(const Matrix&, double* in_out);
        void                    Gauss(const Matrix&, double* in, double* out);
        void                    Gauss(const Matrix&, const double* in, double* out);
        void                    Pardiso(MatrixSkyline&, const std::vector<double>& rhs, std::vector<double>& sol);
        void                    BiCGStabPrecond();
        const std::vector<double>   GetSolution() const{ return m_solution; }
        void                    GetSolution(std::vector<double>& sol) const;
        void                    MatrixprodVector(double*res, std::vector<double>& x, MatrixSkyline& m);
        void                    MatrixprodVector(double*res, double* x, MatrixSkyline& m);
        void                    MatrixprodVector(double*res, double* x, const Matrix& m);
        void                    MatrixprodVector(double*res, const double* x, const Matrix& m);
        ~ESolver();
    private:
        MatrixSkyline           m_matrix;
```

```
        std::vector<std::vector<double» H, V, W;
        std::vector<double>      m_solution;
        std::vector<double>      m_rightvector;
        void                     GMRESBiCGStab();
        void                     MatrixprodVector(double*res, std::vector<double>& x, double*valDiag, double*valL, double*valU, int*rowptr, int*colind, int size);
        void                     MatrixprodVector(double*res, std::vector<double>& x,
                                                  std::vector<double>&valDiag,
                                                  std::vector<double>&valL,
                                                  std::vector<double>&valU,
                                                  std::vector<int>&rowptr,
                                                  std::vector<int>&colind, int size);
        //void                   MatrixprodVector(double*res, std::vector<double>& x, MatrixSkyline& m);
        void                     MatrixprodVector(double*res, double* x,
                                                  std::vector<double>&valDiag,
                                                  std::vector<double>&valL,
                                                  std::vector<double>&valU,
                                                  std::vector<int>&rowptr,
                                                  std::vector<int>&colind, int size);
        void                     MatrixprodVector(double*res, double* x, double*valDiag, double*valL, double*valU, int*rowptr, int*colind, int size);
        void                     MatrixprodVector(double* res, double* x, double* val, int* rowptr, int* colind, int size);
        const double             DotProd(double*, double*, int);
        void                     zero_GMRES(std::vector<std::vector<double»&, const int str, const int stl);
        void                     mult_Ht_H_slae(double, double*, int);
        void                     gauss(std::vector<std::vector<double»&, double*, double*, int);
        int                      find_max(std::vector<std::vector<double»&, int, int);
        void                     Copy(double *x, double *y, int n);
        void                     mult_Vy(double*, double*, int);
        void                     mult_Vy(double*, double*, int, int);
        const double             DotProd(const std::vector<double>&, const std::vector<double>&, int);
        const double             Norm(double*, int);
        const double             Norm(const std::vector<double>&, int);
        double                   m_eps;
        Solvers                   m_solver;
        int                      m_maxiter;
        double*                  m_LUvalL = nullptr;
        double*                  m_LUvalU = nullptr;
        double*                  m_LUvalDiag = nullptr;
        void                     LUPrec();
        void                     LSolve(double*, double*);
        void                     USolve(double*, double*);
    public:
        auto                     GetSolution() -> decltype(m_solution) { return m_solution; }
    };
}
#endif // CORENC_ALGEBRA_MATRIXSKYLINE_H_
```

7.12 CoreNCFEM/FESolution.h File Reference

```
#include <vector>
#include "Point.h"
```

Classes

- class corenc::CSolution
- class corenc::CFESolution
- class corenc::CVecSolution
- class corenc::CFEweights

Namespaces

- namespace corenc

7.13 FESolution.h

Go to the documentation of this file.

```
#ifndef CORENC_FESOLUTION_H
#define CORENC_FESOLUTION_H
#include <vector>
#include "Point.h"
namespace corenc
{
    class CSolution
    {
    public:
        CSolution() {};
        virtual ~CSolution() {};
    };

    class CFESolution :public CSolution
    {
    public:
        CFESolution() :m_w{ 0 } {};
        ~CFESolution() {}
        CFESolution& operator=(const CFESolution& fe)
        {
            m_w = fe.m_w;
            return *this;
        }
        CFESolution& operator=(const double fe)
        {
            m_w = fe;
            return *this;
        }
        CFESolution(const CFESolution& fe) :m_w{ fe.m_w } {}
        CFESolution(const double& fe) : m_w{ fe } {}
        operator double() const { return m_w; }
        /*double& operator=(const double fe)
        {
            m_w = fe;
            return m_w;
        }*/
        const bool operator==(const CFESolution& fe)
        {
            if (fe.m_w == m_w)
                return true;
            return false;
        }
        const bool operator!=(const CFESolution& fe)
        {
            if (fe.m_w != m_w)
                return true;
            return false;
        }
        CFESolution& operator+=(const CFESolution& fe)
        {
            m_w += fe.m_w;
            return *this;
        }
        CFESolution& operator-=(const CFESolution& fe)
        {
            m_w -= fe.m_w;
            return *this;
        }
        CFESolution& operator*=(const CFESolution& fe)
        {
            m_w *= fe.m_w;
            return *this;
        }
        CFESolution& operator/=(const CFESolution& fe)
        {
            m_w /= fe.m_w;
            return *this;
        }
        friend const double operator*(const CFESolution& lhs, const CFESolution& rhs)
        {
            return lhs.m_w * rhs.m_w;
        }
        friend const double operator*(const CFESolution& lhs, const double rhs)
        {
            return lhs.m_w * rhs;
        }
        friend const double operator*(const double lhs, const CFESolution& rhs)
        {
            return lhs * rhs.m_w;
        }
        friend const double operator-(const CFESolution& lhs, const CFESolution& rhs)
        {
```

```
           return lhs.m_w - rhs.m_w;
       }
       friend const double operator+(const CFESolution& lhs, const CFESolution& rhs)
       {
           return lhs.m_w + rhs.m_w;
       }
       friend const double operator/(const CFESolution& lhs, const CFESolution& rhs)
       {
           return lhs.m_w / rhs.m_w;
       }
   private:
        double m_w;
    };
    class CVecSolution :public CSolution
    {
    public:
        CVecSolution() :m_w{ 0 } {};
        ~CVecSolution() {}
        std::vector<double> m_w;
    };

    class CFEweights
    {
    public:
        CFEweights() {};
        ~CFEweights()
        {
            if (m_w.size() > 0)
                std::vector<CFESolution>().swap(m_w);
        };
        const CFESolution                getWeight(const unsigned int i) const { return m_w[i]; };
        const int                       updateWeight(const unsigned int i, const CFESolution& cfe)
        {
            if (i < m_w.size())
            {
                m_w[i] = cfe;
                return 0;
            }
            return 1;
        }
    private:
        std::vector<CFESolution>        m_w;
    };
}

#endif // !CORENC_FESOLUTION_H

```

7.14 CoreNCFEM/FiniteElements/CRectangleBasis2x.cpp File Reference

```
#include "Rectangle.h"
#include <iostream>
```

7.15 CoreNCFEM/FiniteElements/Cube.cpp File Reference

```
#include "Cube.h"
```

7.16 CoreNCFEM/FiniteElements/Cube.h File Reference

```
#include <stdio.h>
#include "Shape.h"
#include "ShapeFunction.h"
#include <iostream>
```

Classes

- class corenc::Mesh::CCube
- class corenc::Mesh::CCubeBasis

Namespaces

- namespace corenc
- namespace corenc::Mesh

Macros

- #define CORENC_MESH_CUBE_H_

7.16.1 Macro Definition Documentation

7.16.1.1 CORENC_MESH_CUBE_H_

```
#define CORENC_MESH_CUBE_H_
```

7.17 Cube.h

Go to the documentation of this file.

```
#pragma once
#ifndef CORENC_MESH_CUBE_H_
#define CORENC_MESH_CUBE_H_

#include <stdio.h>
#include "Shape.h"
#include "ShapeFunction.h"
#include <iostream>
namespace corenc
{
    namespace Mesh
    {
        class CCube : public CShape
        {
        public:
            CCube();
            CCube(const int n1, const int n2, const int n3, const int n4, const int order);
            CCube(const int n1, const int n2, const int n3, const int n4, const int e1, const int e2, const int e3, const int e4, const int order);
            CCube(const int*, const int order);
            CCube(const int*, const int*, const int order);
            CCube(const CCube&);
            CCube& operator=(const CCube& t)
            {
                m_nodes = t.m_nodes;
                m_edges[0] = t.m_edges[0];
                m_edges[1] = t.m_edges[1];
                m_edges[2] = t.m_edges[2];
                m_edges[3] = t.m_edges[3];
                m_number = t.m_number;
                m_order = t.m_order;
                m_px = t.m_px;
                m_py = t.m_py;
                return *this;
            }
            const bool    operator==(const CCube& t)
```

```
        {
            for (unsigned int i = 0; i < 4; ++i)
                if (m_nodes[i] == t.m_nodes[0])
                    for (unsigned int j = 0; j < 4; ++j)
                        if (m_nodes[j] == t.m_nodes[1])
                            for (unsigned int k = 0; k < 4; ++k)
                                if (m_nodes[k] == t.m_nodes[2])
                                    for (unsigned int l = 0; l < 4; ++l)
                                        if (m_nodes[l] == t.m_nodes[3])
                                    return true;
            return false;
        }
        std::istream& operator»(std::istream& is)
        {
            is » m_nodes[0] » m_nodes[1] » m_nodes[2] » m_nodes[3];
            return is;
        }
        ~CCube() {};
        const int                                           GetNode(const int) const;
        const int                                           GetNode(const NODES&) const;
        const int                                           GetEdge(const int) const;
        const int                                           GetFacet(const int) const;
        const int                                           GetNumberOfNodes() const;
        const int                                           GetNumberOfEdges() const;
        const int                                           GetNumberOfFacets() const;
        const double                                        Integrate(const std::function<const double(const Point&)>&, const std::vector<Point>& v) const;
        const Point                                            Integrate(const std::function<const Point(const Point&)>&, const std::vector<Point>& v) const;
        const std::vector<double>                           Integrate(const std::function<const std::vector<double>(const Point&)>&, const std::vector<Point>&) const;
        void                                                SetNode(const int k, const int node);
        const int                                           IncreaseOrder();
        const int                                           SetOrder(const int px, const int py);
        void                                                SetEdge(const int k, const int edge);
        void                                                SetFacet(const int k, const int facet);
    private:
        std::vector<int>                                    m_nodes;
        int                                                 m_edges[4];
        int                                                 m_order;
        int                                                 m_number;
        int                                                 m_px, m_py;
    };

    class CCubeBasis : public CShapeFunction<double>
    {
    public:
        CCubeBasis();
        CCubeBasis(const Point&, const Point&, const Point&, const Point&, const int order);
        CCubeBasis(const Point*, const int order);
        CCubeBasis(const CCubeBasis&);
        CCubeBasis& operator=(const CCubeBasis& t)
        {
            m_normal = t.m_normal;
            m_det = t.m_det;
            m_order = t.m_order;
            m_1dorder = t.m_1dorder;
            m_number = t.m_number;
            m_s = t.m_s;
            m_sp = t.m_sp;
            m_points = t.m_points;
            m_hx = t.m_hx;
            m_hy = t.m_hy;
            return *this;
        }
        ~CCubeBasis() {};
        const int                                           GetNumberOfShapeFunctions() const;
         //const DForm<0>*                                         GetShapeFunction(const int, const Point&) const;
         const double                                        GetShapeFunction(const int, const Point&) const;
         const Point                                             GetGradShapeFunction(const int, const Point&) const;
         const Point                                             GetNormal() const;
         void                                                ReverseNormal();
         const double                                        GetValue(const Point&) const;
         const int                                           IncreaseOrder();
         //const int                                         SetValue(const int, CSolution* value);
         //CSolution*                                        GetValue(const unsigned int);
         //const CFESolution                                 GetValue(const int) const;
         const double                                        GetMeasure() const { return m_det; };
         const double                                        GetWeight(const int, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const;
         //const unsigned int                                        GetOrder() const;
         //const std::function<const DForm<0>*(const Point&)>  GetShapeFunction(const int) const;
     private:
         int                                                 m_number;
```

```
            int                                         m_order;
            int                                         m_1dorder;
            Point                                       m_normal;
            std::vector<Mesh::Point>                    m_points;
            double                                      m_det;
            double                                      m_hx, m_hy;
            const double                                m_x1(const double) const;
            const double                                m_x2(const double) const;
            const double                                m_y1(const double) const;
            const double                                m_y2(const double) const;
            void                                        compD(const Point&, const Point&, const
    Point&);
            void                                        compNormal(const Point&, const Point&,
    const Point&);
            const int                                   createS();
            //std::vector<double>                           m_w[m_number];
            //std::vector<CFESolution>                  m_w{ 4 };
            int                                         m_s;
            int                                         m_sp;
        };
    }
}
#endif // CORENC_MESH_Cube_H_
```

7.18 CoreNCFEM/FiniteElements/CubeHBasis.cpp File Reference

7.19 CoreNCFEM/FiniteElements/Edge.cpp File Reference

```
#include "Edge.h"
#include <iostream>
```

7.20 CoreNCFEM/FiniteElements/Edge.h File Reference

```
#include <stdio.h>
#include "Shape.h"
#include "ShapeFunction.h"
#include <iostream>
#include "../FESolution.h"
```

Classes

- class corenc::Mesh::CEdge
- class corenc::Mesh::CEdgeLinearBasis
- class corenc::Mesh::CEdgeConstantBasis
- class corenc::Mesh::CEdgeMultiBasis
- class corenc::Mesh::CEdgeHermiteBasis
- class corenc::Mesh::CEdge2ndBasis

Namespaces

- namespace corenc
- namespace corenc::Mesh

7.21 Edge.h

Go to the documentation of this file.

```
#ifndef Edge_hpp
#define Edge_hpp

#include <stdio.h>
#include "Shape.h"
#include "ShapeFunction.h"
#include <iostream>
#include "../FESolution.h"
namespace corenc
{
    namespace Mesh
    {
        class CEdge : public CShape
        {
        public:
            CEdge();
            CEdge(const CEdge&);
            CEdge(const int n1, const int n2);
            CEdge(const int*);
            CEdge& operator=(const CEdge& e)
            {
                m_nodes = e.m_nodes;
                m_number = e.m_number;
                return *this;
            }
            friend const bool operator==(const CEdge& e1, const CEdge& e2)
            {
                if (e1.m_nodes[0] == e2.m_nodes[0])
                    if (e1.m_nodes[1] == e2.m_nodes[1])
                        return true;
                if (e1.m_nodes[1] == e2.m_nodes[0])
                    if (e1.m_nodes[0] == e2.m_nodes[1])
                        return true;
                return false;
            }
            friend std::istream& operator»(std::istream& is, CEdge& e)
            {
                is » e.m_nodes[0] » e.m_nodes[1];
                --e.m_nodes[0]; --e.m_nodes[1];
                return is;
            }
            ~CEdge() {};
            const int                          GetNode(const int) const;
            const int                          GetNode(const NODES&) const;
            const int                          GetNumberOfNodes() const;
            void                               SetNode(const int k, const int node);
            const double                       Integrate(const std::function<const double(const Point&)>&, const std::vector<Point>& v) const;
            const Point                           Integrate(const std::function<const Point(const Point&)>&, const std::vector<Point>& v) const;
            const int                          IncreaseOrder();
            const std::vector<double>          Integrate(const std::function<const std::vector<double>(const Point&)>&, const std::vector<Point>&) const;
        private:
            int                                m_number;
            std::vector<int>                   m_nodes;
        };

        class CEdgeLinearBasis : public CShapeFunction<double>
        {
        public:
            CEdgeLinearBasis();
            CEdgeLinearBasis(const Point&, const Point&);
            CEdgeLinearBasis(const Point*);
            CEdgeLinearBasis(const CEdgeLinearBasis&);
            CEdgeLinearBasis& operator=(const CEdgeLinearBasis& e)
            {
                m_number = e.m_number;
                m_p0 = e.m_p0;
                m_p1 = e.m_p1;
                m_normal = e.m_normal;
                m_mes = e.m_mes;
                return *this;
            }
            ~CEdgeLinearBasis() {};
            const int                          GetNumberOfShapeFunctions() const;
            //const DForm<0>*                     GetShapeFunction(const int) const;
            const double                       GetShapeFunction(const int, const Point&) const;
            const Point                           GetGradShapeFunction(const int, const Point&) const;
```

```
        const Point                                          GetNormal() const;
        void                                                 ReverseNormal();
        //const int                                          SetValue(const int, CSolution* value);
        //const int                                          SetValue(const int, const CFESolution& value);
        //const CFESolution                                  GetValue(const Point&) const;
        //const CFESolution                                  GetValue(const int) const;
        const int                                            IncreaseOrder();
        const double                                         GetMeasure() const { return m_mes; };
        const double                                         GetWeight(const int, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const;
        //CSolution*                                     GetValue(const unsigned int);
        //const std::function<const DForm<0>*(const Point&)>         GetShapeFunction(const int) const;
    private:
        int                                                  m_number;
        Point                                               m_p0, m_p1;
        Point                                               m_normal;
        double                                               m_mes;
        void                                                 CompNormal();
        void                                                 CompLenght();
        //std::vector<double>                                    m_w[2];
        //std::vector<CFESolution>                           m_w;

        //const std::function<const double(const Point&p)>      m_psi[2];
    };



    class CEdgeConstantBasis : public CShapeFunction<double>
    {
    public:
        CEdgeConstantBasis();
        CEdgeConstantBasis(const Point&, const Point&);
        CEdgeConstantBasis(const Point*);
        CEdgeConstantBasis(const CEdgeConstantBasis&);
        CEdgeConstantBasis& operator=(const CEdgeConstantBasis& e)
        {
            m_p0 = e.m_p0;
            m_p1 = e.m_p1;
            m_normal = e.m_normal;
            m_mes = e.m_mes;
            return *this;
        }
        ~CEdgeConstantBasis() {};
        const int                                            GetNumberOfShapeFunctions() const;
        //const DForm<0>*                                        GetShapeFunction(const int) const;
        const double                                         GetShapeFunction(const int, const Point&) const;
        const Point                                             GetGradShapeFunction(const int, const Point&) const;
        const Point                                             GetNormal() const;
        void                                                 ReverseNormal();
        const double                                         GetWeight(const int node, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const
        {
            return f(verts[node]);
        };
        //const double                                       GetValue(const Point&) const;
        //const int                                          SetValue(const unsigned int, const double& value);
        //const double                                       GetValue(const unsigned int) const;
        //const int                                          SetValue(const int, CSolution* value);
        //const int                                          SetValue(const int, const CFESolution& value);
        const int                                            IncreaseOrder() { return 1; };
        //const CFESolution                                  GetValue(const Point&) const;
        //CSolution*                                     GetValue(const unsigned int);
        //const CFESolution                                  GetValue(const int) const;
        const double                                         GetMeasure() const { return 0.; };
        //const std::function<const DForm<0>*(const Point&)>         GetShapeFunction(const int) const;
    private:
        static const int                                     m_number = 1;
        Point                                               m_p0;
        Point                                               m_p1;
        Point                                               m_normal;
        double                                               m_mes;
        void                                                 CompNormal();
        void                                                 CompLenght();
        //double                                                 m_w;
        //CFESolution                                            m_w;
        //const std::function<const double(const Point&p)>      m_psi[2];
    };
```

```

        class CEdgeMultiBasis : public CShapeFunction<double>
        {
        public:
            CEdgeMultiBasis();
            CEdgeMultiBasis(const Point&, const Point&);
            CEdgeMultiBasis(const Point*);
            CEdgeMultiBasis(const CEdgeMultiBasis&);
            CEdgeMultiBasis& operator=(const CEdgeMultiBasis& e)
            {
                m_p0 = e.m_p0;
                m_p1 = e.m_p1;
                m_normal = e.m_normal;
                m_mes = e.m_mes;
                //m_w = e.m_w;
                return *this;
            }
            ~CEdgeMultiBasis() {};
            const int                                                GetNumberOfShapeFunctions() const;
            //const DForm<0>*                                            GetShapeFunction(const int) const;
            const double                                             GetShapeFunction(const int, const Point&) const;
            const Point                                                 GetGradShapeFunction(const int, const Point&) const;
            const Point                                                 GetNormal() const;
            void                                                     ReverseNormal();
            const double                                             GetWeight(const int node, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const
            {
                return f(verts[node]);
            };
            //const int                                              SetValue(const  int, CSolution* value);
            const int                                                IncreaseOrder() { return 1; };
            //const int                                              SetValue(const int, const CFESolution& value);
            //const CFESolution                                      GetValue(const Point&) const;
            //const CFESolution                                      GetValue(const int) const;
            const double                                             GetMeasure() const { return 0.; };
            //const std::function<const DForm<0>*(const Point&)>            GetShapeFunction(const int) const;
        private:
            static const int                                         m_number = 2;
            Point                                                   m_p0;
            Point                                                   m_p1;
            Point                                                   m_normal;
            double                                                   m_mes;
            void                                                     CompNormal();
            void                                                     CompLenght();
            //std::vector<double>                                        m_w[m_number];
            //std::vector<CFESolution>                               m_w;
            //const std::function<const double(const Point&p)>       m_psi[2];
        };

        class CEdgeHermiteBasis : public CShapeFunction<double>
        {
        public:
            CEdgeHermiteBasis();
            CEdgeHermiteBasis(const Point&, const Point&);
            CEdgeHermiteBasis(const Point*);
            CEdgeHermiteBasis(const CEdgeHermiteBasis&);
            CEdgeHermiteBasis& operator=(const CEdgeHermiteBasis& e)
            {
                m_p0 = e.m_p0;
                m_p1 = e.m_p1;
                m_normal = e.m_normal;
                m_mes = e.m_mes;
                //m_w = e.m_w;
                return *this;
            }
            ~CEdgeHermiteBasis() {};
            const int                                                GetNumberOfShapeFunctions() const;
            //const DForm<0>*                                            GetShapeFunction(const int) const;
            const double                                             GetShapeFunction(const int, const Point&) const;
            const Point                                                 GetGradShapeFunction(const int, const Point&) const;
            const int                                                IncreaseOrder() { return 1; };
            const Point                                                 GetNormal() const;
            void                                                     ReverseNormal();
            const double                                             GetWeight(const int node, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const
            {
                return f(verts[node]);
            };
```

```
            //const int                                         SetValue(const int, CSolution* value);
            //const int                                         SetValue(const int, const CFESolution& value);
            //const CFESolution                                 GetValue(const Point&) const;
            //const CFESolution                                 GetValue(const int) const;
            const double                                        GetMeasure() const { return 0.; };
            //const std::function<const DForm<0>*(const Point&)>        GetShapeFunction(const int) const;
        private:
            static const int                                    m_number = 4;
            Point                                              m_p0;
            Point                                              m_p1;
            Point                                              m_normal;
            double                                              m_mes;
            void                                                CompNormal();
            void                                                CompLenght();
            //std::vector<double>                                   m_w[m_number];
            //std::vector<CFESolution>                          m_w;
            //const std::function<const double(const Point&p)>  m_psi[2];
        };

        class CEdge2ndBasis : public CShapeFunction<double>
        {
        public:
            CEdge2ndBasis();
            CEdge2ndBasis(const Point&, const Point&);
            CEdge2ndBasis(const Point*);
            CEdge2ndBasis(const CEdge2ndBasis&);
            CEdge2ndBasis& operator=(const CEdge2ndBasis& e)
            {
                m_p0 = e.m_p0;
                m_p1 = e.m_p1;
                m_normal = e.m_normal;
                m_mes = e.m_mes;
                return *this;
            }
            ~CEdge2ndBasis() {};
            const int                                           GetNumberOfShapeFunctions() const;
            //const DForm<0>*                                       GetShapeFunction(const int) const;
            const double                                        GetShapeFunction(const int, const Point&) const;
            const Point                                            GetGradShapeFunction(const int, const Point&) const;
            const Point                                            GetNormal() const;
            void                                                ReverseNormal();
            const double                                        GetWeight(const int node, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const
            {
                return f(verts[node]);
            };
            const int                                           IncreaseOrder() { return 1; };
            //const int                                         SetValue(const int, CSolution* value);
            //const int                                         SetValue(const int, const CFESolution& value);
            //const CFESolution                                 GetValue(const Point&) const;
            const double                                        GetMeasure() const { return 0.; };
            //const CFESolution                                 GetValue(const int) const;
            //const std::function<const DForm<0>*(const Point&)>        GetShapeFunction(const int) const;
        private:
            static const int                                    m_number = 3;
            Point                                              m_p0, m_p1;
            Point                                              m_normal;
            double                                              m_mes;
            void                                                CompNormal();
            void                                                CompLenght();
            //std::vector<CFESolution>                          m_w;
            //const std::function<const double(const Point&p)>  m_psi[2];
        };
    }
}

#endif /* Edge_hpp */
```

7.22 CoreNCFEM/FiniteElements/FiniteElement.h File Reference

```
#include <functional>
#include <iostream>
```

```
#include <vector>
#include "../Point.h"
#include "../FESolution.h"
```

Classes

- class corenc::Mesh::CElement< bool >
- class corenc::Mesh::CElement< T >
- class corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, T >
- class corenc::Mesh::CFiniteElement< Shape, ShapeFunction, DoF, bool >
- class corenc::Mesh::CFiniteElement< Shape, ShapeFunction, bool, bool >

Namespaces

- namespace corenc
- namespace corenc::Mesh

Typedefs

- using corenc::Mesh::function_dp = std::function< const double(const Point &)>

Enumerations

- enum corenc::Mesh::Elements {
  corenc::Mesh::Interval = 0 , corenc::Mesh::Triangle = 1 , corenc::Mesh::Rectangle = 2 , corenc::Mesh::Tetrahedron = 3 ,
  corenc::Mesh::Cube = 4 }

7.23 FiniteElement.h

Go to the documentation of this file.

```
#ifndef CORENC_MESH_FINITEELEMENT_H_
#define CORENC_MESH_FINITEELEMENT_H_

#include <functional>
#include <iostream>
#include <vector>
#include "../Point.h"
#include "../FESolution.h"
namespace corenc
{
    namespace Mesh
    {
        using function_dp = std::function<const double(const Point&)>;
        enum Elements
        {
            Interval = 0,
            Triangle = 1,
            Rectangle = 2,
            Tetrahedron = 3,
            Cube = 4
        };

        template<class T = bool>
        class CElement;

        template<>
        class CElement<bool>
```

```
    {
    public:
        CElement() {}
        virtual ~CElement() {}
        virtual const int                                   GetType() const = 0;
        virtual CElement<>*                                   Clone() const = 0;
        virtual const int                                   GetDoFs() const = 0;
        virtual const int                                   GetNode(const int) const = 0;
        virtual const int                                   GetNeighbour(const int) const = 0;
        virtual void                                        SetNeighbour(const int k, const int elem) = 0;
        virtual void                                        SetType(const int) = 0;
        virtual void                                        SetNode(const int, const int) = 0;
        virtual const int                                   GetNumberOfNodes() const = 0;
        //virtual void          SetShapeFunction(const unsigned int, const std::function<const DiffForm(const Point&)>&) = 0;
        //virtual const DiffForm* GetShapeFunction(const int, const Point&) const = 0;
        //virtual const double                              GetShapeFunction(const unsigned int, const std::vector<double>&) const = 0;
        virtual const double                                GetShapeFunction(const int, const Point&) const = 0;
        virtual const Point                                    GetGradShapeFunction(const int, const Point&) const = 0;
        //virtual const std::vector<double>                 GetGradShapeFunction(const unsigned int, const std::vector<double>&) const = 0;
        virtual const Point                                    GetNormal() const = 0;
        //virtual const std::vector<double>                 GetNormal() const = 0;
        virtual void                                        ReverseNormal() = 0;
        virtual const double                                Integrate(const function_dp&, const std::vector<Point>& v) const = 0;
        //virtual const double                              Integrate(const std::function<const double(const std::vector<double>&)>&, const std::vector<std::vector<double>>& v) const = 0;
        virtual const Point                                    Integrate(const std::function<const Point(const Point&)>&, const std::vector<Point>& v) const = 0;
        //virtual const std::vector<double>                 Integrate(const std::function<const std::vector<double>(const std::vector<double>&)>&, const std::vector<std::vector<double>>& v) const = 0;
        virtual const std::vector<double>                   Integrate(const std::function<const std::vector<double>(const Point&)>&, const std::vector<Point>&) const = 0;
        virtual const double                                GetWeight(const int, const std::vector<Point>& verts, const function_dp& f) const = 0;
        //virtual const Type                                    GetValue(const unsigned number) const = 0;
        //virtual const Type                                    GetValue(const Mesh::Point&) const = 0;
        //virtual const int                                 SetValue(const unsigned int number, const Type& value) = 0;
        //virtual const int                                 SetValue(const int number, CSolution* value) = 0;
        //virtual CSolution*                                    GetValue(const int) = 0;
        virtual const int                                   IncreaseOrder() = 0;
        virtual const double                                GetMeasure() const = 0;
        //virtual const std::vector<double>                 Integrate(const std::function<const std::vector<double>(const std::vector<double>&)>&, const std::vector<std::vector<double>>&) const = 0;
        //virtual std::function<const DiffForm(const Point&)> GetShapeFunction(const int) = 0;
        //virtual const double  GetShapeFunction(const int, const Point&) const = 0;
    };
    template<class T>
    class CElement
    {
    public:
        CElement() {}
        virtual ~CElement() {}
        virtual const int                                   GetType() const = 0;
        virtual CElement*                                   Clone() const = 0;
        virtual const int                                   GetDoFs() const = 0;
        virtual const int                                   GetNode(const int) const = 0;
        virtual const int                                   GetNeighbour(const int) const = 0;
        virtual void                                        SetNeighbour(const int k, const int elem) = 0;
        virtual void                                        SetType(const int) = 0;
        virtual void                                        SetNode(const int, const int) = 0;
        virtual const int                                   GetNumberOfNodes() const = 0;
        //virtual void          SetShapeFunction(const unsigned int, const std::function<const DiffForm(const Point&)>&) = 0;
        //virtual const DiffForm* GetShapeFunction(const int, const Point&) const = 0;
        //virtual const double                              GetShapeFunction(const unsigned int, const std::vector<double>&) const = 0;
        virtual const double                                GetShapeFunction(const int, const Point&) const = 0;
        virtual const Point                                    GetGradShapeFunction(const int, const Point&) const = 0;
        //virtual const std::vector<double>                 GetGradShapeFunction(const unsigned int, const std::vector<double>&) const = 0;
        virtual const Point                                    GetNormal() const = 0;
        //virtual const std::vector<double>                 GetNormal() const = 0;
        virtual void                                        ReverseNormal() = 0;
```

```
        virtual const int                                   IncreaseOrder() = 0;
        virtual const double                                Integrate(const std::function<const double(const Point&)>&, const std::vector<Point>& v) const = 0;
        //virtual const double                              Integrate(const std::function<const double(const std::vector<double>&)>&, const std::vector<std::vector<double>>& v) const = 0;
        virtual const Point                                    Integrate(const std::function<const Point(const Point&)>&, const std::vector<Point>& v) const = 0;
        //virtual const std::vector<double>                 Integrate(const std::function<const std::vector<double>(const std::vector<double>&)>&, const std::vector<std::vector<double>>& v) const = 0;
        virtual const std::vector<double>                   Integrate(const std::function<const std::vector<double>(const Point&)>&, const std::vector<Point>&) const = 0;
        //virtual const Type                                    GetValue(const unsigned number) const = 0;
        //virtual const Type                                    GetValue(const Mesh::Point&) const = 0;
        //virtual const int                                 SetValue(const unsigned int number, const Type& value) = 0;
         //virtual const int                                 SetValue(const unsigned int number, CSolution* value) = 0;
         //virtual CSolution*                             GetValue(const unsigned int) = 0;
         virtual const double                                GetMeasure() const = 0;
         virtual const double                                GetWeight(const int, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const = 0;
         //virtual const T                                       GetValue(const int number) const = 0;
         //virtual const T                                       GetValue(const Point& p) const = 0;
         //virtual const int                                 SetValue(const int number, const T& value) = 0;
         //virtual const std::vector<double>                 Integrate(const std::function<const std::vector<double>(const std::vector<double>&)>&, const std::vector<std::vector<double>>&) const = 0;
         //virtual std::function<const DiffForm(const Point&)> GetShapeFunction(const int) = 0;
         //virtual const double  GetShapeFunction(const int, const Point&) const = 0;
     };




     // Set of nodes
     // Set of shape function
     // Set of degrees of freedom ; don't use pls
     // Type of the weights aligned with the degrees of freedom
     // The weights should be inside of the set of shape functions and the types should be same
     template<class Shape, class ShapeFunction, class DoF = bool, class T = bool>
     class CFiniteElement;
     template<class Shape, class ShapeFunction, class DoF, class T>
     class CFiniteElement: public CElement<T>
     {
     public:
         CFiniteElement() {}
         CFiniteElement(const int* nodes, const Point* points, const int dofs) :
             m_shape{ nodes },
             m_shapefunctions{ points },
             m_dofs{ dofs },
             m_type{ -1 } {
             m_neighbours[0] = -1; m_neighbours[1] = -1;
         };
         CFiniteElement(const int* nodes, const Point* points) :
             m_shape{ nodes },
             m_shapefunctions{ points },
             m_dofs{ 0 },
             m_type{ -1 } {
             m_neighbours[0] = -1; m_neighbours[1] = -1;
         };
         CFiniteElement(const Shape& shape, const ShapeFunction& f, const DoF& d) :
             m_shape{ shape },
             m_shapefunctions{ f },
             m_dofs{ d },
             m_type{ -1 } {
             m_neighbours[0] = -1; m_neighbours[1] = -1;
         };
         CFiniteElement(const Shape& shape, const ShapeFunction& shfunc, const DoF& dofs, const int type) :
             m_shape{ shape },
             m_shapefunctions{ shfunc },
             m_dofs{ dofs },
             m_type{ type } {
             m_neighbours[0] = -1; m_neighbours[1] = -1;
         };
                     CFiniteElement(const Shape& shape, const ShapeFunction& shfunc, const DoF& dofs, const int type, const int* neigs) :
                             m_shape{ shape },
                             m_shapefunctions{ shfunc },
                             m_dofs{ dofs },
                             m_type{ type } {
                             m_neighbours[0] = neigs[0]; m_neighbours[1] = neigs[1];
                     };
```

```
        CFiniteElement(const CFiniteElement<Shape, ShapeFunction, DoF>& e) :
            m_shape{ e.m_shape },
            m_shapefunctions{ e.m_shapefunctions },
            m_dofs{ e.m_dofs },
            m_type{ e.m_type } {
            m_neighbours[0] = e.m_neighbours[0]; m_neighbours[1] = e.m_neighbours[1];
        };
        CElement<T>*         Clone() const
        {
                                return new CFiniteElement<Shape, ShapeFunction, DoF, T>(m_shape, m_shapefunctions, m_dofs, m_type, m_neighbours);
        };
        friend const bool       operator==(const CFiniteElement& e1, const CFiniteElement& e2)
        {
            if (e1.m_shape == e2.m_shape)
                return true;
            return false;
        }
        ~CFiniteElement() {}
        const int               GetType() const;
        const int               GetNode(const int) const;
        const int               GetNeighbour(const int) const;
        const Shape             GetShape() const;
        const ShapeFunction     GetShapeFunctions() const;
        const DoF               GetDoF() const;
        const int               GetDoFs() const;
        void                    SetNeighbour(const int k, const int elem);
        void                    SetType(const int);
        void                    SetShapeFunction(const int, const ShapeFunction&);
        void                    SetDoF(const DoF&);
        void                    SetShape(const Shape&);
        const int               IncreaseOrder();
        void                    SetNode(const int, const int);
        const int               GetNumberOfNodes() const;
        //const int             SetValue(const int number, CSolution* value);
        const double            GetMeasure() const;
        //CSolution*                GetValue(const int);
        //void                    SetShapeFunction(const int, const std::function<const DiffForm(const Point&)>&);
        const double            Integrate(const std::function<const double(const Point&)>&, const std::vector<Point>& v) const;
        const Point                Integrate(const std::function<const Point(const Point&)>&, const std::vector<Point>& v) const;
        const std::vector<double>   Integrate(const std::function<const std::vector<double>(const Point&)>&, const std::vector<Point>&) const;
        //const std::function<const DiffForm(const Point&)>          GetShapeFunction(const int) const;
        //const DiffForm*           GetShapeFunction(const int, const Point&);
        const double            GetShapeFunction(const int, const Point&) const;
        const Point                GetGradShapeFunction(const int, const Point&) const;
        const Point                GetNormal() const;
        void                    ReverseNormal();
        const double            GetWeight(const int, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const;
        //const T                   GetValue(const int number) const;
        //const T                   GetValue(const Point& p) const;
        //const int             SetValue(const int number, const T& value);
        CFiniteElement&         operator=(const CFiniteElement& e)
        {
            m_shape = e.m_shape;
            m_shapefunctions = e.m_shapefunctions;
            m_dofs = e.m_dofs;
            m_type = e.m_type;
            return *this;
        }
        friend std::istream&    operator»(std::istream& is, CFiniteElement& k)
        {
            is » k.m_shape;
            return is;
        }
        //const DiffForm          GetDShapeFunction(const int, const Point&);
    private:
        Shape                   m_shape;
        ShapeFunction           m_shapefunctions;
        DoF                     m_dofs;
        int                     m_type;
        int                     m_neighbours[2];
    };

    template<class Shape, class ShapeFunction, class DoF>
    class CFiniteElement<Shape, ShapeFunction, DoF, bool> : public CElement<>
    {
    public:
        CFiniteElement() {}
        CFiniteElement(const int* nodes, const Point* points, const int dofs) :
            m_shape{ nodes },
            m_shapefunctions{ points },
```

```
            m_dofs{ dofs },
            m_type{ -1 } {
            m_neighbours[0] = -1; m_neighbours[1] = -1;
        };
        CFiniteElement(const int* nodes, const Point* points) :
            m_shape{ nodes },
            m_shapefunctions{ points },
            m_dofs{ 0 },
            m_type{ -1 } {
            m_neighbours[0] = -1; m_neighbours[1] = -1;
        };
        CFiniteElement(const Shape& shape, const ShapeFunction& f, const DoF& d) :
            m_shape{ shape },
            m_shapefunctions{ f },
            m_dofs{ d },
            m_type{ -1 } {
            m_neighbours[0] = -1; m_neighbours[1] = -1;
        };
        CFiniteElement(const Shape& shape, const ShapeFunction& shfunc, const DoF& dofs, const int type) :
            m_shape{ shape },
            m_shapefunctions{ shfunc },
            m_dofs{ dofs },
            m_type{ type } {
            m_neighbours[0] = -1; m_neighbours[1] = -1;
        };
        CFiniteElement(const Shape& shape, const ShapeFunction& shfunc, const DoF& dofs, const int type, const int* neigh) :
            m_shape{ shape },
            m_shapefunctions{ shfunc },
            m_dofs{ dofs },
            m_type{ type } {
            m_neighbours[0] = neigh[0]; m_neighbours[1] = neigh[1];
        };
        CFiniteElement(const CFiniteElement<Shape, ShapeFunction, DoF>& e) :
            m_shape{ e.m_shape },
            m_shapefunctions{ e.m_shapefunctions },
            m_dofs{ e.m_dofs },
            m_type{ e.m_type } {
            m_neighbours[0] = e.m_neighbours[0]; m_neighbours[1] = e.m_neighbours[1];
        };
        friend const bool       operator==(const CFiniteElement& e1, const CFiniteElement& e2)
        {
            if (e1.m_shape == e2.m_shape)
                return true;
            return false;
        }
        // don't forget to delete after the call
        CElement<>*             Clone() const
        {
            return new CFiniteElement<Shape, ShapeFunction, DoF>(m_shape, m_shapefunctions, m_dofs, m_type, m_neighbours);
        };
        ~CFiniteElement() {}
        const int               GetType() const;
        const int               GetNode(const int) const;
        const int               GetNeighbour(const int) const;
        const Shape             GetShape() const;
        const ShapeFunction     GetShapeFunctions() const;
        const DoF               GetDoF() const;
        const int               GetDoFs() const;
        void                    SetNeighbour(const int k, const int elem);
        void                    SetType(const int);
        void                    SetShapeFunction(const int, const ShapeFunction&);
        void                    SetDoF(const DoF&);
        void                    SetShape(const Shape&);
        void                    SetNode(const int, const int);
        const int               GetNumberOfNodes() const;
        //const int             SetValue(const int number, CSolution* value);
        const int               IncreaseOrder();
        const double            GetMeasure() const;
        //CSolution*                GetValue(const int);
        //void                    SetShapeFunction(const int, const std::function<const DiffForm(const Point&)>&);
        const double            Integrate(const std::function<const double(const Point&)>&, const std::vector<Point>& v) const;
        const Point                Integrate(const std::function<const Point(const Point&)>&, const std::vector<Point>& v) const;
        const std::vector<double>   Integrate(const std::function<const std::vector<double>(const Point&)>&, const std::vector<Point>&) const;
        //const std::function<const DiffForm(const Point&)>           GetShapeFunction(const int) const;
        //const DiffForm*            GetShapeFunction(const int, const Point&);
        const double            GetShapeFunction(const int, const Point&) const;
        const Point                GetGradShapeFunction(const int, const Point&) const;
        const Point                GetNormal() const;
        void                    ReverseNormal();
```

```
        const double            GetWeight(const int, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const;
        CFiniteElement&         operator=(const CFiniteElement& e)
        {
            m_shape = e.m_shape;
            m_shapefunctions = e.m_shapefunctions;
            m_dofs = e.m_dofs;
            m_type = e.m_type;
            return *this;
        }
        friend std::istream&    operator»(std::istream& is, CFiniteElement& k)
        {
            is » k.m_shape;
            return is;
        }
        //const DiffForm          GetDShapeFunction(const int, const Point&);
    private:
        Shape                   m_shape;
        ShapeFunction           m_shapefunctions;
        DoF                     m_dofs;
        int                     m_type;
        int                     m_neighbours[2];
    };


    template<class Shape, class ShapeFunction>
    class CFiniteElement<Shape, ShapeFunction, bool, bool> : public CElement<>
    {
    public:
        CFiniteElement() {}
        CFiniteElement(const int* nodes, const Point* points, const int dofs) :
            m_shape{ nodes },
            m_shapefunctions{ points, dofs },
            m_type{ -1 } {
            m_neighbours[0] = -1; m_neighbours[1] = -1;
        };
        CFiniteElement(const int* nodes, const Point* points, const int dofs, const int type) :
            m_shape{ nodes, dofs },
            m_shapefunctions{ points, dofs },
            m_type{ type } {
            m_neighbours[0] = -1; m_neighbours[1] = -1;
        };
        CFiniteElement(const int* nodes, const Point* points) :
            m_shape{ nodes },
            m_shapefunctions{ points },
            m_type{ -1 } {
            m_neighbours[0] = -1; m_neighbours[1] = -1;
        };
        CFiniteElement(const Shape& shape, const ShapeFunction& f) :
            m_shape{ shape },
            m_shapefunctions{ f },
            m_type{ -1 } {
            m_neighbours[0] = -1; m_neighbours[1] = -1;
        };
        CFiniteElement(const Shape& shape, const ShapeFunction& shfunc, const int type) :
            m_shape{ shape },
            m_shapefunctions{ shfunc },
            m_type{ type } {
            m_neighbours[0] = -1; m_neighbours[1] = -1;
        };
                    CFiniteElement(const Shape& shape, const ShapeFunction& shfunc, const int type, const int* neigs) :
                            m_shape{ shape },
                            m_shapefunctions{ shfunc },
                            m_type{ type } {
                            m_neighbours[0] = neigs[0]; m_neighbours[1] = neigs[1];
                    };
        CFiniteElement(const CFiniteElement<Shape, ShapeFunction>&e) :
            m_shape{ e.m_shape },
            m_shapefunctions{ e.m_shapefunctions },
            m_type{ e.m_type } {
            m_neighbours[0] = e.m_neighbours[0]; m_neighbours[1] = e.m_neighbours[1];
        };
        friend const bool       operator==(const CFiniteElement& e1, const CFiniteElement& e2)
        {
            if (e1.m_shape == e2.m_shape)
                return true;
            return false;
        }
        // don't forget to delete after the call
        CElement<>*             Clone() const
        {
                            return new CFiniteElement<Shape, ShapeFunction>(m_shape, m_shapefunctions, m_type, m_neighbours);
        };
        ~CFiniteElement() {}
        const int               GetType() const;
```

```
        const int               GetNode(const int) const;
        const int               GetNeighbour(const int) const;
        const Shape             GetShape() const;
        const ShapeFunction     GetShapeFunctions() const;
        const int               GetDoFs() const;
        void                    SetNeighbour(const int k, const int elem);
        void                    SetType(const int);
        void                    SetShapeFunction(const int, const ShapeFunction&);
        void                    SetShape(const Shape&);
        void                    SetNode(const int, const int);
        const int               GetNumberOfNodes() const;
        //const int             SetValue(const int number, CSolution* value);
        const int               IncreaseOrder();
        const double            GetMeasure() const;
        //CSolution*                GetValue(const int);
        //void                    SetShapeFunction(const int, const std::function<const DiffForm(const Point&)>&);
        const double            Integrate(const std::function<const double(const Point&)>&, const std::vector<Point>& v) const;
        const Point                Integrate(const std::function<const Point(const Point&)>&, const std::vector<Point>& v) const;
        const std::vector<double>   Integrate(const std::function<const std::vector<double>(const Point&)>&, const std::vector<Point>&) const;
        //const std::function<const DiffForm(const Point&)>          GetShapeFunction(const int) const;
        //const DiffForm*             GetShapeFunction(const int, const Point&);
        const double            GetShapeFunction(const int, const Point&) const;
        const Point                GetGradShapeFunction(const int, const Point&) const;
        const Point                GetNormal() const;
        void                    ReverseNormal();
        const double            GetWeight(const int, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const;
        CFiniteElement&         operator=(const CFiniteElement& e)
        {
            m_shape = e.m_shape;
            m_shapefunctions = e.m_shapefunctions;
            m_type = e.m_type;
            return *this;
        }
        friend std::istream&    operator»(std::istream& is, CFiniteElement& k)
        {
            is » k.m_shape;
            return is;
        }
        //const DiffForm          GetDShapeFunction(const int, const Point&);
    private:
        Shape                   m_shape;
        ShapeFunction           m_shapefunctions;
        int                     m_type;
        int                     m_neighbours[2];
    };





    // implementation template<class Shape, class ShapeFunction, class DoF>
    // CFiniteElement<Shape, ShapeFunction, DoF>
    template<class Shape, class ShapeFunction, class DoF>
    const int CFiniteElement<Shape, ShapeFunction, DoF, bool>::GetType() const
    {
        return m_type;
    }
    //template<class Shape, class ShapeFunction, class DoF>
    //const int CFiniteElement<Shape, ShapeFunction, DoF, bool>::SetValue(const int number, CSolution* value)
    //{
    //  return m_shapefunctions.SetValue(number, value);
    //}
    //template<class Shape, class ShapeFunction, class DoF>
    //CSolution* CFiniteElement<Shape, ShapeFunction, DoF, bool>::GetValue(const int p)
    //{
    //  auto&& val = m_shapefunctions.GetValue(p);
    //  return const_cast<CSolution*>(static_cast<const CSolution*>(&val));;
    //}
    template<class Shape, class ShapeFunction, class DoF>
    const int CFiniteElement<Shape, ShapeFunction, DoF, bool>::GetNode(const int k) const
    {
        return m_shape.GetNode(k);
    }
    template<class Shape, class ShapeFunction, class DoF>
    const double CFiniteElement<Shape, ShapeFunction, DoF, bool>::Integrate(const std::function<const double(const Point&)>&f, const std::vector<Point>& v) const
    {
        return m_shape.Integrate(f, v);
    }

```

```
        template<class Shape, class ShapeFunction, class DoF>
        const Point CFiniteElement<Shape, ShapeFunction, DoF, bool>::Integrate(const std::function<const
    Point(const Point&)>&f, const std::vector<Point>& v) const
        {
            return m_shape.Integrate(f, v);
        }

        template<class Shape, class ShapeFunction, class DoF>
        const std::vector<double> CFiniteElement<Shape, ShapeFunction, DoF, bool>::Integrate(const
    std::function<const std::vector<double>(const Point&)>&f, const std::vector<Point>& v) const
        {
            return m_shape.Integrate(f, v);
        }
        template<class Shape, class ShapeFunction, class DoF>
        const int CFiniteElement<Shape, ShapeFunction, DoF, bool>::GetNeighbour(const int k) const
        {
            return m_neighbours[k];
        }

        template<class Shape, class ShapeFunction, class DoF>
        const Point CFiniteElement<Shape, ShapeFunction, DoF, bool>::GetNormal() const
        {
            return m_shapefunctions.GetNormal();
        }

        template<class Shape, class ShapeFunction, class DoF>
        void CFiniteElement<Shape, ShapeFunction, DoF, bool>::ReverseNormal()
        {
            m_shapefunctions.ReverseNormal();
        }

        template<class Shape, class ShapeFunction, class DoF>
        const double CFiniteElement<Shape, ShapeFunction, DoF, bool>::GetMeasure() const
        {
            return m_shapefunctions.GetMeasure();
        }

        //template<class Shape, class ShapeFunction, class DoF, class T>
        //inline const T CFiniteElement<Shape, ShapeFunction, DoF, T>::GetValue(const int number) const
        //{
        //  return m_shapefunctions.GetValue(number);
        //}

        //template<class Shape, class ShapeFunction, class DoF, class T>
        //inline const T CFiniteElement<Shape, ShapeFunction, DoF, T>::GetValue(const Point & p) const
        //{
        //  return m_shapefunctions.GetValue(p);
        //}

        //template<class Shape, class ShapeFunction, class DoF, class T>
        //inline const int CFiniteElement<Shape, ShapeFunction, DoF, T>::SetValue(const int number,
    const T & value)
        //{
        //  return m_shapefunctions.SetValue(number, value);
        //}

        template<class Shape, class ShapeFunction, class DoF>
        const Shape CFiniteElement<Shape, ShapeFunction, DoF, bool>::GetShape() const
        {
            return m_shape;
        }

        template<class Shape, class ShapeFunction, class DoF>
        const ShapeFunction CFiniteElement<Shape, ShapeFunction, DoF, bool>::GetShapeFunctions() const
        {
            return m_shapefunctions;
        }

        template<class Shape, class ShapeFunction, class DoF>
        const DoF CFiniteElement<Shape, ShapeFunction, DoF, bool>::GetDoF() const
        {
            return m_dofs;
        }

        template<class Shape, class ShapeFunction, class DoF>
        const int CFiniteElement<Shape, ShapeFunction, DoF, bool>::GetNumberOfNodes() const
        {
            return m_shape.GetNumberOfNodes();
        }

        template<class Shape, class ShapeFunction, class DoF>
        const int CFiniteElement<Shape, ShapeFunction, DoF, bool>::GetDoFs() const
        {
            return m_shapefunctions.GetNumberOfShapeFunctions();
        }

        template<class Shape, class ShapeFunction, class DoF>
```

```
    void CFiniteElement<Shape, ShapeFunction, DoF, bool>::SetNeighbour(const int k, const int elem)
    {
        m_neighbours[k] = elem;
    }

    template<class Shape, class ShapeFunction, class DoF>
    void CFiniteElement<Shape, ShapeFunction, DoF, bool>::SetShapeFunction(const int k, const ShapeFunction& func)
    {
        m_shapefunctions = func;
    }

    template<class Shape, class ShapeFunction, class DoF>
    void CFiniteElement<Shape, ShapeFunction, DoF, bool>::SetDoF(const DoF& dof)
    {
        m_dofs = dof;
    }

    template<class Shape, class ShapeFunction, class DoF>
    void CFiniteElement<Shape, ShapeFunction, DoF, bool>::SetShape(const Shape &shape)
    {
        m_shape = shape;
    }

    template<class Shape, class ShapeFunction, class DoF>
    void CFiniteElement<Shape, ShapeFunction, DoF, bool>::SetType(const int k)
    {
        m_type = k;
    }

    template<class Shape, class ShapeFunction, class DoF>
    void CFiniteElement<Shape, ShapeFunction, DoF, bool>::SetNode(const int k, const int node)
    {
        m_shape.SetNode(k, node);
    }
    template<class Shape, class ShapeFunction, class DoF>
    const double CFiniteElement<Shape, ShapeFunction, DoF>::GetShapeFunction(const int k, const Mesh::Point &p) const
    {
        return m_shapefunctions.GetShapeFunction(k, p);
    }

    template<class Shape, class ShapeFunction, class DoF>
    const Point CFiniteElement<Shape, ShapeFunction, DoF, bool>::GetGradShapeFunction(const int k, const Mesh::Point &p) const
    {
        return m_shapefunctions.GetGradShapeFunction(k, p);
    }

    template<class Shape, class ShapeFunction, class DoF>
    const int CFiniteElement<Shape, ShapeFunction, DoF, bool>::IncreaseOrder()
    {
        if (m_shape.IncreaseOrder())
            return 1;
        if (m_shapefunctions.IncreaseOrder())
            return 1;
        return 0;
    }
    template<class Shape, class ShapeFunction, class DoF>
    const double CFiniteElement<Shape, ShapeFunction, DoF>::GetWeight(const int, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const
    {
        return 0.0;
    }
    // fin.

    // implementation template<class Shape, class ShapeFunction, class DoF, class T>
    // CFiniteElement<Shape, ShapeFunction, DoF, T>
    template<class Shape, class ShapeFunction, class DoF, class T>
    const int CFiniteElement<Shape, ShapeFunction, DoF, T>::GetType() const
    {
        return m_type;
    }
    template<class Shape, class ShapeFunction, class DoF, class T>
    const double CFiniteElement<Shape, ShapeFunction, DoF, T>::GetMeasure() const
    {
        return m_shapefunctions.GetMeasure();
    }
    //template<class Shape, class ShapeFunction, class DoF, class T>
    //const int CFiniteElement<Shape, ShapeFunction, DoF, T>::SetValue(const int number, CSolution* value)
    //{
    //  return m_shapefunctions.SetValue(number, value);
    //}
    //template<class Shape, class ShapeFunction, class DoF, class T>
    //CSolution* CFiniteElement<Shape, ShapeFunction, DoF, T>::GetValue(const int p)
    //{
```

```
        //  auto&& val = m_shapefunctions.GetValue(p);
        //  return const_cast<CSolution*>(static_cast<const CSolution*>(&val));
        //}
        template<class Shape, class ShapeFunction, class DoF, class T>
        const int CFiniteElement<Shape, ShapeFunction, DoF, T>::GetNode(const int k) const
        {
            return m_shape.GetNode(k);
        }
        template<class Shape, class ShapeFunction, class DoF, class T>
        const double CFiniteElement<Shape, ShapeFunction, DoF, T>::Integrate(const std::function<const
    double(const Point&)>&f, const std::vector<Point>& v) const
        {
            return m_shape.Integrate(f, v);
        }

        template<class Shape, class ShapeFunction, class DoF, class T>
        const Point CFiniteElement<Shape, ShapeFunction, DoF, T>::Integrate(const std::function<const
    Point(const Point&)>&f, const std::vector<Point>& v) const
        {
            return m_shape.Integrate(f, v);
        }

        template<class Shape, class ShapeFunction, class DoF, class T>
        const std::vector<double> CFiniteElement<Shape, ShapeFunction, DoF, T>::Integrate(const
    std::function<const std::vector<double>(const Point&)>&f, const std::vector<Point>& v) const
        {
            return m_shape.Integrate(f, v);
        }
        template<class Shape, class ShapeFunction, class DoF, class T>
        const int CFiniteElement<Shape, ShapeFunction, DoF, T>::GetNeighbour(const int k) const
        {
            return m_neighbours[k];
        }

        template<class Shape, class ShapeFunction, class DoF, class T>
        const Point CFiniteElement<Shape, ShapeFunction, DoF, T>::GetNormal() const
        {
            return m_shapefunctions.GetNormal();
        }

        template<class Shape, class ShapeFunction, class DoF, class T>
        void CFiniteElement<Shape, ShapeFunction, DoF, T>::ReverseNormal()
        {
            m_shapefunctions.ReverseNormal();
        }

        template<class Shape, class ShapeFunction, class DoF, class T>
                const double CFiniteElement<Shape, ShapeFunction, DoF, T>::GetWeight(const int node,
    const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const
        {
                        return m_shapefunctions.GetWeight(node, verts, f);
        }


        template<class Shape, class ShapeFunction, class DoF, class T>
        const Shape CFiniteElement<Shape, ShapeFunction, DoF, T>::GetShape() const
        {
            return m_shape;
        }

        template<class Shape, class ShapeFunction, class DoF, class T>
        const ShapeFunction CFiniteElement<Shape, ShapeFunction, DoF, T>::GetShapeFunctions() const
        {
            return m_shapefunctions;
        }

        template<class Shape, class ShapeFunction, class DoF, class T>
        const DoF CFiniteElement<Shape, ShapeFunction, DoF, T>::GetDoF() const
        {
            return m_dofs;
        }

        template<class Shape, class ShapeFunction, class DoF, class T>
        const int CFiniteElement<Shape, ShapeFunction, DoF, T>::GetNumberOfNodes() const
        {
            return m_shape.GetNumberOfNodes();
        }

        template<class Shape, class ShapeFunction, class DoF, class T>
        const int CFiniteElement<Shape, ShapeFunction, DoF, T>::GetDoFs() const
        {
            return m_shapefunctions.GetNumberOfShapeFunctions();
        }

        template<class Shape, class ShapeFunction, class DoF, class T>
        void CFiniteElement<Shape, ShapeFunction, DoF, T>::SetNeighbour(const int k, const int elem)
        {
```

```
        m_neighbours[k] = elem;
    }

    template<class Shape, class ShapeFunction, class DoF, class T>
    void CFiniteElement<Shape, ShapeFunction, DoF, T>::SetShapeFunction(const int k, const
ShapeFunction& func)
    {
        m_shapefunctions = func;
    }

    template<class Shape, class ShapeFunction, class DoF, class T>
    void CFiniteElement<Shape, ShapeFunction, DoF, T>::SetDoF(const DoF& dof)
    {
        m_dofs = dof;
    }

    template<class Shape, class ShapeFunction, class DoF, class T>
    void CFiniteElement<Shape, ShapeFunction, DoF, T>::SetShape(const Shape &shape)
    {
        m_shape = shape;
    }

    template<class Shape, class ShapeFunction, class DoF, class T>
    void CFiniteElement<Shape, ShapeFunction, DoF, T>::SetType(const int k)
    {
        m_type = k;
    }

    template<class Shape, class ShapeFunction, class DoF, class T>
    void CFiniteElement<Shape, ShapeFunction, DoF, T>::SetNode(const int k, const int node)
    {
        m_shape.SetNode(k, node);
    }
    template<class Shape, class ShapeFunction, class DoF, class T>
    const double CFiniteElement<Shape, ShapeFunction, DoF, T>::GetShapeFunction(const int k, const
Mesh::Point &p) const
    {
        return m_shapefunctions.GetShapeFunction(k, p);
    }

    template<class Shape, class ShapeFunction, class DoF, class T>
    const Point CFiniteElement<Shape, ShapeFunction, DoF, T>::GetGradShapeFunction(const int k,
const Mesh::Point &p) const
    {
        return m_shapefunctions.GetGradShapeFunction(k, p);
    }
    template<class Shape, class ShapeFunction, class DoF, class T>
    const int CFiniteElement<Shape, ShapeFunction, DoF, T>::IncreaseOrder()
    {
        if (m_shape.IncreaseOrder())
            return 1;
        if (m_shapefunctions.IncreaseOrder())
            return 1;
        return 0;
    }

    // implementation template<class Shape, class ShapeFunction>
    // CFiniteElement<Shape, ShapeFunction>

    template<class Shape, class ShapeFunction>
    const int CFiniteElement<Shape, ShapeFunction, bool, bool>::GetType() const
    {
        return m_type;
    }
    template<class Shape, class ShapeFunction>
    const double CFiniteElement<Shape, ShapeFunction, bool, bool>::GetMeasure() const
    {
        return m_shapefunctions.GetMeasure();
    }
    //template<class Shape, class ShapeFunction>
    //const int CFiniteElement<Shape, ShapeFunction, bool, bool>::SetValue(const int number,
CSolution* value)
    //{
    //  return m_shapefunctions.SetValue(number, value);
    //}
    //template<class Shape, class ShapeFunction>
    //CSolution* CFiniteElement<Shape, ShapeFunction, bool, bool>::GetValue(const int p)
    //{
    //  auto&& val = m_shapefunctions.GetValue(p);
    //  return const_cast<CSolution*>(static_cast<const CSolution*>(&val));;
    //}
    template<class Shape, class ShapeFunction>
    const int CFiniteElement<Shape, ShapeFunction, bool, bool>::GetNode(const int k) const
    {
        return m_shape.GetNode(k);
    }
    template<class Shape, class ShapeFunction>
```

```
    const double CFiniteElement<Shape, ShapeFunction, bool, bool>::Integrate(const std::function<const double(const Point&)>&f, const std::vector<Point>& v) const
    {
        return m_shape.Integrate(f, v);
    }

    template<class Shape, class ShapeFunction>
    const Point CFiniteElement<Shape, ShapeFunction, bool, bool>::Integrate(const std::function<const Point(const Point&)>&f, const std::vector<Point>& v) const
    {
        return m_shape.Integrate(f, v);
    }

    template<class Shape, class ShapeFunction>
    const std::vector<double> CFiniteElement<Shape, ShapeFunction, bool, bool>::Integrate(const std::function<const std::vector<double>(const Point&)>&f, const std::vector<Point>& v) const
    {
        return m_shape.Integrate(f, v);
    }
    template<class Shape, class ShapeFunction>
    const int CFiniteElement<Shape, ShapeFunction, bool, bool>::GetNeighbour(const int k) const
    {
        return m_neighbours[k];
    }

    template<class Shape, class ShapeFunction>
    const Point CFiniteElement<Shape, ShapeFunction, bool, bool>::GetNormal() const
    {
        return m_shapefunctions.GetNormal();
    }

    template<class Shape, class ShapeFunction>
    void CFiniteElement<Shape, ShapeFunction, bool, bool>::ReverseNormal()
    {
        m_shapefunctions.ReverseNormal();
    }


    template<class Shape, class ShapeFunction>
    const Shape CFiniteElement<Shape, ShapeFunction, bool, bool>::GetShape() const
    {
        return m_shape;
    }

    template<class Shape, class ShapeFunction>
    const ShapeFunction CFiniteElement<Shape, ShapeFunction, bool, bool>::GetShapeFunctions() const
    {
        return m_shapefunctions;
    }


    template<class Shape, class ShapeFunction>
    const int CFiniteElement<Shape, ShapeFunction, bool, bool>::GetNumberOfNodes() const
    {
        return m_shape.GetNumberOfNodes();
    }

    template<class Shape, class ShapeFunction>
    const int CFiniteElement<Shape, ShapeFunction, bool, bool>::GetDoFs() const
    {
        return m_shapefunctions.GetNumberOfShapeFunctions();
    }

    template<class Shape, class ShapeFunction>
    void CFiniteElement<Shape, ShapeFunction, bool, bool>::SetNeighbour(const int k, const int elem)
    {
        m_neighbours[k] = elem;
    }

    template<class Shape, class ShapeFunction>
    void CFiniteElement<Shape, ShapeFunction, bool, bool>::SetShapeFunction(const int k, const ShapeFunction& func)
    {
        m_shapefunctions = func;
    }

    template<class Shape, class ShapeFunction>
    void CFiniteElement<Shape, ShapeFunction, bool, bool>::SetShape(const Shape &shape)
    {
        m_shape = shape;
    }

    template<class Shape, class ShapeFunction>
    void CFiniteElement<Shape, ShapeFunction, bool, bool>::SetType(const int k)
    {
        m_type = k;
    }
```

```
        template<class Shape, class ShapeFunction>
        void CFiniteElement<Shape, ShapeFunction, bool, bool>::SetNode(const int k, const int node)
        {
            m_shape.SetNode(k, node);
        }
        template<class Shape, class ShapeFunction>
        const double CFiniteElement<Shape, ShapeFunction, bool, bool>::GetShapeFunction(const int k, const Mesh::Point &p) const
        {
            return m_shapefunctions.GetShapeFunction(k, p);
        }

        template<class Shape, class ShapeFunction>
        const Point CFiniteElement<Shape, ShapeFunction, bool, bool>::GetGradShapeFunction(const int k, const Mesh::Point &p) const
        {
            return m_shapefunctions.GetGradShapeFunction(k, p);
        }

        template<class Shape, class ShapeFunction>
        const int CFiniteElement<Shape, ShapeFunction, bool, bool>::IncreaseOrder()
        {
            if(m_shape.IncreaseOrder())
                return 1;
            if (m_shapefunctions.IncreaseOrder())
                return 1;
            return 0;
        }
        template<class Shape, class ShapeFunction>
        const double CFiniteElement<Shape, ShapeFunction, bool, bool>::GetWeight(const int node, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const
        {
            return m_shapefunctions.GetWeight(node, verts, f);
        }
        // fin.

    }

}


#endif /* CORENC_MESH_FINITEELEMENT_H_ */
```

7.24 CoreNCFEM/FiniteElements/FiniteElement2D.h File Reference

```
#include <functional>
#include <iostream>
#include <vector>
#include "../Point.h"
#include "../FESolution.h"
```

Classes

- class corenc::Mesh::CElement2D< bool >
- class corenc::Mesh::CElement2D< T >
- class corenc::Mesh::CFiniteElement2D< Shape, ShapeFunction >

Namespaces

- namespace corenc
- namespace corenc::Mesh

7.25 FiniteElement2D.h

Go to the documentation of this file.

```
#ifndef FINITEELEMENT2D_H
#define FINITEELEMENT2D_H
#include <functional>
#include <iostream>
#include <vector>
#include "../Point.h"
#include "../FESolution.h"
namespace corenc
{
    namespace Mesh
    {
    // 2d element

        template<class T = bool>
        class CElement2D;
        using function_dp = std::function<const double(const Point&)>;
        template<>
        class CElement2D<bool>
        {
        public:
            CElement2D() {}
            virtual ~CElement2D() {}
            virtual const int                    GetType() const = 0;
            virtual CElement2D<>*                Clone() const = 0;
            virtual const int                    GetDoFs() const = 0;
            virtual const int                    GetNode(const int) const = 0;
            virtual const int                    GetNeighbour(const int) const = 0;
            virtual void                         SetNeighbour(const int k, const int elem) = 0;
            virtual void                         SetType(const int) = 0;
            virtual void                         SetNode(const int, const int) = 0;
            virtual const int                    GetNumberOfNodes() const = 0;
            virtual const double                 GetShapeFunction(const int, const Point&) const = 0;
            virtual const Point                  GetGradShapeFunction(const int, const Point&) const = 0;
            virtual const Point                  GetNormal() const = 0;
            virtual void                         ReverseNormal() = 0;
            virtual const int                    SetOrder(const int px, const int py) = 0;
            virtual const double                 Integrate(const function_dp&, const std::vector<Point>& v) const = 0;
            virtual const Point                  Integrate(const std::function<const Point(const Point&)>&, const std::vector<Point>& v) const = 0;
            virtual const std::vector<double>    Integrate(const std::function<const std::vector<double>(const Point&)>&, const std::vector<Point>&) const = 0;
            virtual const double                 GetWeight(const int, const std::vector<Point>& verts, const function_dp& f) const = 0;
            virtual const int                    IncreaseOrder() = 0;
            virtual const double                 GetMeasure() const = 0;
        };

        template<class T>
        class CElement2D
        {
        public:
            CElement2D() {}
            virtual ~CElement2D() {}
            virtual const int                    GetType() const = 0;
            virtual CElement2D*                  Clone() const = 0;
            virtual const int                    GetDoFs() const = 0;
            virtual const int                    GetNode(const int) const = 0;
            virtual const int                    GetNeighbour(const int) const = 0;
            virtual void                         SetNeighbour(const int k, const int elem) = 0;
            virtual void                         SetType(const int) = 0;
            virtual void                         SetNode(const int, const int) = 0;
            virtual const int                    GetNumberOfNodes() const = 0;
            virtual const double                 GetShapeFunction(const int, const Point&) const = 0;
            virtual const Point                  GetGradShapeFunction(const int, const Point&) const = 0;
            virtual const Point                  GetNormal() const = 0;
            virtual void                         ReverseNormal() = 0;
            virtual const int                    IncreaseOrder() = 0;
            virtual const int                    SetOrder(const int px, const int py) = 0;
            virtual const double                 Integrate(const std::function<const double(const Point&)>&, const std::vector<Point>& v) const = 0;
            virtual const Point                  Integrate(const std::function<const Point(const Point&)>&, const std::vector<Point>& v) const = 0;
            virtual const std::vector<double>    Integrate(const std::function<const std::vector<double>(const Point&)>&, const std::vector<Point>&) const = 0;
            virtual const double                 GetMeasure() const = 0;
```

```
        virtual const double                    GetWeight(const int, const
std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const = 0;
    };

    template<class Shape, class ShapeFunction>
    class CFiniteElement2D : public CElement2D<>
    {
    public:
        CFiniteElement2D() {}
        CFiniteElement2D(const int* nodes, const Point* points, const int dofs) :
            m_shape{ nodes },
            m_shapefunctions{ points, dofs },
            m_type{ -1 } {
            m_neighbours[0] = -1; m_neighbours[1] = -1;
        };
        CFiniteElement2D(const int* nodes, const Point* points, const int dofs, const int type) :
            m_shape{ nodes, dofs },
            m_shapefunctions{ points, dofs },
            m_type{ type } {
            m_neighbours[0] = -1; m_neighbours[1] = -1;
        };
        CFiniteElement2D(const int* nodes, const Point* points) :
            m_shape{ nodes },
            m_shapefunctions{ points },
            m_type{ -1 } {
            m_neighbours[0] = -1; m_neighbours[1] = -1;
        };
        CFiniteElement2D(const Shape& shape, const ShapeFunction& f) :
            m_shape{ shape },
            m_shapefunctions{ f },
            m_type{ -1 } {
             m_neighbours[0] = -1; m_neighbours[1] = -1;
         };
         CFiniteElement2D(const Shape& shape, const ShapeFunction& shfunc, const int type) :
             m_shape{ shape },
             m_shapefunctions{ shfunc },
             m_type{ type } {
             m_neighbours[0] = -1; m_neighbours[1] = -1;
         };
         CFiniteElement2D(const Shape& shape, const ShapeFunction& shfunc, const int type, const int*
neigs) :
                             m_shape{ shape },
                             m_shapefunctions{ shfunc },
                             m_type{ type } {
                             m_neighbours[0] = neigs[0]; m_neighbours[1] = neigs[1];
                     };
         CFiniteElement2D(const CFiniteElement2D&e) :
             m_shape{ e.m_shape },
             m_shapefunctions{ e.m_shapefunctions },
             m_type{ e.m_type } {
             m_neighbours[0] = e.m_neighbours[0]; m_neighbours[1] = e.m_neighbours[1];
         };
         friend const bool        operator==(const CFiniteElement2D& e1, const CFiniteElement2D& e2)
         {
             if (e1.m_shape == e2.m_shape)
                 return true;
             return false;
         }
         // don't forget to delete after the call
         CElement2D<>*            Clone() const
         {
                             return new CFiniteElement2D(m_shape, m_shapefunctions, m_type,
m_neighbours);
         };
         ~CFiniteElement2D() {}
         const int                GetType() const;
         const int                GetNode(const int) const;
         const int                GetNeighbour(const int) const;
         const Shape              GetShape() const;
         const ShapeFunction      GetShapeFunctions() const;
         const int                GetDoFs() const;
         void                     SetNeighbour(const int k, const int elem);
         void                     SetType(const int);
         void                     SetShapeFunction(const int, const ShapeFunction&);
         void                     SetShape(const Shape&);
         const int                SetOrder(const int px, const int py);
         void                     SetNode(const int, const int);
         const int                GetNumberOfNodes() const;
         //const int              SetValue(const int number, CSolution* value);
         const int                IncreaseOrder();
         const double             GetMeasure() const;
         //CSolution*                   GetValue(const int);
         //void                       SetShapeFunction(const int, const std::function<const
DiffForm(const Point&)>&);
         const double             Integrate(const std::function<const double(const Point&)>&, const
std::vector<Point>& v) const;
         const Point                 Integrate(const std::function<const Point(const Point&)>&, const
```

```
std::vector<Point>& v) const;
        const std::vector<double>   Integrate(const std::function<const std::vector<double>(const Point&)>&, const std::vector<Point>&) const;
        //const std::function<const DiffForm(const Point&)>          GetShapeFunction(const int) const;
        //const DiffForm*          GetShapeFunction(const int, const Point&);
        const double             GetShapeFunction(const int, const Point&) const;
        const Point                 GetGradShapeFunction(const int, const Point&) const;
        const Point                 GetNormal() const;
        void                     ReverseNormal();
        const double             GetWeight(const int, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const;
        CFiniteElement2D&          operator=(const CFiniteElement2D& e)
        {
            m_shape = e.m_shape;
            m_shapefunctions = e.m_shapefunctions;
            m_type = e.m_type;
            return *this;
        }
        friend std::istream&     operator>>(std::istream& is, CFiniteElement2D& k)
        {
            is >> k.m_shape;
            return is;
        }
        //const DiffForm          GetDShapeFunction(const int, const Point&);
    private:
        Shape                    m_shape;
        ShapeFunction            m_shapefunctions;
        int                      m_type;
        int                      m_neighbours[2];
    };


    template<class Shape, class ShapeFunction>
    const int CFiniteElement2D<Shape, ShapeFunction>::GetType() const
    {
        return m_type;
    }
    template<class Shape, class ShapeFunction>
    const double CFiniteElement2D<Shape, ShapeFunction>::GetMeasure() const
    {
        return m_shapefunctions.GetMeasure();
    }
    //template<class Shape, class ShapeFunction>
    //const int CFiniteElement2D<Shape, ShapeFunction>::SetValue(const int number, CSolution* value)
    //{
    //  return m_shapefunctions.SetValue(number, value);
    //}
    //template<class Shape, class ShapeFunction>
    //CSolution* CFiniteElement2D<Shape, ShapeFunction>::GetValue(const int p)
    //{
    //  auto&& val = m_shapefunctions.GetValue(p);
    //  return const_cast<CSolution*>(static_cast<const CSolution*>(&val));;
    //}
    template<class Shape, class ShapeFunction>
    const int CFiniteElement2D<Shape, ShapeFunction>::GetNode(const int k) const
    {
        return m_shape.GetNode(k);
    }
    template<class Shape, class ShapeFunction>
    const double CFiniteElement2D<Shape, ShapeFunction>::Integrate(const std::function<const double(const Point&)>&f, const std::vector<Point>& v) const
    {
        return m_shape.Integrate(f, v);
    }

    template<class Shape, class ShapeFunction>
    const Point CFiniteElement2D<Shape, ShapeFunction>::Integrate(const std::function<const Point(const Point&)>&f, const std::vector<Point>& v) const
    {
        return m_shape.Integrate(f, v);
    }

    template<class Shape, class ShapeFunction>
    const std::vector<double> CFiniteElement2D<Shape, ShapeFunction>::Integrate(const std::function<const std::vector<double>(const Point&)>&f, const std::vector<Point>& v) const
    {
        return m_shape.Integrate(f, v);
    }
    template<class Shape, class ShapeFunction>
    const int CFiniteElement2D<Shape, ShapeFunction>::GetNeighbour(const int k) const
    {
        return m_neighbours[k];
    }

    template<class Shape, class ShapeFunction>
    const Point CFiniteElement2D<Shape, ShapeFunction>::GetNormal() const
```

```
        {
            return m_shapefunctions.GetNormal();
        }

        template<class Shape, class ShapeFunction>
        void CFiniteElement2D<Shape, ShapeFunction>::ReverseNormal()
        {
            m_shapefunctions.ReverseNormal();
        }


        template<class Shape, class ShapeFunction>
        const Shape CFiniteElement2D<Shape, ShapeFunction>::GetShape() const
        {
            return m_shape;
        }

        template<class Shape, class ShapeFunction>
        const ShapeFunction CFiniteElement2D<Shape, ShapeFunction>::GetShapeFunctions() const
        {
            return m_shapefunctions;
        }


        template<class Shape, class ShapeFunction>
        const int CFiniteElement2D<Shape, ShapeFunction>::GetNumberOfNodes() const
        {
            return m_shape.GetNumberOfNodes();
        }

        template<class Shape, class ShapeFunction>
        const int CFiniteElement2D<Shape, ShapeFunction>::GetDoFs() const
        {
            return m_shapefunctions.GetNumberOfShapeFunctions();
        }

        template<class Shape, class ShapeFunction>
        void CFiniteElement2D<Shape, ShapeFunction>::SetNeighbour(const int k, const int elem)
        {
            m_neighbours[k] = elem;
        }

        template<class Shape, class ShapeFunction>
        void CFiniteElement2D<Shape, ShapeFunction>::SetShapeFunction(const int k, const ShapeFunction&
    func)
        {
            m_shapefunctions = func;
        }

        template<class Shape, class ShapeFunction>
        void CFiniteElement2D<Shape, ShapeFunction>::SetShape(const Shape &shape)
        {
            m_shape = shape;
        }

        template<class Shape, class ShapeFunction>
        void CFiniteElement2D<Shape, ShapeFunction>::SetType(const int k)
        {
            m_type = k;
        }

        template<class Shape, class ShapeFunction>
        void CFiniteElement2D<Shape, ShapeFunction>::SetNode(const int k, const int node)
        {
            m_shape.SetNode(k, node);
        }
        template<class Shape, class ShapeFunction>
        const double CFiniteElement2D<Shape, ShapeFunction>::GetShapeFunction(const int k, const
    Mesh::Point &p) const
        {
            return m_shapefunctions.GetShapeFunction(k, p);
        }

        template<class Shape, class ShapeFunction>
        const Point CFiniteElement2D<Shape, ShapeFunction>::GetGradShapeFunction(const int k, const
    Mesh::Point &p) const
        {
            return m_shapefunctions.GetGradShapeFunction(k, p);
        }

        template<class Shape, class ShapeFunction>
        const int CFiniteElement2D<Shape, ShapeFunction>::IncreaseOrder()
        {
            if(m_shape.IncreaseOrder())
                return 1;
            if (m_shapefunctions.IncreaseOrder())
                return 1;
```

```
            return 0;
        }

        template<class Shape, class ShapeFunction>
        const int CFiniteElement2D<Shape, ShapeFunction>::SetOrder(const int px, const int py)
        {
            if(m_shape.SetOrder(px, py))
                return 1;
            if (m_shapefunctions.SetOrder(px, py))
                return 1;
            return 0;
        }

        template<class Shape, class ShapeFunction>
        const double CFiniteElement2D<Shape, ShapeFunction>::GetWeight(const int node, const
      std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const
        {
            return m_shapefunctions.GetWeight(node, verts, f);
        }
        // fin.
    }
}

#endif // FINITEELEMENT2D_H
```

7.26 CoreNCFEM/FiniteElements/Node.cpp File Reference

```
#include "Node.h"
```

7.27 CoreNCFEM/FiniteElements/Node.h File Reference

```
#include <stdio.h>
#include "Shape.h"
#include "ShapeFunction.h"
#include <iostream>
```

Classes

- class corenc::Mesh::CNode
- class corenc::Mesh::CNodeBasis

Namespaces

- namespace corenc
- namespace corenc::Mesh

7.28 Node.h

Go to the documentation of this file.

```
#ifndef Node_hpp
#define Node_hpp

#include <stdio.h>
#include "Shape.h"
#include "ShapeFunction.h"
#include <iostream>

namespace corenc
{
    namespace Mesh
    {
        class CNode : public CShape
        {
        public:
            CNode();
            CNode(const CNode&);
            CNode(const int n);
            CNode(const int*n);
            CNode& operator=(const CNode& e)
            {
                m_node = e.m_node;
                return *this;
            }
            friend const bool operator==(const CNode& e1, const CNode& e2)
            {
                if (e1.m_node == e2.m_node)
                    return true;
                return false;
            }
            friend std::istream& operator»(std::istream& is, CNode& e)
            {
                is » e.m_node;
                --e.m_node;
                return is;
            }
            ~CNode() {};
            const int                                         GetNode(const int) const;
            const int                                         GetNode(const NODES&) const;
            const int                                         IncreaseOrder() { return 1; };
            const int                                         GetNumberOfNodes() const;
            void                                              SetNode(const int k, const int node);
            const double                                      Integrate(const std::function<const double(const Point&)>&, const std::vector<Point>& v) const;
            const Point                                          Integrate(const std::function<const Point(const Point&)>&, const std::vector<Point>& v) const;
            const std::vector<double>                         Integrate(const std::function<const std::vector<double>(const Point&)>&, const std::vector<Point>&) const;
        private:
            const int                                         m_number = 1;
            int                                               m_node;
        };

        class CNodeBasis : public CShapeFunction<double>
        {
        public:
            CNodeBasis();
            CNodeBasis(const Point*);
            CNodeBasis(const CNodeBasis&e)
            {
                m_p0 = e.m_p0;
                m_normal = e.m_normal;
            };
            CNodeBasis& operator=(const CNodeBasis& e)
            {
                m_p0 = e.m_p0;
                m_normal = e.m_normal;
                return *this;
            }
            ~CNodeBasis() {};
            const int                                         GetNumberOfShapeFunctions() const;
            //const DForm<0>*                                     GetShapeFunction(const int) const;
            const double                                      GetShapeFunction(const int, const Point&) const;
            const Point                                          GetGradShapeFunction(const int, const Point&) const;
            const Point                                          GetNormal() const;
            void                                              ReverseNormal();
            const double                                      GetWeight(const int node, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const
            {
```

```
                return f(verts[node]);
            };
            //const int                                              SetValue(const int, CSolution*
    value);
            //const int                                              SetValue(const int, const
    CFESolution& value);
            const int                                                IncreaseOrder() { return 1; };
            //const CFESolution                                      GetValue(const Point&) const;
            const double                                             GetMeasure() const { return 0.; };
            //const CFESolution                                      GetValue(const int) const;
            //const std::function<const DForm<0>*(const Point&)>            GetShapeFunction(const int)
    const;
        private:
            static const int                                         m_number = 1;
            Point                                                   m_p0;
            Point                                                   m_normal;
            //CFESolution                                                m_w;
            //const std::function<const double(const Point&p)>       m_psi[2];
        };
    }

}
#endif /* Node_hpp */
```

7.29 CoreNCFEM/FiniteElements/Rectangle.cpp File Reference

```
#include "Rectangle.h"
```

7.30 CoreNCFEM/FiniteElements/Rectangle.h File Reference

```
#include <stdio.h>
#include "Shape.h"
#include "ShapeFunction.h"
#include <iostream>
```

Classes

- class corenc::Mesh::CRectangle
- class corenc::Mesh::CRectangleBasis
- class corenc::Mesh::CRectangleHBasis
- class corenc::Mesh::CRectangleBasis2x
- class corenc::Mesh::CRectangleBasis2y
- class corenc::Mesh::CRectangleBasis2
- class corenc::Mesh::CRectangleConstantBasis

Namespaces

- namespace corenc
- namespace corenc::Mesh

7.31 Rectangle.h

Go to the documentation of this file.

```
#ifndef CORENC_MESH_RECTANGLE_H_
#define CORENC_MESH_RECTANGLE_H_

#include <stdio.h>
#include "Shape.h"
#include "ShapeFunction.h"
#include <iostream>
namespace corenc
{
    namespace Mesh
    {
        class CRectangle : public CShape
        {
        public:
            CRectangle();
            CRectangle(const int n1, const int n2, const int n3, const int n4, const int order);
            CRectangle(const int n1, const int n2, const int n3, const int n4, const int e1, const int e2, const int e3, const int e4, const int order);
            CRectangle(const int*, const int order);
            CRectangle(const int*, const int*, const int order);
            CRectangle(const CRectangle&);
            CRectangle& operator=(const CRectangle& t)
            {
                m_nodes = t.m_nodes;
                m_edges[0] = t.m_edges[0];
                m_edges[1] = t.m_edges[1];
                m_edges[2] = t.m_edges[2];
                m_edges[3] = t.m_edges[3];
                m_number = t.m_number;
                m_order = t.m_order;
                m_px = t.m_px;
                m_py = t.m_py;
                return *this;
            }
            const bool    operator==(const CRectangle& t)
            {
                for (unsigned int i = 0; i < 4; ++i)
                    if (m_nodes[i] == t.m_nodes[0])
                        for (unsigned int j = 0; j < 4; ++j)
                            if (m_nodes[j] == t.m_nodes[1])
                                for (unsigned int k = 0; k < 4; ++k)
                                    if (m_nodes[k] == t.m_nodes[2])
                                        for (unsigned int l = 0; l < 4; ++l)
                                            if (m_nodes[l] == t.m_nodes[3])
                                        return true;
                return false;
            }
            std::istream& operator»(std::istream& is)
            {
                is » m_nodes[0] » m_nodes[1] » m_nodes[2] » m_nodes[3];
                return is;
            }
            ~CRectangle() {};
            const int                                    GetNode(const int) const;
            const int                                    GetNode(const NODES&) const;
            const int                                    GetEdge(const int) const;
            const int                                    GetFacet(const int) const;
            const int                                    GetNumberOfNodes() const;
            const int                                    GetNumberOfEdges() const;
            const int                                    GetNumberOfFacets() const;
            const double                                 Integrate(const std::function<const double(const Point&)>&, const std::vector<Point>& v) const;
            const Point                                     Integrate(const std::function<const Point(const Point&)>&, const std::vector<Point>& v) const;
            const std::vector<double>                    Integrate(const std::function<const std::vector<double>(const Point&)>&, const std::vector<Point>&) const;
            void                                         SetNode(const int k, const int node);
            const int                                    IncreaseOrder();
            const int                                    SetOrder(const int px, const int py);
            void                                         SetEdge(const int k, const int edge);
            void                                         SetFacet(const int k, const int facet);
        private:
            std::vector<int>                             m_nodes;
            int                                          m_edges[4];
            int                                          m_order;
            int                                          m_number;
            int                                          m_px, m_py;
        };

        class CRectangleBasis : public CShapeFunction<double>
        {
        public:
```

```
        CRectangleBasis();
        CRectangleBasis(const Point&, const Point&, const Point&, const Point&, const int order);
        CRectangleBasis(const Point*, const int order);
        CRectangleBasis(const CRectangleBasis&);
        CRectangleBasis& operator=(const CRectangleBasis& t)
        {
            m_normal = t.m_normal;
            m_det = t.m_det;
            m_order = t.m_order;
            m_1dorder = t.m_1dorder;
            m_number = t.m_number;
            m_s = t.m_s;
            m_sp = t.m_sp;
            m_points = t.m_points;
            m_hx = t.m_hx;
            m_hy = t.m_hy;
            return *this;
        }
        ~CRectangleBasis() {};
        const int                                    GetNumberOfShapeFunctions() const;
        //const DForm<0>*                                GetShapeFunction(const int, const Point&) const;
         const double                                 GetShapeFunction(const int, const Point&) const;
         const Point                                     GetGradShapeFunction(const int, const Point&) const;
         const Point                                     GetNormal() const;
         void                                         ReverseNormal();
         const double                                 GetValue(const Point&) const;
         const int                                    IncreaseOrder();
         //const int                                  SetValue(const int, CSolution* value);
         //CSolution*                                 GetValue(const unsigned int);
         //const CFESolution                          GetValue(const int) const;
         const double                                 GetMeasure() const { return m_det; };
         const double                                 GetWeight(const int, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const;
         //const unsigned int                            GetOrder() const;
         //const std::function<const DForm<0>*(const Point&)>  GetShapeFunction(const int) const;
    private:
         int                                          m_number;
         int                                          m_order;
         int                                          m_1dorder;
         Point                                        m_normal;
         std::vector<Mesh::Point>                     m_points;
         double                                       m_det;
         double                                       m_hx, m_hy;
         const double                                 m_x1(const double) const;
         const double                                 m_x2(const double) const;
         const double                                 m_y1(const double) const;
         const double                                 m_y2(const double) const;
         void                                         compD(const Point&, const Point&, const Point&);
         void                                         compNormal(const Point&, const Point&, const Point&);
         const int                                    createS();
         //std::vector<double>                            m_w[m_number];
         //std::vector<CFESolution>                   m_w{ 4 };
         int                                          m_s;
         int                                          m_sp;
    };


    class CRectangleHBasis : public CShapeFunction<double>
    {
    public:
         CRectangleHBasis();
         CRectangleHBasis(const Point&, const Point&, const Point&, const Point&, const int order);
         CRectangleHBasis(const Point&, const Point&, const Point&, const Point&, const int px, const int py);
         CRectangleHBasis(const Point*, const int order);
         CRectangleHBasis(const Point*, const int px, const int py);
         CRectangleHBasis(const CRectangleHBasis&);
         CRectangleHBasis& operator=(const CRectangleHBasis& t)
         {
             m_normal = t.m_normal;
             m_det = t.m_det;
             m_order = t.m_order;
             m_1dorder = t.m_1dorder;
             m_number = t.m_number;
             m_s = t.m_s;
             m_sp = t.m_sp;
             m_points = t.m_points;
             m_hx = t.m_hx;
             m_hy = t.m_hy;
             m_px = t.m_px;
             m_py = t.m_py;
             return *this;
```

```
        }
        ~CRectangleHBasis() {};
        const int                                            GetNumberOfShapeFunctions() const;
        //const DForm<0>*                                      GetShapeFunction(const int, const Point&) const;
        const double                                         GetShapeFunction(const int, const Point&) const;
        const Point                                          GetGradShapeFunction(const int, const Point&) const;
        const Point                                          GetNormal() const;
        void                                                 ReverseNormal();
        const double                                         GetValue(const Point&) const;
        const int                                            IncreaseOrder();
        const int                                            SetOrder(const int px, const int py);
        //const int                                          SetValue(const int, CSolution* value);
        //CSolution*                                         GetValue(const unsigned int);
        //const CFESolution                                  GetValue(const int) const;
        const double                                         GetMeasure() const { return m_det; };
        const double                                         GetWeight(const int, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const;
        //const unsigned int                                   GetOrder() const;
        //const std::function<const DForm<0>*(const Point&)>  GetShapeFunction(const int) const;
    private:
        int                                                  m_number;
        int                                                  m_order;
        int                                                  m_1dorder;
        Point                                                m_normal;
        std::vector<Mesh::Point>                             m_points;
        double                                               m_det;
        double                                               m_hx, m_hy;
        int                                                  m_px, m_py;
        const double                                         m_x1(const double) const;
        const double                                         m_x2(const double) const;
        const double                                         m_y1(const double) const;
        const double                                         m_y2(const double) const;
        const double                                         m_xi(const double, const int n) const;
        const double                                         m_dxi(const double, const int n) const;
        void                                                 compD(const Point&, const Point&, const Point&);
        void                                                 compNormal(const Point&, const Point&, const Point&);
        const int                                            createS();
        //std::vector<double>                                  m_w[m_number];
        //std::vector<CFESolution>                           m_w{ 4 };
        int                                                  m_s;
        int                                                  m_sp;
    };

    class CRectangleBasis2x : public CShapeFunction<double>
    {
    public:
        CRectangleBasis2x();
        CRectangleBasis2x(const Point&, const Point&, const Point&, const Point&, const int order);
        CRectangleBasis2x(const Point*, const int order);
        CRectangleBasis2x(const CRectangleBasis2x&);
        CRectangleBasis2x& operator=(const CRectangleBasis2x& t)
        {
            m_normal = t.m_normal;
            m_det = t.m_det;
            m_order = t.m_order;
            m_1dorder = t.m_1dorder;
            m_number = t.m_number;
            m_s = t.m_s;
            m_sp = t.m_sp;
            m_points = t.m_points;
            m_hx = t.m_hx;
            m_hy = t.m_hy;
            return *this;
        }
        ~CRectangleBasis2x() {};
        const int                                            GetNumberOfShapeFunctions() const;
        //const DForm<0>*                                      GetShapeFunction(const int, const Point&) const;
        const double                                         GetShapeFunction(const int, const Point&) const;
        const Point                                          GetGradShapeFunction(const int, const Point&) const;
        const Point                                          GetNormal() const;
        void                                                 ReverseNormal();
        const double                                         GetValue(const Point&) const;
        const int                                            IncreaseOrder();
        //const int                                          SetValue(const int, CSolution* value);
        //CSolution*                                         GetValue(const unsigned int);
        //const CFESolution                                  GetValue(const int) const;
        const double                                         GetMeasure() const { return m_det; };
        const double                                         GetWeight(const int, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const;
```

```
        //const unsigned int                                GetOrder() const;
        //const std::function<const DForm<0>*(const Point&)>  GetShapeFunction(const int) const;
    private:
        int                                                 m_number;
        int                                                 m_order;
        int                                                 m_1dorder;
        Point                                               m_normal;
        std::vector<Mesh::Point>                            m_points;
        double                                              m_det;
        double                                              m_hx, m_hy;
        const double                                        m_x1(const double) const;
        const double                                        m_x2(const double) const;
        const double                                        m_x3(const double) const;
        const double                                        m_dx3(const double) const;
        const double                                        m_y1(const double) const;
        const double                                        m_y2(const double) const;
        void                                                compD(const Point&, const Point&, const Point&);
        void                                                compNormal(const Point&, const Point&, const Point&);
        const int                                           createS();
        //std::vector<double>                                   m_w[m_number];
        //std::vector<CFESolution>                          m_w{ 4 };
        int                                                 m_s;
        int                                                 m_sp;
    };

    class CRectangleBasis2y : public CShapeFunction<double>
    {
    public:
        CRectangleBasis2y();
        CRectangleBasis2y(const Point&, const Point&, const Point&, const Point&, const int order);
        CRectangleBasis2y(const Point*, const int order);
        CRectangleBasis2y(const CRectangleBasis2y&);
        CRectangleBasis2y& operator=(const CRectangleBasis2y& t)
        {
            m_normal = t.m_normal;
            m_det = t.m_det;
            m_order = t.m_order;
            m_1dorder = t.m_1dorder;
            m_number = t.m_number;
            m_s = t.m_s;
            m_sp = t.m_sp;
            m_points = t.m_points;
            m_hx = t.m_hx;
            m_hy = t.m_hy;
            return *this;
        }
        ~CRectangleBasis2y() {};
        const int                                           GetNumberOfShapeFunctions() const;
        //const DForm<0>*                                       GetShapeFunction(const int, const Point&) const;
        const double                                        GetShapeFunction(const int, const Point&) const;
        const Point                                            GetGradShapeFunction(const int, const Point&) const;
        const Point                                            GetNormal() const;
        void                                                ReverseNormal();
        const double                                        GetValue(const Point&) const;
        const int                                           IncreaseOrder();
        //const int                                         SetValue(const int, CSolution* value);
        //CSolution*                                        GetValue(const unsigned int);
        //const CFESolution                                 GetValue(const int) const;
        const double                                        GetMeasure() const { return m_det; };
        const double                                        GetWeight(const int, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const;
        //const unsigned int                                GetOrder() const;
        //const std::function<const DForm<0>*(const Point&)>  GetShapeFunction(const int) const;
    private:
        int                                                 m_number;
        int                                                 m_order;
        int                                                 m_1dorder;
        Point                                               m_normal;
        std::vector<Mesh::Point>                            m_points;
        double                                              m_det;
        double                                              m_hx, m_hy;
        const double                                        m_x1(const double) const;
        const double                                        m_x2(const double) const;
        const double                                        m_y3(const double) const;
        const double                                        m_dy3(const double) const;
        const double                                        m_y1(const double) const;
        const double                                        m_y2(const double) const;
        void                                                compD(const Point&, const Point&, const Point&);
        void                                                compNormal(const Point&, const Point&, const Point&);
        const int                                           createS();
```

```
            //std::vector<double>                                 m_w[m_number];
            //std::vector<CFESolution>                          m_w{ 4 };
            int                                                 m_s;
            int                                                 m_sp;
        };

        class CRectangleBasis2 : public CShapeFunction<double>
        {
        public:
            CRectangleBasis2();
            CRectangleBasis2(const Point&, const Point&, const Point&, const Point&, const int order);
            CRectangleBasis2(const Point*, const int order);
            CRectangleBasis2(const CRectangleBasis2&);
            CRectangleBasis2& operator=(const CRectangleBasis2& t)
            {
                m_normal = t.m_normal;
                m_det = t.m_det;
                m_order = t.m_order;
                m_1dorder = t.m_1dorder;
                m_number = t.m_number;
                m_s = t.m_s;
                m_sp = t.m_sp;
                m_points = t.m_points;
                m_hx = t.m_hx;
                m_hy = t.m_hy;
                return *this;
            }
            ~CRectangleBasis2() {};
            const int                                           GetNumberOfShapeFunctions() const;
            //const DForm<0>*                                       GetShapeFunction(const int, const Point&) const;
            const double                                        GetShapeFunction(const int, const Point&) const;
            const Point                                            GetGradShapeFunction(const int, const Point&) const;
            const Point                                            GetNormal() const;
            void                                                ReverseNormal();
            const double                                        GetValue(const Point&) const;
            const int                                           IncreaseOrder();
            //const int                                         SetValue(const int, CSolution* value);
            //CSolution*                                        GetValue(const unsigned int);
            //const CFESolution                                 GetValue(const int) const;
            const double                                        GetMeasure() const { return m_det; };
            const double                                        GetWeight(const int, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const;
            //const unsigned int                                    GetOrder() const;
            //const std::function<const DForm<0>*(const Point&)>  GetShapeFunction(const int) const;
        private:
            int                                                 m_number;
            int                                                 m_order;
            int                                                 m_1dorder;
            Point                                               m_normal;
            std::vector<Mesh::Point>                            m_points;
            double                                              m_det;
            double                                              m_hx, m_hy;
            const double                                        m_x1(const double) const;
            const double                                        m_x2(const double) const;
            const double                                        m_x3(const double) const;
            const double                                        m_dx3(const double) const;
            const double                                        m_y1(const double) const;
            const double                                        m_y2(const double) const;
            const double                                        m_y3(const double) const;
            const double                                        m_dy3(const double) const;
            void                                                compD(const Point&, const Point&, const Point&);
            void                                                compNormal(const Point&, const Point&, const Point&);
            const int                                           createS();
            //std::vector<double>                                   m_w[m_number];
            //std::vector<CFESolution>                          m_w{ 4 };
            int                                                 m_s;
            int                                                 m_sp;
        };

        class CRectangleConstantBasis : public CShapeFunction<double>
        {
        public:
            CRectangleConstantBasis();
            CRectangleConstantBasis(const Point&, const Point&, const Point&, const Point&, const int order);
            CRectangleConstantBasis(const Point*, const int order);
            CRectangleConstantBasis(const CRectangleConstantBasis&);
            CRectangleConstantBasis& operator=(const CRectangleConstantBasis& t)
            {
                m_normal = t.m_normal;
                m_det = t.m_det;
                m_order = t.m_order;
```

```
                m_1dorder = t.m_1dorder;
                m_number = t.m_number;
                m_s = t.m_s;
                m_sp = t.m_sp;
                m_points = t.m_points;
                m_hx = t.m_hx;
                m_hy = t.m_hy;
                return *this;
            }
            ~CRectangleConstantBasis() {};
            const int                                                    GetNumberOfShapeFunctions() const;
            //const DForm<0>*                                               GetShapeFunction(const int, const Point&) const;
            const double                                                 GetShapeFunction(const int, const Point&) const;
            const Point                                                     GetGradShapeFunction(const int, const Point&) const;
            const Point                                                     GetNormal() const;
            void                                                         ReverseNormal();
            const double                                                 GetValue(const Point&) const;
            const int                                                    IncreaseOrder();
            //const int                                                  SetValue(const int, CSolution* value);
            //CSolution*                                                 GetValue(const unsigned int);
            //const CFESolution                                          GetValue(const int) const;
            const double                                                 GetMeasure() const { return m_det; };
            //const unsigned int                                            GetOrder() const;
            //const std::function<const DForm<0>*(const Point&)>  GetShapeFunction(const int) const;
        private:
            int                                                          m_number;
            int                                                          m_order;
            int                                                          m_1dorder;
            Point                                                        m_normal;
            std::vector<Mesh::Point>                                     m_points;
            double                                                       m_det;
            double                                                       m_hx, m_hy;
            const double                                                 m_x1(const double) const;
            const double                                                 m_x2(const double) const;
            const double                                                 m_y1(const double) const;
            const double                                                 m_y2(const double) const;
            void                                                         compD(const Point&, const Point&, const Point&);
            void                                                         compNormal(const Point&, const Point&, const Point&);
            const int                                                    createS();
            //std::vector<double>                                           m_w[m_number];
            //std::vector<CFESolution>                                   m_w{ 1 };
            int                                                          m_s;
            int                                                          m_sp;
        };
    }
}
#endif // CORENC_MESH_RECTANGLE_H_
```

7.32 CoreNCFEM/FiniteElements/RectangleBasis2.cpp File Reference

```
#include "Rectangle.h"
#include <iostream>
```

7.33 CoreNCFEM/FiniteElements/RectangleBasis2y.cpp File Reference

```
#include "Rectangle.h"
#include <iostream>
```

7.34 CoreNCFEM/FiniteElements/RectangleHBasis.cpp File Reference

```
#include "Rectangle.h"
#include <random>
```

```
#include <iostream>
#include "../../CoreNCA/Matrix.h"
#include "../../CoreNCA/MatrixSkyline.h"
```

7.35 CoreNCFEM/FiniteElements/Shape.h File Reference

```
#include <functional>
#include <vector>
#include "../Point.h"
```

Classes

- class corenc::Mesh::CShape

Namespaces

- namespace corenc
- namespace corenc::Mesh

Typedefs

- using corenc::scalar_func = std::function< const double(const Mesh::Point &)>
- using corenc::vector_func = std::function< const Mesh::Point(const Mesh::Point &)>

Enumerations

- enum class corenc::Mesh::NODES { corenc::Mesh::FIRST , corenc::Mesh::LAST }

7.36 Shape.h

Go to the documentation of this file.

```
#ifndef CORENC_MESH_Shape_h
#define CORENC_MESH_Shape_h
#include <functional>
#include <vector>
#include "../Point.h"
namespace corenc
{
    using scalar_func = std::function<const double(const Mesh::Point&)>;
    using vector_func = std::function<const Mesh::Point(const Mesh::Point&)>;
     namespace Mesh
     {
         //class Point;
         enum class NODES
         {
             FIRST,
             LAST
         };
         class CShape
         {
         public:
             CShape() {}
             CShape(const int*) {}
```

```
            virtual ~CShape() {}
            virtual const int                   GetNumberOfNodes() const { return 0; };
            virtual const int                   GetNumberOfEdges() const { return 0; };
            virtual const int                   GetNumberOfFacets() const { return 0; };
            virtual const int                   GetNode(const int) const { return 1; };
            virtual const int                   GetNode(const NODES&) const { return 1; };
            virtual const int                   GetEdge(const int) const { return -1; };
            virtual const int                   GetFacet(const int) const { return -1; };
            virtual const double                Integrate(const scalar_func&, const std::vector<Point>&) const = 0;
            virtual const Point                Integrate(const vector_func&, const std::vector<Point>&) const = 0;
            virtual const std::vector<double>   Integrate(const std::function<const std::vector<double>(const Point&)>&, const std::vector<Point>&) const = 0;
            virtual void                        SetNode(const int, const int) = 0;
            virtual void                        SetEdge(const int, const int) {};
            virtual void                        SetFacet(const int, const int) {};
            //virtual std::istream&         operator»(std::istream&) = 0;
        };
    }

}
#endif /* CORENC_MESH_Shape_h */
```

7.37 CoreNCFEM/FiniteElements/ShapeFunction.h File Reference

```
#include "../Point.h"
#include <functional>
#include "../FESolution.h"
```

Classes

- class corenc::Mesh::CShapeFunction< Type >

Namespaces

- namespace corenc
- namespace corenc::Mesh

7.38 ShapeFunction.h

Go to the documentation of this file.

```
#ifndef CORENC_MESH_ShapeFunction_h
#define CORENC_MESH_ShapeFunction_h
#include "../Point.h"
#include <functional>
#include "../FESolution.h"
namespace corenc
{
    namespace Mesh
    {
        template<class Type>
        class CShapeFunction
        {
        public:
            CShapeFunction() {}
            CShapeFunction(const Point*) {}
            virtual ~CShapeFunction() {}
            virtual const int               GetNumberOfShapeFunctions() const = 0;
            //virtual const  std::function<const DiffForm*(const Point&)> GetShapeFunction(const int) const = 0;
            //virtual const DiffForm*  GetShapeFunction(const int) const = 0;
            virtual const double            GetShapeFunction(const int, const Point&) const = 0;
            virtual const Point                GetGradShapeFunction(const int, const Point&) const = 0;
```

```
            virtual const Point                GetNormal() const = 0;
            virtual void                    ReverseNormal() = 0;
            virtual const double            GetMeasure() const = 0;
            //virtual const Type                  GetValue(const Point&) const = 0;
            //virtual const int             SetValue(const unsigned int, const Type& value) = 0;
            //virtual const Type                  GetValue(const unsigned int) const = 0;
            //virtual const int             SetValue(const )
            //virtual CSolution*        GetValue(const unsigned int) = 0;
            //virtual const int             SetValue(const int, CSolution*) = 0;
        };
    }
}
#endif /* CORENC_MESH_ShapeFunction_h */
```

7.39 CoreNCFEM/FiniteElements/Triangle.cpp File Reference

```
#include "Triangle.h"
#include <iostream>
#include <algorithm>
#include <random>
#include "../../CoreNCA/Matrix.h"
#include "../../CoreNCA/MatrixSkyline.h"
```

Functions

- const Point mid_point (const Point &p1, const Point &p2)
- const Point s_point (const Point &p1, const Point &p2, const double s)
- const Point center_point (const Point &p1, const Point &p2, const Point &p3)

7.39.1 Function Documentation

7.39.1.1 center_point()

```
const Point center_point (
            const Point & p1,
            const Point & p2,
            const Point & p3 )
```

7.39.1.2 mid_point()

```
const Point mid_point (
            const Point & p1,
            const Point & p2 )
```

7.39.1.3 s_point()

```
const Point s_point (
            const Point & p1,
            const Point & p2,
            const double s )
```

7.40 CoreNCFEM/FiniteElements/Triangle.h File Reference

```
#include <stdio.h>
#include "Shape.h"
#include "ShapeFunction.h"
#include <iostream>
```

Classes

- class corenc::Mesh::CTriangle
- class corenc::Mesh::CTriangleBasis
- class corenc::Mesh::CTriangleLagrangeBasis

Namespaces

- namespace corenc
- namespace corenc::Mesh

7.41 Triangle.h

Go to the documentation of this file.

```
#ifndef CORENC_MESH_TRIANGLE_H_
#define CORENC_MESH_TRIANGLE_H_

#include <stdio.h>
#include "Shape.h"
#include "ShapeFunction.h"
#include <iostream>
namespace corenc
{
    namespace Mesh
    {
        class CTriangle : public CShape
        {
        public:
            CTriangle();
            CTriangle(const int n1, const int n2, const int n3, const int order);
            CTriangle(const int n1, const int n2, const int n3, const int e1, const int e2, const int e3, const int order);
            CTriangle(const int*, const int order);
            CTriangle(const int*, const int*, const int order);
            CTriangle(const CTriangle&);
            CTriangle& operator=(const CTriangle& t)
            {
                m_nodes = t.m_nodes;
                m_edges[0] = t.m_edges[0];
                m_edges[1] = t.m_edges[1];
                m_edges[2] = t.m_edges[2];
                m_number = t.m_number;
                m_order = t.m_order;
                return *this;
            }
```

```
            const bool    operator==(const CTriangle& t)
            {
                for (unsigned int i = 0; i < 3; ++i)
                    if (m_nodes[i] == t.m_nodes[0])
                        for (unsigned int j = 0; j < 3; ++j)
                            if (m_nodes[j] == t.m_nodes[1])
                                for (unsigned int k = 0; k < 3; ++k)
                                    if (m_nodes[k] == t.m_nodes[2])
                                        return true;
                return false;
            }
            std::istream& operator»(std::istream& is)
            {
                is » m_nodes[0] » m_nodes[1] » m_nodes[2];
                return is;
            }
            ~CTriangle() {};
            const int                                           GetNode(const int) const;
            const int                                           GetNode(const NODES&) const;
            const int                                           GetEdge(const int) const;
            const int                                           GetFacet(const int) const;
            const int                                           GetNumberOfNodes() const;
            const int                                           GetNumberOfEdges() const;
            const int                                           GetNumberOfFacets() const;
            const double                                        Integrate(const std::function<const double(const Point&)>&, const std::vector<Point>& v) const;
            const Point                                            Integrate(const std::function<const Point(const Point&)>&, const std::vector<Point>& v) const;
            const std::vector<double>                           Integrate(const std::function<const std::vector<double>(const Point&)>&, const std::vector<Point>&) const;
            void                                                SetNode(const int k, const int node);
            const int                                           IncreaseOrder();
            void                                                SetEdge(const int k, const int edge);
            void                                                SetFacet(const int k, const int facet);
        private:
            std::vector<int>                                    m_nodes;
            int                                                 m_edges[3];
            int                                                 m_order;
            int                                                 m_number;
            void                                                SetOrder();
        };

        class CTriangleBasis : public CShapeFunction<double>
        {
        public:
            CTriangleBasis();
            CTriangleBasis(const Point&, const Point&, const Point&, const int order);
            CTriangleBasis(const Point*, const int order);
            CTriangleBasis(const CTriangleBasis&);
            CTriangleBasis& operator=(const CTriangleBasis& t)
            {
                m_normal = t.m_normal;
                m_det = t.m_det;
                m_order = t.m_order;
                m_1dorder = t.m_1dorder;
                m_number = t.m_number;
                m_alpha[0][0] = t.m_alpha[0][0];
                m_alpha[0][1] = t.m_alpha[0][1];
                m_alpha[0][2] = t.m_alpha[0][2];

                m_alpha[1][0] = t.m_alpha[1][0];
                m_alpha[1][1] = t.m_alpha[1][1];
                m_alpha[1][2] = t.m_alpha[1][2];

                m_alpha[2][0] = t.m_alpha[2][0];
                m_alpha[2][1] = t.m_alpha[2][1];
                m_alpha[2][2] = t.m_alpha[2][2];
                m_s = t.m_s;
                m_sp = t.m_sp;
                m_all = t.m_all;
                return *this;
            }
             ~CTriangleBasis() {};
             const int                                          GetNumberOfShapeFunctions() const;
             //const DForm<0>*                                        GetShapeFunction(const int, const Point&) const;
             const double                                       GetShapeFunction(const int, const Point&) const;
             const Point                                           GetGradShapeFunction(const int, const Point&) const;
             const Point                                           GetNormal() const;
             void                                               ReverseNormal();
             const double                                       GetValue(const Point&) const;
             const int                                          IncreaseOrder();
             //const int                                        SetValue(const int, CSolution* value);
             //CSolution*                                       GetValue(const unsigned int);
             //const CFESolution                                GetValue(const int) const;
```

```
        const double                                        GetMeasure() const { return fabs(m_det); };
        const double                                        GetWeight(const int, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const;
        //const unsigned int                                    GetOrder() const;
        //const std::function<const DForm<0>*(const Point&)>  GetShapeFunction(const int) const;
    private:
        int                                                 m_number;
        int                                                 m_order;
        int                                                 m_1dorder;
        double                                              m_alpha[3][3];
        Point                                               m_normal;
        double                                              m_det;
        const double                                        m_L(const int, const Point&) const;
        const double                                        m_xi(const int, const Point&) const;
        void                                                compD(const Point&, const Point&, const Point&);
        void                                                compAlpha(const Point&, const Point&, const Point&);
        void                                                compNormal(const Point&, const Point&, const Point&);
        const int                                           createS();
        //std::vector<double>                                   m_w[m_number];
        //std::vector<CFESolution>                          m_w{ 3 };
        int                                                 m_s;
        int                                                 m_sp;
        std::vector<int>                                    m_all;
    };

    class CTriangleLagrangeBasis : public CShapeFunction<double>
    {
    public:
        CTriangleLagrangeBasis();
        CTriangleLagrangeBasis(const Point&, const Point&, const Point&, const int order);
        CTriangleLagrangeBasis(const Point*, const int order);
        CTriangleLagrangeBasis(const CTriangleLagrangeBasis&);
        CTriangleLagrangeBasis& operator=(const CTriangleLagrangeBasis& t)
        {
            m_normal = t.m_normal;
            m_det = t.m_det;
            m_order = t.m_order;
            m_1dorder = t.m_1dorder;
            m_number = t.m_number;
            m_alpha[0][0] = t.m_alpha[0][0];
            m_alpha[0][1] = t.m_alpha[0][1];
            m_alpha[0][2] = t.m_alpha[0][2];

            m_alpha[1][0] = t.m_alpha[1][0];
            m_alpha[1][1] = t.m_alpha[1][1];
            m_alpha[1][2] = t.m_alpha[1][2];

            m_alpha[2][0] = t.m_alpha[2][0];
            m_alpha[2][1] = t.m_alpha[2][1];
            m_alpha[2][2] = t.m_alpha[2][2];
            m_s = t.m_s;
            m_sp = t.m_sp;
            m_all = t.m_all;
            return *this;
        }
        ~CTriangleLagrangeBasis() {};
        const int                                           GetNumberOfShapeFunctions() const;
        //const DForm<0>*                                       GetShapeFunction(const int, const Point&) const;
        const double                                        GetShapeFunction(const int, const Point&) const;
        const Point                                             GetGradShapeFunction(const int, const Point&) const;
        const Point                                             GetNormal() const;
        void                                                ReverseNormal();
        const double                                        GetValue(const Point&) const;
        const int                                           IncreaseOrder();
        //const int                                         SetValue(const int, CSolution* value);
        //CSolution*                                        GetValue(const unsigned int);
        //const CFESolution                                 GetValue(const int) const;
        const double                                        GetAlpha(const int i, const int j) const { return m_alpha[i][j]; }
        const double                                        GetMeasure() const { return fabs(m_det); };
        const double                                        GetWeight(const int, const std::vector<Point>& verts, const std::function<const double(const Point&)>& f) const;
        //const unsigned int                                    GetOrder() const;
        //const std::function<const DForm<0>*(const Point&)>  GetShapeFunction(const int) const;
    private:
        int                                                 m_number;
        int                                                 m_order;
        int                                                 m_1dorder;
        double                                              m_alpha[3][3];
```

```
            Point                                  m_normal;
            double                                 m_det;
            const double                           m_L(const int, const Point&) const;
            const double                           m_xi(const int, const Point&) const;
            void                                   compD(const Point&, const Point&, const
    Point&);
            void                                   compAlpha(const Point&, const Point&,
    const Point&);
            void                                   compNormal(const Point&, const Point&,
    const Point&);
            const int                              createS();
            //std::vector<double>                      m_w[m_number];
            //std::vector<CFESolution>             m_w{ 3 };
            int                                    m_s;
            int                                    m_sp;
            std::vector<int>                       m_all;
        };
    }
}
#endif // CORENC_MESH_TRIANGLE_H_
```

7.42 CoreNCFEM/FiniteElements/TriangleLagrange.cpp File Reference

```
#include "Triangle.h"
#include <iostream>
#include <algorithm>
#include <random>
#include "../../CoreNCA/Matrix.h"
#include "../../CoreNCA/MatrixSkyline.h"
```

Namespaces

- namespace wtf

Functions

- const Point wtf::mid_point (const Point &p1, const Point &p2)
- const Point wtf::s_point (const Point &p1, const Point &p2, const double s)
- const Point wtf::center_point (const Point &p1, const Point &p2, const Point &p3)

7.43 CoreNCFEM/FiniteElements/TriangleLinear.cpp File Reference

```
#include "TriangleLinear.h"
#include <iostream>
```

7.44 CoreNCFEM/FiniteElements/TriangleLinear.h File Reference

```
#include <stdio.h>
#include "Shape.h"
#include "ShapeFunction.h"
#include <iostream>
```

Classes

- class corenc::Mesh::CTriangleLinear
- class corenc::Mesh::CTriangleLinearBasis
- class corenc::Mesh::CTriangleBasis

Namespaces

- namespace corenc
- namespace corenc::Mesh

7.45 TriangleLinear.h

Go to the documentation of this file.

```
#ifndef CORENC_MESH_TRIANGLELINEAR_H_
#define CORENC_MESH_TRIANGLELINEAR_H_

#include <stdio.h>
#include "Shape.h"
#include "ShapeFunction.h"
#include <iostream>
namespace corenc
{
    namespace Mesh
    {
        class CTriangleLinear : public CShape
        {
        public:
            CTriangleLinear();
            CTriangleLinear(const int n1, const int n2, const int n3);
            CTriangleLinear(const int n1, const int n2, const int n3, const int e1, const int e2, const int e3);
            CTriangleLinear(const int*);
            CTriangleLinear(const int*, const int*);
            CTriangleLinear(const CTriangleLinear&);
            CTriangleLinear& operator=(const CTriangleLinear& t)
            {
                m_nodes[0] = t.m_nodes[0];
                m_nodes[1] = t.m_nodes[1];
                m_nodes[2] = t.m_nodes[2];
                m_edges[0] = t.m_edges[0];
                m_edges[1] = t.m_edges[1];
                m_edges[2] = t.m_edges[2];
                return *this;
            }
            const bool    operator==(const CTriangleLinear& t)
            {
                for (unsigned int i = 0; i < 3; ++i)
                    if (m_nodes[i] == t.m_nodes[0])
                        for (unsigned int j = 0; j < 3; ++j)
                            if (m_nodes[j] == t.m_nodes[1])
                                for (unsigned int k = 0; k < 3; ++k)
                                    if (m_nodes[k] == t.m_nodes[2])
                                        return true;
                return false;
            }
            std::istream& operator»(std::istream& is)
            {
                is » m_nodes[0] » m_nodes[1] » m_nodes[2];
                return is;
            }
            ~CTriangleLinear() {};
            const int                                   GetNode(const int) const;
            const int                                   GetNode(const NODES&) const;
            const int                                   GetEdge(const int) const;
            const int                                   GetFacet(const int) const;
            const int                                   GetNumberOfNodes() const;
            const int                                   GetNumberOfEdges() const;
            const int                                   GetNumberOfFacets() const;
            const double                                Integrate(const std::function<const double(const Point&)>&, const std::vector<Point>& v) const;
            const Point                                    Integrate(const std::function<const Point(const Point&)>&, const std::vector<Point>& v) const;
```

```
        const std::vector<double>                    Integrate(const std::function<const std::vector<double>(const Point&)>&, const std::vector<Point>&) const;
        void                                         SetNode(const int k, const int node);
        const int                                    IncreaseOrder() { return 1; };
        void                                         SetEdge(const int k, const int edge);
        void                                         SetFacet(const int k, const int facet);
    private:
        int                                          m_nodes[3];
        int                                          m_edges[3];
    };

    class CTriangleLinearBasis : public CShapeFunction<double>
    {
    public:
        CTriangleLinearBasis();
        CTriangleLinearBasis(const Point&, const Point&, const Point&);
        CTriangleLinearBasis(const Point*);
        CTriangleLinearBasis(const CTriangleLinearBasis&);
        CTriangleLinearBasis& operator=(const CTriangleLinearBasis& t)
        {
            m_normal = t.m_normal;
            m_det = t.m_det;
            m_alpha[0][0] = t.m_alpha[0][0];
            m_alpha[0][1] = t.m_alpha[0][1];
            m_alpha[0][2] = t.m_alpha[0][2];

            m_alpha[1][0] = t.m_alpha[1][0];
            m_alpha[1][1] = t.m_alpha[1][1];
            m_alpha[1][2] = t.m_alpha[1][2];

            m_alpha[2][0] = t.m_alpha[2][0];
            m_alpha[2][1] = t.m_alpha[2][1];
            m_alpha[2][2] = t.m_alpha[2][2];
            return *this;
        }
        ~CTriangleLinearBasis() {};
        const int                                    GetNumberOfShapeFunctions() const;
        //const DForm<0>*                                GetShapeFunction(const int, const Point&) const;
        const double                                 GetShapeFunction(const int, const Point&) const;
        const Point                                     GetGradShapeFunction(const int, const Point&) const;
        const Point                                     GetNormal() const;
        void                                         ReverseNormal();
        const double                                 GetValue(const Point&) const;
        const int                                    IncreaseOrder() { return 1; };
         //const int                                    SetValue(const int, CSolution* value);
         //CSolution*                                   GetValue(const unsigned int);
         //const CFESolution                            GetValue(const int) const;
         const double                                 GetMeasure() const { return m_det; };
         //const std::function<const DForm<0>*(const Point&)>  GetShapeFunction(const int) const;
     private:
         static const int                             m_number = 3;
         double                                       m_alpha[3][3];
         Point                                        m_normal;
         double                                       m_det;
         const double                                 L(const int, const Point&) const;
         void                                         compD(const Point&, const Point&, const Point&);
         void                                         compAlpha(const Point&, const Point&, const Point&);
         void                                         compNormal(const Point&, const Point&, const Point&);
         //std::vector<double>                              m_w[m_number];
         //std::vector<CFESolution>                     m_w{3};
     };

     class CTriangleBasis : public CShapeFunction<double>
     {
     public:
         CTriangleBasis();
         CTriangleBasis(const Point&, const Point&, const Point&, const int order);
         CTriangleBasis(const Point*, const int order);
         CTriangleBasis(const CTriangleBasis&);
         CTriangleBasis& operator=(const CTriangleBasis& t)
         {
             m_normal = t.m_normal;
             m_det = t.m_det;
             m_order = t.m_order;
             m_number = t.m_number;
             m_alpha[0][0] = t.m_alpha[0][0];
             m_alpha[0][1] = t.m_alpha[0][1];
             m_alpha[0][2] = t.m_alpha[0][2];

             m_alpha[1][0] = t.m_alpha[1][0];
             m_alpha[1][1] = t.m_alpha[1][1];
```

```
                m_alpha[1][2] = t.m_alpha[1][2];

                m_alpha[2][0] = t.m_alpha[2][0];
                m_alpha[2][1] = t.m_alpha[2][1];
                m_alpha[2][2] = t.m_alpha[2][2];
                return *this;
            }
            ~CTriangleBasis() {};
            const int                                           GetNumberOfShapeFunctions() const;
            //const DForm<0>*                                       GetShapeFunction(const int, const Point&) const;
            const double                                        GetShapeFunction(const int, const Point&) const;
            const Point                                            GetGradShapeFunction(const int, const Point&) const;
            const Point                                            GetNormal() const;
            void                                                ReverseNormal();
            const double                                        GetValue(const Point&) const;
            //const int                                         SetValue(const int, CSolution* value);
            //CSolution*                                        GetValue(const unsigned int);
            //const CFESolution                                 GetValue(const int) const;
            //const std::function<const DForm<0>*(const Point&)>  GetShapeFunction(const int) const;
        private:
            int                                                 m_number;
            int                                                 m_order;
            double                                              m_alpha[3][3];
            Point                                               m_normal;
            double                                              m_det;
            const double                                        L(const int, const Point&) const;
            void                                                compD(const Point&, const Point&, const Point&);
            void                                                compAlpha(const Point&, const Point&, const Point&);
            void                                                compNormal(const Point&, const Point&, const Point&);
            //std::vector<double>                                   m_w[m_number];
            //std::vector<CFESolution>                          m_w{ 3 };
        };
    }
}
#endif // CORENC_MESH_TRIANGLELINEAR_H_
```

7.46 CoreNCFEM/FiniteSolver.h File Reference

Classes

- class corenc::CFiniteSolver< Method, Mesh, Solver >

Namespaces

- namespace corenc

7.47 FiniteSolver.h

Go to the documentation of this file.

```
#ifndef CORENC_FINITESOLVER_H
#define CORENC_FINITESOLVER_H

namespace corenc
{
    template<class Method, class Mesh, class Solver>
    class CFiniteSolver
    {
    public:
        CFiniteSolver() {};
        ~CFiniteSolver() {};
        void                        Solve();
     private:
        Method*                     m_method;
        Mesh*                       m_mesh;
```

```
        Solver*                     m_solver;
    };

    template<class Method, class Mesh, class Solver>
    void CFiniteSolver<Method, Mesh, Solver>::Solve()
    {
        m_method->Assemble();
        return;
    }
}

#endif // !CORENC_FINITESOLVER_H
```

7.48 CoreNCFEM/GaussianField.h File Reference

```
#include <algorithm>
#include <vector>
#include <cmath>
#include "Point.h"
```

Classes

- struct corenc::GaussianProcess
- struct corenc::GaussianKernel

Namespaces

- namespace corenc

7.49 GaussianField.h

Go to the documentation of this file.

```
#ifndef CORENC_GAUSSIANPROCESS_H_
#define CORENC_GAUSSIANPROCESS_H_
#include <algorithm>
#include <vector>
#include <cmath>
#include "Point.h"
namespace corenc
{
    struct GaussianProcess
    {
        double sigma2;
        double l;
        double a;
        double b;
        double c;
        double A;
        double B;
        size_t K = 1;
        std::vector<double> lambda;
        GaussianProcess(const double L, const size_t num)
        {
            K = num;
            sigma2 = L;
            l = 2 * L;
            a = 1. / (4 * sigma2);
            b = 1. / (2 * l * l);
            c = sqrt(a * a + 2 * a * b);
            A = a + b + c;
            B = b / A;
            lambda.resize(K);
```

```
            for (size_t i = 0; i < K; ++i)
                lambda[i] = std::pow(B, i) * sqrt(2 * a / A);
        }
        const double He(const int i, const double x) const
        {
            switch (i)
            {
            case 0:
                return 1.;
            case 1:
                return x;
            case 2:
                return x * x - 1.;
            case 3:
                return x * x * x - 3. * x;
            case 4:
                return x * x * x * x - 6. * x * x + 3.;
            case 5:
                return x * x * x * x * x - 10. * x * x * x + 15. * x;
            case 6:
                return x * x * x * x * x * x - 15. * x * x * x * x + 45. * x * x - 25.;
            case 7:
                return x * x * x * x * x * x * x - 21. * x * x * x * x * x + 105. * x * x * x - 105. * x;
            case 8:
                return x * x * x * x * x * x * x * x - 28. * x * x * x * x * x * x + 210. * x * x * x * x
    - 420. * x * x + 105;
            case 9:
                return x * x * x * x * x * x * x * x * x - 36. * x * x * x * x * x * x * x + 378. * x * x * x
    * x * x - 1260. * x * x * x + 945. * x;
            default:
                return x * x * x * x * x * x * x * x * x * x - 45. * x * x * x * x * x * x * x * x + 630.
    * x * x * x * x * x * x - 3150. * x * x * x *x + 4725. * x * x - 945.;
                break;
            }
        };
        const double phi(const int i, const double x) const
        {
            return exp(-(c - a) * x * x) * He(i, x * sqrt(2 * c));
        };
    };
    /*enum class gkernels
    {
        gexponent,
        gker1,
        gker2,
        gker3
    };*/
    struct GaussianKernel
    {
        int N;
        const double gpexp(const Mesh::Point& a) const
        {
            return exp(-12.5 * (a.x * a.x + a.y * a.y));
        }

        const double gpstep(const Mesh::Point& a) const
        {
            if (fabs(a.x) < 0.5 && fabs(a.y) < 0.5)
                return 1.;
            return 0.;
        }
        std::vector<Mesh::Point> _centrs;
        GaussianKernel(const int _n, const std::vector<Mesh::Point>& centers) :
            N{ _n }, _centrs{ centers } {}
        const double get_gp(const std::vector<double>& a, const Mesh::Point& p) const
        {
            double sum = 0;
            for (auto i = 0; i < N; ++i)
                sum += a[i] * gpexp(p - _centrs[i]);
            return sum;
        }

     };
}
#endif // CORENC_GAUSSIANPROCESS_H_
```

7.50 CoreNCFEM/Grids/Mesh1D.cpp File Reference

```
#include "Mesh1D.h"
#include "../FiniteElements/Node.h"
```

```
#include "../FiniteElements/Edge.h"
#include <iostream>
```

7.51 CoreNCFEM/Grids/Mesh1D.h File Reference

```
#include <stdio.h>
#include "../Mesh.h"
#include "../Point.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <map>
#include <unordered_map>
#include <typeinfo>
```

Classes

- class corenc::Mesh::CMesh1D

Namespaces

- namespace corenc
- namespace corenc::Mesh

7.52 Mesh1D.h

Go to the documentation of this file.

```
#ifndef CORENC_Mesh1D_hpp
#define CORENC_Mesh1D_hpp

#include <stdio.h>
#include "../Mesh.h"

#include "../Point.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <map>
#include <unordered_map>
#include <typeinfo>
namespace corenc
{
    namespace Mesh
    {

        class CMesh1D : public CMesh<CFESolution>
        {
        public:
            CMesh1D();
            CMesh1D(const std::string& domain_name);
            CMesh1D(const std::string& domain_file, const std::string& init_file);
            CMesh1D(const double x0, const double x1, const unsigned n, const int order, const
    std::function<const double(const Point&)>& init_func);
            CMesh1D(const double x0, const double x1, const unsigned n, const int order, const
    std::function<const double(const Point&)>& init_func, const std::function<const double(const
    Point&)>& init_derivative);
```

```
            CMesh1D(const CMesh1D&);
            CMesh1D& operator=(const CMesh1D& m)
            {
                auto sz = m.m_elems.size();
                m_elems.resize(sz);
                for (int i = 0; i < sz; ++i)
                    m_elems[i] = m.m_elems[i]->Clone();
            }
            const unsigned int                          GetNumberOfElements() const;
            const unsigned int                          GetNumberOfNodes() const;
            const unsigned int                          GetNumberOfBoundaries() const;
            const int                                   FindElement(const Point&) const;
            const Point                                    GetNode(const unsigned int) const;
            const CElement<CFESolution>*               GetElement(const unsigned int) const;
            const CElement<CFESolution>*               GetBoundary(const unsigned int) const;
            const double                                getSolution(const unsigned int element, const unsigned int node) const;
            const double                                getParameter(Parameters, const unsigned int, const Point& p) const;
            const double                                getParameter(Parameters, const unsigned int, const int) const;
            const std::vector<double>                   getSolution() const { return m_solution; };
            const int                                   updateSolution(const std::vector<double>& new_solution);
            const int                                   updateSolution(const unsigned int element, const unsigned int node, const double value);
            const int                                   updateSolution(const unsigned int element, const unsigned int node, CSolution* value);
            const int                                   updateSolution(const unsigned int node, const double value);
            const int                                   setParameter(Parameters, const double, const unsigned int);
            const double                                getMinSize() const { return m_minsize; };
            ~CMesh1D();
        private:
            std::vector<CElement<CFESolution>*>         m_elems;
            std::vector<CElement<CFESolution>*>         m_bnds;
            std::vector<Point>                          m_points;
            std::vector<double>                         m_solution;
            std::vector<int>                            m_nums;
            std::vector<double>                         m_params;
            double                                      m_minsize{0.};
        public:
            auto                                        GetElements() -> decltype(m_elems) { return m_elems; };
            auto                                        GetBoundary() -> decltype(m_bnds) { return m_bnds; };
        };
    }
}
#endif /* CORENC_Mesh1D_hpp */
```

7.53 CoreNCFEM/Grids/RegularMesh.cpp File Reference

```
#include <stdio.h>
#include "RegularMesh.h"
#include "../FiniteElements/Rectangle.h"
#include "../FiniteElements/Edge.h"
#include <iostream>
#include <algorithm>
#include <numeric>
```

Functions

- template< class T >
  vector< size_t > sort_indexes (const vector< T > &v)

7.53.1 Function Documentation

7.53.1.1 sort_indexes()

```
template<class T >
vector< size_t > sort_indexes (
            const vector< T > & v )
```

7.54 CoreNCFEM/Grids/RegularMesh.h File Reference

```
#include "../Mesh.h"
#include "../FiniteElements/FiniteElement2D.h"
#include "../Point.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <map>
#include "../Parameter.h"
```

Classes

- class corenc::Mesh::CRegularMesh

Namespaces

- namespace corenc
- namespace corenc::Mesh

7.55 RegularMesh.h

Go to the documentation of this file.

```
#ifndef CORENC_MESH_RegularMesh_h
#define CORENC_MESH_RegularMesh_h
#include "../Mesh.h"
#include "../FiniteElements/FiniteElement2D.h"
#include "../Point.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <map>
#include "../Parameter.h"
namespace corenc
{
    namespace Mesh
    {
        class CRegularMesh// : public CMesh<>
        {
        public:
            CRegularMesh();
            CRegularMesh(const std::string& file_name);
            CRegularMesh(const CRegularMesh&);
            // nx ny number of elements on x y
            CRegularMesh(const Point& p1, const Point& p2, const int nx, const int ny);
            CRegularMesh(const Point& p1, const Point& p2, const int nx, const int ny, const int px,
    const int py);
            CRegularMesh(const double x1, const double y1, const double x2, const double y2, const int
    nx, const int ny);
            CRegularMesh& operator=(const CRegularMesh& tr)
```

```
        {
            const int sz_el = (int)tr.m_elems.size();
            const int sz_pt = (int)tr.m_points.size();
            const int sz_bpt = (int)tr.m_basepoints.size();
            const int sz_bel = (int)tr.m_elemsbase.size();
            const int sz_ed = (int)tr.m_edges.size();
            const int sz_bed = (int)tr.m_edgesbase.size();
            m_elems.resize(sz_el);
            m_edges.resize(sz_ed);
            m_points.resize(sz_pt);
            m_basepoints.resize(sz_bpt);
            m_elemsbase.resize(sz_bel);
            m_edgesbase.resize(sz_bed);
            int i = 0;
            for (i = 0; i < sz_el; ++i)
                m_elems[i] = tr.m_elems[i]->Clone();
            for (i = 0; i < sz_ed; ++i)
                m_edges[i] = tr.m_edges[i]->Clone();
            for (i = 0; i < sz_pt; ++i)
                m_points[i] = tr.m_points[i];
            for (i = 0; i < sz_bpt; ++i)
                m_basepoints[i] = tr.m_basepoints[i];
            for (i = 0; i < sz_bel; ++i)
                m_elemsbase[i] = tr.m_elemsbase[i]->Clone();
            for (i = 0; i < sz_bed; ++i)
                m_edgesbase[i] = tr.m_edgesbase[i]->Clone();
            m_bnds.resize(tr.m_bnds.size());
            for (i = 0; i < m_bnds.size(); ++i)
                m_bnds[i] = tr.m_bnds[i];
            m_offsets = tr.m_offsets;
            m_params = tr.m_params;
            m_order = tr.m_order;
            m_inodes = tr.m_inodes;
            return *this;
        }
        CRegularMesh*           Clone() const
        {
            return new CRegularMesh(*this);
        };
        const unsigned int                  GetNumberOfElements() const;
        const unsigned int                  GetNumberOfNodes() const;
        const int                           GetNumberOfINodes() const;
        const unsigned int                  GetNumberOfBoundaries() const;
        const int                           FindElement(const Point&) const;
        const Point                            GetNode(const unsigned int) const;
        const CElement2D<>*                 GetElement(const unsigned int) const;
        const CElement<>*                 GetBoundary(const unsigned int) const;
        const double                        getMinSize() const { return 0.; };
        const double                        getSolution(const unsigned int element, const unsigned int node) const;
        const int                           updateSolution(const unsigned int element, const unsigned int node, const double value);
        const std::vector<double>           getSolution() const;
        const int                           updateSolution(const std::vector<double>&);
        const int                           updateSolution(const unsigned int element, const unsigned int node, CSolution* value);
        const double                        getParameter(Parameters, const unsigned int, const Point&) const;
        const double                        getParameter(Parameters, const unsigned int, const int) const;
        const int                           setParameter(Parameters, const double, const unsigned int);
        const int                           setParameter(const CParameter&, const unsigned int type);
        const int                           updateSolution(const unsigned int node, const double value);
        const int                           refine_hx();
        const int                           refine_hy();
        const int                           refine_h();
        const int                           refine_p();
        const int                           refine_hp();
        const int                           interpolate(const int node) const;
        ~CRegularMesh();
    private:
        std::vector<CElement2D<>*>          m_elems;
        std::vector<CElement<>*>            m_edges;
        std::vector<Point>                  m_points;
        std::vector<Point>                  m_basepoints;
        std::vector<CElement2D<>*>          m_elemsbase;
        std::vector<CElement<>*>            m_edgesbase;
        std::map<int, CParameter>           m_params;
         std::vector<int>                    m_offsets;
         std::vector<int>                    m_bnds;
         const double                        CompSquare(const Point& p1, const Point& p2, const Point& p3) const;
         int                                 m_order;
         int                                 m_inodes;
     public:
```

```
            auto                         GetElements() -> decltype(m_elems) { return m_elems; };
            auto                         GetBoundary() -> decltype(m_edges) { return m_edges; };
        };
    }
}
#endif /* CORENC_MESH_RegularMesh_h */

```

7.56 CoreNCFEM/Grids/RegularMesh3D.cpp File Reference

```
#include <stdio.h>
#include "RegularMesh3D.h"
#include "../FiniteElements/Cube.h"
#include "../FiniteElements/Edge.h"
#include <iostream>
#include <algorithm>
#include <numeric>
```

Functions

- template< class T >
  vector< size_t > sort_indexes (const vector< T > &v)

7.56.1 Function Documentation

7.56.1.1 sort_indexes()

```
template<class T >
vector< size_t > sort_indexes (
            const vector< T > & v )
```

7.57 CoreNCFEM/Grids/RegularMesh3D.h File Reference

```
#include "../Mesh.h"
#include "../FiniteElements/FiniteElement2D.h"
#include "../Point.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <map>
#include "../Parameter.h"
```

Classes

- class corenc::Mesh::CRegularMesh3D

Namespaces

- namespace corenc
- namespace corenc::Mesh

7.58 RegularMesh3D.h

Go to the documentation of this file.

```
#ifndef CORENC_MESH_RegularMesh3D_h
#define CORENC_MESH_RegularMesh3D_h
#include "../Mesh.h"
#include "../FiniteElements/FiniteElement2D.h"
#include "../Point.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <map>
#include "../Parameter.h"
namespace corenc
{
    namespace Mesh
    {
        class CRegularMesh3D// : public CMesh<>
        {
        public:
            CRegularMesh3D();
            CRegularMesh3D(const std::string& file_name);
            CRegularMesh3D(const CRegularMesh3D&);
            // nx ny number of elements on x y
            CRegularMesh3D(const Point& p1, const Point& p2, const int nx, const int ny);
            CRegularMesh3D(const Point& p1, const Point& p2, const int nx, const int ny, const int px, const int py);
            CRegularMesh3D(const double x1, const double y1, const double x2, const double y2, const int nx, const int ny);
            CRegularMesh3D& operator=(const CRegularMesh3D& tr)
            {
                const int sz_el = (int)tr.m_elems.size();
                const int sz_pt = (int)tr.m_points.size();
                const int sz_bpt = (int)tr.m_basepoints.size();
                const int sz_bel = (int)tr.m_elemsbase.size();
                const int sz_ed = (int)tr.m_edges.size();
                const int sz_bed = (int)tr.m_edgesbase.size();
                m_elems.resize(sz_el);
                m_edges.resize(sz_ed);
                m_points.resize(sz_pt);
                m_basepoints.resize(sz_bpt);
                m_elemsbase.resize(sz_bel);
                m_edgesbase.resize(sz_bed);
                int i = 0;
                for (i = 0; i < sz_el; ++i)
                    m_elems[i] = tr.m_elems[i]->Clone();
                for (i = 0; i < sz_ed; ++i)
                    m_edges[i] = tr.m_edges[i]->Clone();
                for (i = 0; i < sz_pt; ++i)
                    m_points[i] = tr.m_points[i];
                for (i = 0; i < sz_bpt; ++i)
                    m_basepoints[i] = tr.m_basepoints[i];
                for (i = 0; i < sz_bel; ++i)
                    m_elemsbase[i] = tr.m_elemsbase[i]->Clone();
                for (i = 0; i < sz_bed; ++i)
                    m_edgesbase[i] = tr.m_edgesbase[i]->Clone();
                m_bnds.resize(tr.m_bnds.size());
                for (i = 0; i < m_bnds.size(); ++i)
                    m_bnds[i] = tr.m_bnds[i];
                m_offsets = tr.m_offsets;
                m_params = tr.m_params;
                m_order = tr.m_order;
                m_inodes = tr.m_inodes;
                return *this;
            }
            CRegularMesh3D*           Clone() const
            {
                return new CRegularMesh3D(*this);
            };
            const unsigned int               GetNumberOfElements() const;
            const unsigned int               GetNumberOfNodes() const;
            const int                        GetNumberOfINodes() const;
            const unsigned int               GetNumberOfBoundaries() const;
            const int                        FindElement(const Point&) const;
```

```
            const Point                       GetNode(const unsigned int) const;
            const CElement<>*             GetElement(const unsigned int) const;
            const CElement<>*             GetBoundary(const unsigned int) const;
            const double                    getMinSize() const { return 0.; };
            const double                    getSolution(const unsigned int element, const unsigned int node) const;
            const int                       updateSolution(const unsigned int element, const unsigned int node, const double value);
            const std::vector<double>       getSolution() const;
            const int                       updateSolution(const std::vector<double>&);
            const int                       updateSolution(const unsigned int element, const unsigned int node, CSolution* value);
            const double                    getParameter(Parameters, const unsigned int, const Point&) const;
            const double                    getParameter(Parameters, const unsigned int, const int) const;
            const int                       setParameter(Parameters, const double, const unsigned int);
            const int                       setParameter(const CParameter&, const unsigned int type);
            const int                       updateSolution(const unsigned int node, const double value);
            const int                       refine_hx();
            const int                       refine_hy();
            const int                       refine_h();
            const int                       refine_p();
            const int                       refine_hp();
            const int                       interpolate(const int node) const;
            ~CRegularMesh3D();
        private:
            std::vector<CElement<>*>        m_elems;
            std::vector<CElement<>*>        m_edges;
            std::vector<Point>              m_points;
            std::vector<Point>              m_basepoints;
            std::vector<CElement<>*>        m_elemsbase;
            std::vector<CElement<>*>        m_edgesbase;
            std::map<int, CParameter>       m_params;
            std::vector<int>                m_offsets;
            std::vector<int>                m_bnds;
            const double                    CompSquare(const Point& p1, const Point& p2, const Point& p3) const;
            int                             m_order;
            int                             m_inodes;
        public:
            auto                            GetElements() -> decltype(m_elems) { return m_elems; };
            auto                            GetBoundary() -> decltype(m_edges) { return m_edges; };
        };
    }
}
#endif /* CORENC_MESH_RegularMesh3D_h */

```

7.59 CoreNCFEM/Grids/TriangularMesh.cpp File Reference

```
#include <stdio.h>
#include "TriangularMesh.h"
#include "../FiniteElements/Triangle.h"
#include "../FiniteElements/Edge.h"
#include <iostream>
#include <algorithm>
#include <numeric>
#include <random>
```

Functions

- template< class T >
  vector< size_t > sort_indexes (const vector< T > &v)

7.59.1 Function Documentation

7.59.1.1 sort_indexes()

```
template<class T >
vector< size_t > sort_indexes (
            const vector< T > & v )
```

7.60 CoreNCFEM/Grids/TriangularMesh.h File Reference

```
#include "../Mesh.h"
#include "../Point.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <map>
#include "../Parameter.h"
```

Classes

- class corenc::Mesh::CTriangularMesh

Namespaces

- namespace corenc
- namespace corenc::Mesh

7.61 TriangularMesh.h

Go to the documentation of this file.

```
#ifndef CORENC_MESH_TriangularMesh_h
#define CORENC_MESH_TriangularMesh_h
#include "../Mesh.h"

#include "../Point.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <map>
#include "../Parameter.h"
namespace corenc
{
    namespace Mesh
    {
        class CTriangularMesh : public CMesh<>
        {
        public:
            CTriangularMesh();
            CTriangularMesh(const std::string& file_name);
            CTriangularMesh(const CTriangularMesh&);
            CTriangularMesh(const Point& p1, const Point& p2, const int nx, const int ny);
            CTriangularMesh& operator=(const CTriangularMesh& tr)
            {
                const int sz_el = tr.m_elems.size();
                const int sz_pt = tr.m_points.size();
                const int sz_bpt = tr.m_basepoints.size();
                const int sz_bel = tr.m_elemsbase.size();
                const int sz_ed = tr.m_edges.size();
                const int sz_bed = tr.m_edgesbase.size();
```

```
            m_elems.resize(sz_el);
            m_edges.resize(sz_ed);
            m_points.resize(sz_pt);
            m_basepoints.resize(sz_bpt);
            m_elemsbase.resize(sz_bel);
            m_edgesbase.resize(sz_bed);
            int i = 0;
            for (i = 0; i < sz_el; ++i)
                m_elems[i] = tr.m_elems[i]->Clone();
            for (i = 0; i < sz_ed; ++i)
                m_edges[i] = tr.m_edges[i]->Clone();
            for (i = 0; i < sz_pt; ++i)
                m_points[i] = tr.m_points[i];
            for (i = 0; i < sz_bpt; ++i)
                m_basepoints[i] = tr.m_basepoints[i];
            for (i = 0; i < sz_bel; ++i)
                m_elemsbase[i] = tr.m_elemsbase[i]->Clone();
            for (i = 0; i < sz_bed; ++i)
                m_edgesbase[i] = tr.m_edgesbase[i]->Clone();
            m_params = tr.m_params;
            m_bnds.resize(tr.m_bnds.size());
            for (i = 0; i < m_bnds.size(); ++i)
                m_bnds[i] = tr.m_bnds[i];
            m_order = tr.m_order;
            m_offsets = tr.m_offsets;
            return *this;
        }
        CTriangularMesh*         Clone() const
        {
            return new CTriangularMesh(*this);
        };
        //const bool operator<(const CTriangularMesh& mesh) const
        //{
        //  if(m_points.size() < mesh.m_points.size())
        //      return true;
        //  return false;
        //}
        const unsigned int                  GetNumberOfElements() const;
        const unsigned int                  GetNumberOfNodes() const;
        const unsigned int                  GetNumberOfBoundaries() const;
        const int                           FindElement(const Point&) const;
        const Point                            GetNode(const unsigned int) const;
        const CElement<>*                     GetElement(const unsigned int) const;
        const CElement<>*                 GetBoundary(const unsigned int) const;
        const double                        getMinSize() const { return 0.; };
        const double                        getSolution(const unsigned int element, const unsigned int node) const;
        const int                           updateSolution(const unsigned int element, const unsigned int node, const double value);
        const std::vector<double>           getSolution() const;
        const int                           updateSolution(const std::vector<double>&);
        const int                           updateSolution(const unsigned int element, const unsigned int node, CSolution* value);
        const double                        getParameter(Parameters, const unsigned int, const Point&) const;
        const double                        getParameter(Parameters, const unsigned int, const int) const;
        const int                           setParameter(Parameters, const double, const unsigned int);
        const int                           setParameter(const CParameter&, const unsigned int type);
        const int                           updateSolution(const unsigned int node, const double value);
        const int                           refine_h();
        const int                           refine_p();
        const int                           refine_hp();
        const int                           set4thOrder();
        const int                           set2ndOrder();
        const int                           set3rdOrder();
        const int                           interpolate(const int node) const;
        const int                           GetNumberOfINodes() const;
        ~CTriangularMesh();
    private:
        std::vector<CElement<>*>            m_elems;
        std::vector<CElement<>*>            m_edges;
        std::vector<Point>                  m_points;
        std::vector<Point>                  m_basepoints;
        std::vector<CElement<>*>            m_elemsbase;
        std::vector<CElement<>*>            m_edgesbase;
        std::map<int, CParameter>           m_params;
        std::vector<int>                    m_offsets;
        std::vector<int>                    m_bnds;
        const double                        CompSquare(const Point& p1, const Point& p2, const Point& p3) const;
        void                                set3rdNodes();
        void                                set4thNodes_1();
        void                                set4thNodes_2();
        int                                 m_order;
```

```
        public:
            auto                                GetElements() -> decltype(m_elems) { return m_elems; };
            auto                                GetBoundary() -> decltype(m_edges) { return m_edges; };
        };
    }
}
#endif /* CORENC_MESH_TriangularMesh_h */

```

7.62 CoreNCFEM/Grids/TriangularMeshLinear.cpp File Reference

```
#include <stdio.h>
#include "TriangularMeshLinear.h"
#include "../FiniteElements/TriangleLinear.h"
#include "../FiniteElements/Edge.h"
#include <iostream>
#include <algorithm>
```

7.63 CoreNCFEM/Grids/TriangularMeshLinear.h File Reference

```
#include "../Mesh.h"
#include "../Point.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <map>
#include "../Parameter.h"
```

Classes

- class corenc::Mesh::CTriangularMeshLinear

Namespaces

- namespace corenc
- namespace corenc::Mesh

7.64 TriangularMeshLinear.h

Go to the documentation of this file.

```
#ifndef CORENC_MESH_TriangularMesh_h
#define CORENC_MESH_TriangularMesh_h
#include "../Mesh.h"

#include "../Point.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <map>
```

```
#include "../Parameter.h"
namespace corenc
{
    namespace Mesh
    {
        class CTriangularMeshLinear : public CMesh<>
        {
        public:
            CTriangularMeshLinear();
            CTriangularMeshLinear(const std::string& file_name);
            CTriangularMeshLinear(const CTriangularMeshLinear&);
            const unsigned int                  GetNumberOfElements() const;
            const unsigned int                  GetNumberOfNodes() const;
            const unsigned int                  GetNumberOfBoundaries() const;
            const int                           FindElement(const Point&) const;
            const Point                            GetNode(const unsigned int) const;
            const CElement<>*                     GetElement(const unsigned int) const;
            const CElement<>*                 GetBoundary(const unsigned int) const;
            const double                        getMinSize() const { return 0.; };
            const double                        getSolution(const unsigned int element, const unsigned int node) const;
            const int                           updateSolution(const unsigned int element, const unsigned int node, const double value);
            const std::vector<double>           getSolution() const;
            const int                           updateSolution(const std::vector<double>&);
            const int                           updateSolution(const unsigned int element, const unsigned int node, CSolution* value);
            const double                        getParameter(Parameters, const unsigned int, const Point&) const;
            const double                        getParameter(Parameters, const unsigned int, const int) const;
            const int                           setParameter(Parameters, const double, const unsigned int);
            const int                           setParameter(const CParameter&, const unsigned int type);
            const int                           updateSolution(const unsigned int node, const double value);
            const int                           refine_h();
            ~CTriangularMeshLinear();
        private:
            std::vector<CElement<>*>            m_elems;
            std::vector<CElement<>*>            m_edges;
            std::vector<Point>                  m_points;
            std::map<int, CParameter>           m_params;
        public:
            auto                                GetElements() -> decltype(m_elems) { return m_elems; };
            auto                                GetBoundary() -> decltype(m_edges) { return m_edges; };
        };
    }
}
#endif /* CORENC_MESH_TriangularMesh_h */
```

7.65 CoreNCFEM/Mesh.h File Reference

```
#include "FiniteElements/FiniteElement.h"
```

Classes

- class corenc::Mesh::CMesh< T >
- class corenc::Mesh::CMesh< bool >

Namespaces

- namespace corenc
- namespace corenc::Mesh

Enumerations

- enum corenc::Mesh::Meshes { corenc::Mesh::Mesh1D = 0 , corenc::Mesh::TriangularMesh = 1 , corenc::Mesh::TetrahedralMesh = 2 }

7.66 Mesh.h

Go to the documentation of this file.

```
#ifndef CORENC_MESH_Mesh_h
#define CORENC_MESH_Mesh_h
#include "FiniteElements/FiniteElement.h"
namespace corenc
{
    namespace Mesh
    {
        enum Meshes
        {
             Mesh1D = 0,
             TriangularMesh = 1,
             TetrahedralMesh = 2
         };
         template<class T = bool>
         class CMesh;
         template<class T>
         class CMesh
         {
         public:
             CMesh() {}
             virtual ~CMesh() {}
             virtual const unsigned int            GetNumberOfNodes() const = 0;
             virtual const unsigned int            GetNumberOfElements() const = 0;
             virtual const int                     FindElement(const Point&) const = 0;
             virtual const unsigned int            GetNumberOfBoundaries() const = 0;
             virtual const CElement<T>*             GetElement(const unsigned int) const = 0;
             virtual const CElement<T>*             GetBoundary(const unsigned int) const = 0;
             virtual const Point                       GetNode(const unsigned int) const = 0;
             virtual const double                  getSolution(const unsigned int element, const unsigned int node) const = 0;
             virtual const int                     updateSolution(const unsigned int element, const unsigned int node, const double value) = 0;
             virtual const std::vector<double>     getSolution() const = 0;
             virtual const int                     updateSolution(const std::vector<double>&) = 0;
             //virtual const int                   updateSolution(const std::vector<CSolution*>&) = 0;
             virtual const int                     updateSolution(const unsigned int element, const unsigned int node, CSolution* value) = 0;
             virtual const double                  getParameter(Parameters, const unsigned int, const Point&) const = 0;
             virtual const double                  getParameter(Parameters, const unsigned int, const int) const = 0;
             virtual const int                     setParameter(Parameters, const double, const unsigned int) = 0;
             virtual const double                  getMinSize() const = 0;
             virtual const int                     updateSolution(const unsigned int node, const double value) = 0;
         };
         template<>
         class CMesh<bool>
         {
         public:
             CMesh() {}
             virtual ~CMesh() {}
             virtual const unsigned int            GetNumberOfNodes() const = 0;
             virtual const unsigned int            GetNumberOfElements() const = 0;
             virtual const int                     FindElement(const Point&) const = 0;
             virtual const unsigned int            GetNumberOfBoundaries() const = 0;
             virtual const CElement<>*           GetElement(const unsigned int) const = 0;
             virtual const CElement<>*           GetBoundary(const unsigned int) const = 0;
             virtual const Point                       GetNode(const unsigned int) const = 0;
             virtual const double                  getSolution(const unsigned int element, const unsigned int node) const = 0;
             virtual const int                     updateSolution(const unsigned int element, const unsigned int node, const double value) = 0;
             virtual const std::vector<double>     getSolution() const = 0;
             virtual const int                     updateSolution(const std::vector<double>&) = 0;
             //virtual const int                   updateSolution(const std::vector<CSolution*>&) = 0;
             virtual const int                     updateSolution(const unsigned int element, const unsigned int node, CSolution* value) = 0;
             virtual const double                  getParameter(Parameters, const unsigned int, const Point& p) const = 0;
```

```
            virtual const double                    getParameter(Parameters, const unsigned int, const
      int) const = 0;
            virtual const int                       setParameter(Parameters, const double, const unsigned
      int) = 0;
            virtual const double                    getMinSize() const = 0;
        };
    }
}

#endif /* CORENC_MESH_Mesh_h */
```

7.67 CoreNCFEM/Methods/CSMethod.h File Reference

Classes

- class Methods::CSMethod

Namespaces

- namespace Methods

7.68 CSMethod.h

Go to the documentation of this file.

```
#ifndef CORENCFEM_METHODS_CSMethod_h
#define CORENCFEM_METHODS_CSMethod_h

namespace Methods
{
    class CSMethod
    {
    public:
        CSMethod() {}
         virtual ~CSMethod() {}
     };
}

#endif /* CORENCFEM_METHODS_CSMethod_h */
```

7.69 CoreNCFEM/Methods/dg_flux.h File Reference

Namespaces

- namespace corenc
- namespace corenc::method

Enumerations

- enum class corenc::method::DGFlux {
  corenc::method::EIP , corenc::method::EBaumannOden , corenc::method::EBaumannOdenIP , corenc::method::ENIPG
  ,
  corenc::method::EUpwind , corenc::method::ECentral , corenc::method::ELaxFriedrichs , corenc::method::IIP
  ,
  corenc::method::IBaumannOden , corenc::method::IBaumannOdenIP , corenc::method::INIPG , corenc::method::IUpwind
  ,
  corenc::method::ICentral , corenc::method::ILaxFriedrichs , corenc::method::CUSTOM , corenc::method::NOFLUX
  }

7.70 dg_flux.h

Go to the documentation of this file.

```
#ifndef CORENC_METHOD_DG_FLUX_H_
#define CORENC_METHOD_DG_FLUX_H_
namespace corenc
{
    namespace method
    {
        enum class DGFlux
        {
            EIP,
             EBaumannOden,
             EBaumannOdenIP,
             ENIPG,
             EUpwind,
             ECentral,
             ELaxFriedrichs,
             IIP,
             IBaumannOden,
             IBaumannOdenIP,
             INIPG,
             IUpwind,
             ICentral,
             ILaxFriedrichs,
             CUSTOM,
             NOFLUX,
         };

     }
 }
 #endif // !CORENC_METHOD_DG_FLUX_H_
```

7.71 CoreNCFEM/Methods/DGMethod.h File Reference

```
#include <functional>
#include <set>
#include "../Point.h"
#include "../Parameter.h"
#include "CSMethod.h"
#include <memory>
#include <cmath>
#include <map>
#include <algorithm>
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
```

Classes

- class corenc::method::CDGMethod< Type >
- class corenc::method::DGMethod< Problem, Grid, Matrix >

Namespaces

- namespace corenc
- namespace corenc::Mesh
- namespace corenc::method

7.72 DGMethod.h

Go to the documentation of this file.

```
#ifndef DGMethod_H
#define DGMethod_H

// DGMethod.h describes an abstract interface and functions for a DG method with zero Dirichlet boundaries
#include <functional>
#include <set>
#include "../Point.h"
#include "../Parameter.h"
#include "CSMethod.h"
#include <memory>
#include <cmath>
#include <map>
#include <algorithm>
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
namespace corenc
{
    namespace Mesh
    {
        class Point;
    }
    namespace method
    {
        // class Type = Type of the solution, for ex vector or double, or even more specific


        template<class Type>
        class CDGMethod
        {
        public:
            CDGMethod() {};
            virtual ~CDGMethod() {};
            virtual const int                           Assemble() = 0;
            virtual const Type                          GetSolution(const std::vector<double>& point) const = 0;
            virtual const std::vector<Type>             GetSolution() const = 0;
            virtual const Type                          GetMaxSolution() const = 0;
            virtual const Type                          GetMinSolution() const = 0;
        };

        template<class Problem, class Grid, class Matrix>
        class DGMethod
        {
        public:
            DGMethod() :
                m_problem{nullptr},
                m_Grid{nullptr},
                m_GlobalMatrix{nullptr},
                m_RightMatrix{nullptr},
                m_rhsvector{nullptr}
            {}
            DGMethod(
                Problem* p,
                Grid* g,
                Matrix* m,
                std::vector<double>* rhs):
                m_problem{ p },
                m_Grid{ g->Clone() },
                m_GlobalMatrix{ m },
                m_N{ g->GetNumberOfElements() },
                m_Ns{ g->GetNumberOfBoundaries() },
                m_rhsvector{ rhs }{
                //GeneratePortrait();
            }
            DGMethod(
                Problem* p,
                Grid* g,
                Matrix* m,
                Matrix* rm,
                std::vector<double>* rhs):
                m_problem{ p },
                m_Grid{ g->Clone() },
                m_GlobalMatrix{ m },
                m_RightMatrix{ rm },
                m_N{ g->GetNumberOfElements() },
                m_Ns{ g->GetNumberOfBoundaries() },
                m_rhsvector{ rhs }{
                //GeneratePortrait();
            }
            DGMethod(const std::shared_ptr<Grid>& grid) :m_Grid{ grid->Clone() } {}
```

```
        DGMethod(Grid* grid) :m_Grid{ grid->Clone() } {}
        DGMethod(const DGMethod& meth) :
            m_Grid{ meth.m_Grid->Clone() },
            //m_GlobalMatrix{ meth.m_GlobalMatrix->Clone() },
            //m_rhsvector{ meth.m_rhsvector },
            //m_problem{ meth.m_problem },
            m_time{ meth.m_time },
            //m_solution{ meth.m_solution },
            m_size{ meth.m_size },
            m_N{ meth.m_N },
            m_Ns{ meth.m_Ns },
            m_nums{ meth.m_nums }
        {};
        void                        Discretization();
        const double                GetValue(const Mesh::Point&) const;
        const double                GetValue(const Mesh::Point&, const std::vector<double>& vec) const;
        const double                GetValue(const Mesh::Point&, const std::vector<double>& vec, const int num) const;
        //const Mesh::Point         GetGradValue(const Mesh::Point&, const std::vector<double>& vec) const;
         //const Mesh::Point         GetLambdaGrad(const Mesh::Point&, const std::vector<double>& vec) const;
         const double                GetEffective(const std::vector<double>& vec) const;
         void                        ProjectSolution(std::vector<double>&, std::function<const double(const Mesh::Point&, const std::vector<double>&, const int)> GetValue, std::vector<double>& sol);
         void                        ProjectSolution(std::vector<double>&, std::function<const double(const Mesh::Point&, const std::vector<double>&)> GetValue, std::vector<double>& sol, const int);
         void                        LoadSolution(const std::vector<double>& vec);
         const std::vector<double>   SetSolution(const int sol, const int liq, const double, const double, const double);
         void                        GetSolution(std::vector<double>& vec);
         void                        Rediscretization(const std::shared_ptr<Grid>&);
         void                        Rediscretization();
         void                        SetTimeStep(const double& step) { m_step = step; m_time = step; }
         Matrix*                     GetGlobalMatrix() const;
         Grid*                       GetMesh() { return m_Grid; }
         const std::vector<double>   GetRightVector() const;
         void                        OutDatFormat(const Mesh::Point& min, const Mesh::Point& max, const std::string& file_name, const std::vector<double>& vec) const;
         void                        OutMeshFormat(const std::string& file_name, const std::vector<double>& vec);
         void                        OutMeshTimeFormat(const std::string& file_name, const std::vector<double>& vec);
         static const double         GetSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p);
         static const double         GetSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p, const int nfem);
         static const Mesh::Point GetGradSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p);
         static const Mesh::Point GetGradSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p, const int n);
         ~DGMethod();
     private:
         void                        GeneratePortrait();
         void                        AssemblGlobal();
         void                        MainConditions();
         void                        SecondConditions();
         void                        ThirdConditions();
         void                        StefanConditions();
         void                        ApplySources();
         const int                   AssembleLocalMatrix(const int);
         const int                   AssembleIDUDVMatrix(const int);
         const int                   AssembleIDUVMatrix(const int);
         const int                   AssembleIUDVMatrix(const int);
         const int                   AssembleRUVMatrix(const int);
         const int                   AssembleSUPGMatrix(const int);
         const int                   AssembleLocalMatrix(const int, const int);
         const int                   AssembleInter();
         Grid*                       m_Grid = nullptr;
         Matrix*                     m_GlobalMatrix = nullptr;
         Matrix*                 m_RightMatrix = nullptr;
         Problem*                    m_problem = nullptr;
         std::vector<double>         m_solution;
         std::vector<double>*        m_rhsvector;
         unsigned int                m_size;
         double                      m_step{ 0.1 };
         double                      m_time{ 0.1 };
         unsigned int                m_N;
         unsigned int                m_Ns;
         std::vector<unsigned int>   m_nums;
         // interpolation nodes
         std::vector<std::vector<int>»   m_inums;

```

```
        };

        template<class Problem, class Grid, class Matrix>
        void DGMethod<Problem, Grid, Matrix>::Discretization()
        {
            GeneratePortrait();
            AssemblGlobal();
            AssembleInter();
            //ApplySources();
            //SecondConditions();
            //ThirdConditions();
            MainConditions();
            //StefanConditions();
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethod<Problem, Grid, Matrix>::GeneratePortrait()
        {
            const auto& el = m_Grid->GetElement(0);
            int order = m_Grid->GetElement(0)->GetDoFs();
            std::vector<std::set<unsigned int» temp;
            //m_Ns = m_Grid->GetNumberOfINodes();
            m_Ns = m_Grid->GetNumberOfBoundaries();
            m_N = m_Grid->GetNumberOfElements();
            //temp.resize(m_Grid->GetNumberOfINodes());
            unsigned i, j, k;
            m_nums.resize(m_N);
            m_inums.resize(m_N);
            int size;
            m_size = 0;
            std::cout « "nums" « std::endl;
            for (k = 0; k < m_N; ++k)
            {
                const auto& elem{ m_Grid->GetElement(k) };
                size = 0;
                m_inums[k].resize(order);
                for (i = 0; i < order; ++i)
                {
                    {
                        m_inums[k][i] = size;
                        ++size;
                    }
                }
                m_nums[k] = m_size;
                m_size += size;
                std::cout « k « "\t" « m_nums[k] « std::endl;
            }
            int sz = m_Ns;
            int nk, ne;
            int sizej = 0;
            int sizei = 0;
            temp.resize(m_size);
            for (k = 0; k < sz; ++k)
            {
                auto bound = m_Grid->GetBoundary(k);
                nk = bound->GetNeighbour(0);
                ne = bound->GetNeighbour(1);
                std::cout « nk « ne « std::endl;
                sizei = 0;
                sizej = 0;
                if (ne != -1)
                {
                    auto elemk = m_Grid->GetElement(nk);
                    auto eleme = m_Grid->GetElement(ne);
                    size = 0;
                    for (i = 0; i < order; ++i)
                    {
                            for (j = i + 1; j < order; ++j)
                            {
                                {
                                    temp[m_nums[nk] + m_inums[nk][j]].insert(m_nums[nk] + m_inums[nk][i]);
                                }
                            }
                    }
                    for (i = 0; i < order; ++i)
                    {
                            for (j = 0; j < order; ++j)
                            {
                                int jnode = m_Grid->interpolate(eleme->GetNode(j));
                                    temp[m_nums[ne] + m_inums[ne][j]].insert(m_nums[nk] + m_inums[nk][i]);
                            }
                    }
                }
                else
                {
                    sizei = 0;
```

```
                    sizej = 0;
                    auto elemk = m_Grid->GetElement(nk);
                    size = 0;
                    for (i = 0; i < order; ++i)
                    {
                            for (j = i + 1; j < order; ++j)
                            {
                                    temp[m_nums[nk] + m_inums[nk][j]].insert(m_nums[nk] + m_inums[nk][i]);
                                    //temp[m_nums[nk] + sizej].insert(m_nums[nk] + sizei);
                            }
                    }
                }
            }
            if(m_problem->findTerm(Terms::RUV))
                m_RightMatrix->Create(temp.size(), temp);

      //      for (auto & it : temp)
    //        {
  //              for (auto& it2 : it)
//                  std::cout « it2 « "\t";
              //  std::cout « std::endl;
            //}
            //m_GlobalMatrix = std::shared_ptr<Matrix>(new Matrix(m_Grid->GetNumberOfNodes(), temp));
            //m_rhsvector.resize(m_Grid->GetNumberOfNodes());
            //std::cout « temp.size() « std::endl;
            m_GlobalMatrix->Create(temp.size(), temp);
            m_rhsvector->resize(temp.size());
            //m_solution.resize(m_Grid->GetNumberOfNodes());
            //for (int l = 0; l < m_Grid->GetNumberOfNodes(); ++l)
            //  m_solution[l] = 20;
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethod<Problem, Grid, Matrix>::AssemblGlobal()
        {
            int l;
            //std::vector<std::future<int» futures;
            int i, j, k, nodes;
            double mij;
            const int terms{ (int)m_problem->getNumberOfTerms() };
            for (k = 0; k < terms; ++k)
            {
                switch (m_problem->getTerm(k))
                {
                    case Terms::IDUDV:
                        for (l = 0; l < m_N; ++l)
                        {
                            AssembleIDUDVMatrix(l);
                        }
                        break;
                    case Terms::IDUV:
                        for (l = 0; l < m_N; ++l)
                            AssembleIDUVMatrix(l);
                        break;
                    case Terms::IUDV:
                        for (l = 0; l < m_N; ++l)
                            AssembleIUDVMatrix(l);
                        break;
                    case Terms::SUPG:
                        for (l = 0; l < m_N; ++l)
                            AssembleSUPGMatrix(l);
                        break;
                    case Terms::RUV:
                        for (l = 0; l < m_N; ++l)
                            AssembleRUVMatrix(l);
                        break;
                    default:
                        break;
                }
            }
            //for (l = 0; l < m_N; ++l)
                //futures.push_back(async(&DGMethod<Problem, Grid, Matrix>::AssembleLocalMatrix, this, l));
             //   AssembleLocalMatrix(l, 0);
            //for (auto &it : futures)
            //it.get();
        }

        template<class Problem, class Grid, class Matrix>
        const int DGMethod<Problem, Grid, Matrix>::AssembleIDUDVMatrix(const int l)
        {
            int i, j, k, nodes;
            double mij;
            const auto& elem{ m_Grid->GetElement(l) };
            const int dofs{ (int)elem->GetDoFs() };
            const int terms{ (int)m_problem->getNumberOfTerms() };
            nodes = elem->GetNumberOfNodes();
```

```
            std::vector<Mesh::Point> points(nodes);
            for (i = 0; i < nodes; ++i)
                points[i] = m_Grid->GetNode(elem->GetNode(i));
            int sizei = 0, sizej = 0;
            for (i = 0; i < (int)dofs; ++i)
            {
                for (j = 0; j < (int)dofs; ++j)
                {
                    auto M = [&](const Mesh::Point& p)
                    {
                        //auto m = elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                        return m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p) * elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                    };
                    //mij = m_Grid->getParameter(Parameters::DIFFUSION, l, j) * elem->Integrate(M, points);
                    mij = elem->Integrate(M, points);
                    //m_GlobalMatrix->AddElement(inode, jnode, mij);
                    m_GlobalMatrix->AddElement(m_nums[l] + i, m_nums[l] + j, mij);
                }
            }
            return 0;
        }

        template<class Problem, class Grid, class Matrix>
        const int DGMethod<Problem, Grid, Matrix>::AssembleIDUVMatrix(const int l)
        {
            int i, j, k, nodes;
            double mij;
            const auto& elem{ m_Grid->GetElement(l) };
            const int dofs{ (int)elem->GetDoFs() };
            const int terms{ (int)m_problem->getNumberOfTerms() };
            nodes = elem->GetNumberOfNodes();
            std::vector<Mesh::Point> points(nodes);
            for (i = 0; i < nodes; ++i)
                points[i] = m_Grid->GetNode(elem->GetNode(i));
            int sizei = 0, sizej = 0;
            for (i = 0; i < (int)dofs; ++i)
            {
                for (j = 0; j < (int)dofs; ++j)
                {
                    auto M = [&](const Mesh::Point& p)
                    {
                        return m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * elem->GetShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                    };
                    auto _mij = elem->Integrate(M, points);
                    //m_GlobalMatrix->AddElement(inode, jnode, _mij);
                    m_GlobalMatrix->AddElement(m_nums[l] + i, m_nums[l] + j, _mij);
                }
            }
            return 0;
        }

        template<class Problem, class Grid, class Matrix>
        const int DGMethod<Problem, Grid, Matrix>::AssembleIUDVMatrix(const int l)
        {
            int i, j, k, nodes;
            double mij;
            const auto& elem{ m_Grid->GetElement(l) };
            const int dofs{ (int)elem->GetDoFs() };
            const int terms{ (int)m_problem->getNumberOfTerms() };
            nodes = elem->GetNumberOfNodes();
            std::vector<Mesh::Point> points(nodes);
            for (i = 0; i < nodes; ++i)
                points[i] = m_Grid->GetNode(elem->GetNode(i));
            int sizei = 0, sizej = 0;
            for (i = 0; i < dofs; ++i)
            {
                for (j = 0; j < dofs; ++j)
                {
                    auto M = [&](const Mesh::Point& p)
                    {
                        return elem->GetGradShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                    };
                    //mij = m_CoarseGrid->getParameter(Parameters::ADVECTION, l, j) * m_flux(m_CoarseGrid->getSolution(l, j)) * elem->Integrate(M, points).x;
                    mij = elem->Integrate(M, points).x;
                    //m_GlobalMatrix->AddElement(inode, jnode, mij);
                    m_GlobalMatrix->AddElement(m_nums[l] + i, m_nums[l] + j, mij);
                }
            }
            return 0;
        }


        template<class Problem, class Grid, class Matrix>
```

```
    const int DGMethod<Problem, Grid, Matrix>::AssembleRUVMatrix(const int l)
    {
        int i, j, k, nodes;
        double mij;
        const auto& elem{ m_Grid->GetElement(l) };
        const int dofs{ (int)elem->GetDoFs() };
        const int terms{ (int)m_problem->getNumberOfTerms() };
        nodes = elem->GetNumberOfNodes();
        std::vector<Mesh::Point> points(nodes);
        for (i = 0; i < nodes; ++i)
            points[i] = m_Grid->GetNode(elem->GetNode(i));
        int sizei = 0, sizej = 0;
        for (i = 0; i < (int)dofs; ++i)
        {
            for (j = 0; j < (int)dofs; ++j)
            {
                auto M = [&](const Mesh::Point& p)
                {
                    double vel = sqrt(m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0));
                    double h = elem->GetMeasure();
                    double Pe = vel * h / 6. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                    double tau = 0.;
                    //double Pe = vel * h / 2. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);

                    if (Pe >= 1)
                        tau = h / 2. / vel;
                    else
                        tau = h * h / 12. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                    auto supg = tau * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * elem->GetGradShapeFunction(i, p) * elem->GetShapeFunction(j, p) * elem->GetShapeFunction(i, p);
                    return elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);// + supg;
                };
                mij = elem->Integrate(M, points);

                    m_RightMatrix->AddElement(m_nums[l] + i, m_nums[l] + j, mij);
            }
        }
        return 0;
    }

    template<class Problem, class Grid, class Matrix>
    const int DGMethod<Problem, Grid, Matrix>::AssembleSUPGMatrix(const int l)
    {
        int i, j, k, nodes;
        double mij;
        const auto& elem{ m_Grid->GetElement(l) };
        const int dofs{ (int)elem->GetDoFs() };
        const int terms{ (int)m_problem->getNumberOfTerms() };
        nodes = elem->GetNumberOfNodes();
        std::vector<Mesh::Point> points(nodes);
        for (i = 0; i < nodes; ++i)
            points[i] = m_Grid->GetNode(elem->GetNode(i));
        for (i = 0; i < (int)dofs; ++i)
        {
            for (j = 0; j < (int)dofs; ++j)
            {
                auto M = [&](const Mesh::Point& p)
                {
                    double vel = sqrt(m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0));
                    double h = elem->GetMeasure();
                    //double Pe = vel * h / 6. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                    double tau = 0.;
                    double Pe = vel * h / 2. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                    //double beta = h / 2. / vel * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                    //double beta = h / std::sqrt(3.) * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                    //double beta = h / 2. * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                    //double beta = h / 2. * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                    //beta = 0.;
                    //for (int ii = 0; ii < (int)dofs; ++ii)
                        //beta += m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * elem->GetGradShapeFunction(ii, p);
                    //return beta * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) *
                    //        elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                    if (Pe >= 1)
                        tau = h / 2. / vel;
                    else
                        tau = h * h / 12. / m_problem->get_parameter(Terms::IDUDV, l,
```

```
elem->GetType(), p);
                        //return 0.;
                        return tau * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) *
                            elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                    };

                    //double tau =
                    auto _mij = elem->Integrate(M, points);
                    m_GlobalMatrix->AddElement(m_nums[l] + i, m_nums[l] + j, _mij);
                }
            }
            return 0;
    }


    template<class Problem, class Grid, class Matrix>
    const int DGMethod<Problem, Grid, Matrix>::AssembleInter()
    {
        const auto mu = 1e6;
        for (int l = 0; l < m_Ns; ++l)
        {
            const auto& bound{ m_Grid->GetBoundary(l) };
            const auto& nk{ bound->GetNeighbour(0) };
            const auto& ne{ bound->GetNeighbour(1) };
            const auto& elemk{ m_Grid->GetElement(nk) };
            const auto& dofs{ bound->GetDoFs() };
            const auto& dofsk{ elemk->GetDoFs() };
            std::vector<Mesh::Point> points(dofs);
            for (int i = 0; i < dofs; ++i)
            {
                points[i] = m_Grid->GetNode(bound->GetNode(i));
            }
            if (ne < 0)
                continue;
            const auto& eleme{ m_Grid->GetElement(ne) };
            for (int i = 0; i < dofsk; ++i)
            {
                for (int j = 0; j < dofsk; ++j)
                {
                    auto Tkk = [&](const Mesh::Point& p)
                    {
                        auto kappa = m_problem->get_parameter(Terms::IDUDV, l, elemk->GetType(), p);
                        auto val1 = bound->GetNormal() * elemk->GetShapeFunction(j, p)  * elemk->GetGradShapeFunction(i, p);
                        auto val2 = bound->GetNormal() * elemk->GetShapeFunction(i, p)  * elemk->GetGradShapeFunction(j, p);

                        auto ip = bound->GetNormal() * bound->GetNormal() * elemk->GetShapeFunction(j, p) * elemk->GetShapeFunction(i, p);
                        return 0.5 * kappa * (val2 - val1) + mu * ip;
                    };
                    auto mj = bound->Integrate(Tkk, points);
                    std::cout « mj « std::endl;
                    m_GlobalMatrix->AddElement(m_nums[nk] + i, m_nums[nk] + j, mj);
                }
            }

            for (int i = 0; i < dofsk; ++i)
            {
                for (int j = 0; j < dofsk; ++j)
                {
                    auto Tkk = [&](const Mesh::Point& p)
                    {
                        auto kappa = m_problem->get_parameter(Terms::IDUDV, l, eleme->GetType(), p);
                        auto val1 = bound->GetNormal() * eleme->GetShapeFunction(j, p)  * elemk->GetGradShapeFunction(i, p);
                        auto val2 = bound->GetNormal() * elemk->GetShapeFunction(i, p)  * eleme->GetGradShapeFunction(j, p);

                        auto ip = bound->GetNormal() * bound->GetNormal() * eleme->GetShapeFunction(j, p) * elemk->GetShapeFunction(i, p);
                        return 0.5 * kappa * (val2 + val1) + mu * ip;
                    };
                    auto mj = bound->Integrate(Tkk, points);
                    m_GlobalMatrix->AddElement(m_nums[nk] + i, m_nums[ne] + j, mj);
                }
            }


            for (int i = 0; i < dofsk; ++i)
            {
                for (int j = 0; j < dofsk; ++j)
                {
                    auto Tkk = [&](const Mesh::Point& p)
                    {
                        auto kappa = m_problem->get_parameter(Terms::IDUDV, l, eleme->GetType(), p);
```

```
                        auto val1 = bound->GetNormal() * eleme->GetShapeFunction(j, p)  * eleme->GetGradShapeFunction(i, p);
                        auto val2 = bound->GetNormal() * eleme->GetShapeFunction(i, p)  * eleme->GetGradShapeFunction(j, p);

                        auto ip = bound->GetNormal() * bound->GetNormal() * eleme->GetShapeFunction(j, p) * eleme->GetShapeFunction(i, p);
                        return 0.5 * kappa * (val2 - val1) + mu * ip;
                    };
                    auto mj = bound->Integrate(Tkk, points);
                    m_GlobalMatrix->AddElement(m_nums[ne] + i, m_nums[ne] + j, mj);
                }
            }

            for (int i = 0; i < dofsk; ++i)
            {
                for (int j = 0; j < dofsk; ++j)
                {
                    auto Tkk = [&](const Mesh::Point& p)
                    {
                        auto kappa = m_problem->get_parameter(Terms::IDUDV, l, elemk->GetType(), p);
                        auto val1 = bound->GetNormal() * elemk->GetShapeFunction(j, p)  * eleme->GetGradShapeFunction(i, p);
                        auto val2 = bound->GetNormal() * eleme->GetShapeFunction(i, p)  * elemk->GetGradShapeFunction(j, p);

                        auto ip = bound->GetNormal() * bound->GetNormal() * elemk->GetShapeFunction(j, p) * eleme->GetShapeFunction(i, p);
                        return 0.5 * kappa * (val2 + val1) + mu * ip;
                    };
                    auto mj = bound->Integrate(Tkk, points);
                    m_GlobalMatrix->AddElement(m_nums[ne] + i, m_nums[nk] + j, mj);
                }
            }
        }
        return 0;
    }

    template<class Problem, class Grid, class Matrix>
    const int DGMethod<Problem, Grid, Matrix>::AssembleLocalMatrix(const int l, const int old)
    {
        int i, j, k, nodes;
        double mij;
        const auto& elem{ m_Grid->GetElement(l) };
        const int dofs{ (int)elem->GetDoFs() };
        const int terms{ (int)m_problem->getNumberOfTerms() };
        nodes = elem->GetNumberOfNodes();
        std::vector<Mesh::Point> points(nodes);
        for (i = 0; i < nodes; ++i)
            points[i] = m_Grid->GetNode(elem->GetNode(i));
        for (k = 0; k < terms; ++k)
        {
            switch (m_problem->getTerm(k))
            {
            case Terms::IUV:
                for (i = 0; i < (int)dofs; ++i)
                {
                    for (j = 0; j < (int)dofs; ++j)
                    {
                        auto M = [&](const Mesh::Point& p)
                        {
                            return m_problem->get_parameter(Terms::IUV, l, elem->GetType(), p) * elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                        };
                        mij = elem->Integrate(M, points);
                        auto inode = m_Grid->interpolate(elem->GetNode(i));
                        auto jnode = m_Grid->interpolate(elem->GetNode(j));
                        if (inode > -1 && jnode > -1)
                            m_GlobalMatrix->AddElement(inode, jnode, mij);
                    }
                }
                break;
            case Terms::IDUDV:
                for (i = 0; i < (int)dofs; ++i)
                {
                    for (j = 0; j < (int)dofs; ++j)
                    {
                        auto inode = m_Grid->interpolate(elem->GetNode(i));
                        auto jnode = m_Grid->interpolate(elem->GetNode(j));
                        if (inode == -1 || jnode == -1)
                            continue;
                        auto M = [&](const Mesh::Point& p)
                        {
                            //auto m = elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                            return m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p) * elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
```

```
                        };
                        //mij = m_Grid->getParameter(Parameters::DIFFUSION, l, j) * elem->Integrate(M, points);
                        mij = elem->Integrate(M, points);
                        m_GlobalMatrix->AddElement(inode, jnode, mij);
                    }
                }
                break;
            case Terms::IDUV:
                for (i = 0; i < (int)dofs; ++i)
                {
                    for (j = 0; j < (int)dofs; ++j)
                    {
                        auto inode = m_Grid->interpolate(elem->GetNode(i));
                        auto jnode = m_Grid->interpolate(elem->GetNode(j));
                        if (inode == -1 || jnode == -1)
                            continue;
                        auto M = [&](const Mesh::Point& p)
                        {
                            return m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * elem->GetShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                        };
                        auto _mij = elem->Integrate(M, points);
                        m_GlobalMatrix->AddElement(inode, jnode, _mij);
                    }
                }
                break;
            case Terms::IUDV:
                for (i = 0; i < dofs; ++i)
                {
                    for (j = 0; j < dofs; ++j)
                    {
                        auto inode = m_Grid->interpolate(elem->GetNode(i));
                        auto jnode = m_Grid->interpolate(elem->GetNode(j));
                        if (inode == -1 || jnode == -1)
                            continue;
                        auto M = [&](const Mesh::Point& p)
                        {
                            return elem->GetGradShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                        };
                        //mij = m_CoarseGrid->getParameter(Parameters::ADVECTION, l, j) * m_flux(m_CoarseGrid->getSolution(l, j)) * elem->Integrate(M, points).x;
                        mij = elem->Integrate(M, points).x;
                        m_GlobalMatrix->AddElement(inode, jnode, mij);
                    }
                }
                break;
            case Terms::EUV:
                for (i = 0; i < dofs; ++i)
                {
                    for (j = 0; j < dofs; ++j)
                    {
                        auto M = [&](const Mesh::Point& p)
                        {
                            return elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                        };
                        mij = elem->Integrate(M, points);
                        m_rhsvector->operator[](elem->GetNode(i)) += m_Grid->getParameter(Parameters::MASS, l, j) * m_Grid->getSolution(l, j) * mij;
                        //m_rhsvector->operator[](m_nums[l] + i) += m_CoarseGrid->getParameter(Parameters::MASS, l, points[j]) * elem->GetValue(j) * mij;
                    }
                }
                break;
            case Terms::EDUDV:
                for (i = 0; i < dofs; ++i)
                {
                    for (j = 0; j < dofs; ++j)
                    {
                        auto M = [&](const Mesh::Point& p)
                        {
                            return elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                        };
                        mij = elem->Integrate(M, points);
                        m_rhsvector->operator[](elem->GetNode(i)) += m_Grid->getParameter(Parameters::DIFFUSION, l, j) * m_Grid->getSolution(l, j) * mij;
                    }
                }
                break;
            case Terms::EDUV:
                for (i = 0; i < dofs; ++i)
                {
                    for (j = 0; j < dofs; ++j)
                    {
                        auto M = [&](const Mesh::Point& p)
                        {
```

```
                            return elem->GetShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                        };
                        mij = elem->Integrate(M, points).x;
                        m_rhsvector->operator[](elem->GetNode(i)) += m_Grid->getParameter(Parameters::ADVECTION, l, j) * mij;
                    }
                }
                break;
            case Terms::EUDV:
                for (i = 0; i < dofs; ++i)
                {
                    for (j = 0; j < dofs; ++j)
                    {
                        auto M = [&](const Mesh::Point& p)
                        {
                            return elem->GetGradShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                        };
                        mij = elem->Integrate(M, points).x;
                        m_rhsvector->operator[](elem->GetNode(i)) += m_Grid->getParameter(Parameters::ADVECTION, l, j) * mij;// *mij;
                    }
                }
                break;
            case Terms::EFV:
                for (i = 0; i < dofs; ++i)
                {
                    /*for (j = 0; j < dofs; ++j)
                    {
                        auto M = [&](const Mesh::Point& p)
                        {
                            return m_problem->get_parameter(Terms::EFV, elem->GetType(), l, j, p) * elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                        };
                        mij = elem->Integrate(M, points);
                        m_rhsvector->operator[](elem->GetNode(i)) += mij;
                    }*/
                    auto M = [&](const Mesh::Point& p)
                    {
                        return m_problem->get_parameter(Terms::EFV, elem->GetType(), l, i, p) * elem->GetShapeFunction(i, p);
                    };
                    mij = elem->Integrate(M, points);
                    m_rhsvector->operator[](elem->GetNode(i)) += mij;
                }
                break;
            case Terms::RUV:
                for (i = 0; i < (int)dofs; ++i)
                {
                    for (j = 0; j < (int)dofs; ++j)
                    {
                        auto M = [&](const Mesh::Point& p)
                        {
                            return elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                        };
                        mij = elem->Integrate(M, points);
                        auto inode = m_Grid->interpolate(elem->GetNode(i));
                        auto jnode = m_Grid->interpolate(elem->GetNode(j));
                        if (inode > -1 && jnode > -1)
                            m_RightMatrix->AddElement(inode, jnode, mij);
                    }
                }
                break;
            case Terms::SUPG:
                for (i = 0; i < (int)dofs; ++i)
                {
                    for (j = 0; j < (int)dofs; ++j)
                    {
                        auto inode = m_Grid->interpolate(elem->GetNode(i));
                        auto jnode = m_Grid->interpolate(elem->GetNode(j));
                        if (inode == -1 || jnode == -1)
                            continue;
                        auto M = [&](const Mesh::Point& p)
                        {
                            double vel = sqrt(m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0));
                            double h = elem->GetMeasure();
                            //double Pe = vel * h / 6. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                            double tau = 0.;
                            double Pe = vel * h / 2. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                            //double beta = h / 2. / vel * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                            double beta = h / std::sqrt(3.) * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                            //double beta = h / 2. * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
```

```
                                //beta = 0.;
                                //for (int ii = 0; ii < (int)dofs; ++ii)
                                    //beta += m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * elem->GetGradShapeFunction(ii, p);
                                return beta * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) *
                                        elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                                if (Pe >= 1)
                                    tau = h / 2. / vel;
                                else
                                    tau = h * h / 12. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                                //return 0.;
                                return tau * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) *
                                        elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                            };

                            //double tau =
                            auto _mij = elem->Integrate(M, points);
                            m_GlobalMatrix->AddElement(inode, jnode, _mij);
                        }
                    }
                    break;
                default:
                    break;
                }
            }
            return 0;
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethod<Problem, Grid, Matrix>::MainConditions()
        {
            double mu{ 1e8 };
            const auto n = m_problem->get_number_of_boundaries();
            const auto m = m_Grid->GetNumberOfBoundaries();
            for (int i = 0; i < n; ++i)
            {
                const auto& type = m_problem->get_boundary_type(i);
                for (int j = 0; j < m; ++j)
                {
                    const auto& row = m_Grid->GetBoundary(j);
                    if (row->GetType() == type)
                    {
                        const int dofs = (int)row->GetDoFs();
                        const int dofs2 = 2;
                        const auto& elem_num = row->GetNeighbour(0);
                        const auto& elem = m_Grid->GetElement(elem_num);
                        const int dofs_elem = elem->GetDoFs();
                        std::vector<Mesh::Point> points(dofs_elem);
                        std::vector<Mesh::Point> bpoints(dofs);
                        for (int k = 0; k < dofs_elem; ++k)
                            points[k] = m_Grid->GetNode(elem->GetNode(k));
                        for (int k = 0; k < dofs; ++k)
                            bpoints[k] = m_Grid->GetNode(row->GetNode(k));
                        for (int ii = 0; ii < dofs_elem; ++ii)
                        {
                            for (int jj = 0; jj < dofs_elem; ++jj)
                            {
                                auto M = [&](const Mesh::Point& p)
                                {
                                    return elem->GetShapeFunction(ii, p) * elem->GetShapeFunction(jj, p);// + supg;
                                };
                                auto mj = mu * row->Integrate(M, bpoints);
                                m_GlobalMatrix->AddElement(m_nums[elem_num] + ii, m_nums[elem_num] + jj, mj);
                            }
                            auto MM = [&](const Mesh::Point& p)
                            {
                                return elem->GetWeight(elem_num, points, [=](const Mesh::Point& p) { return m_problem->get_boundary_parameter(0, type, p); }) * elem->GetShapeFunction(ii, p);
                            };
                            auto mij = row->Integrate(MM, points);
                            std::cout « mij « std::endl;
                            m_rhsvector->operator[](m_nums[elem_num] + ii) += mij;
                        }
                        /*for (int k = 0; k < dofs; ++k)
                        {
                            int l = 0;
                            for (; l < dofs_elem; ++l)
                            {
                                if (elem->GetNode(l) == row->GetNode(k))
                                    break;
                            }
```

```

                        m_GlobalMatrix->NullRow(row->GetNode(k));
                        //m_GlobalMatrix->operator()(row->GetNode(k), row->GetNode(k)) *= mu;
                        //m_rhsvector->operator[](row->GetNode(k)) = m_problem->get_boundary_parameter(0, type, m_Grid->GetNode(row->GetNode(k)));
                        //m_rhsvector->operator[](row->GetNode(k)) = m_problem->get_boundary_parameter(0, type, elem_num, l, m_Grid->GetNode(row->GetNode(k)));
                        m_rhsvector->operator[](row->GetNode(k)) = elem->GetWeight(l, points, [=](const Mesh::Point& p) { return m_problem->get_boundary_parameter(0, type, p); });
                        if(m_problem->findTerm(Terms::RUV))
                            {
                                m_RightMatrix->NullRow(row->GetNode(k));
                                //m_RightMatrix->operator()(row->GetNode(k), row->GetNode(k)) *= mu;
                            }
                    }*/
                    /*for (int k = dofs2; k < dofs; ++k)
                    {
                        m_GlobalMatrix->NullRow(row->GetNode(k));
                        m_rhsvector->operator[](row->GetNode(k)) = 0;
                    }*/
                }
            }
        }
        /*for (auto bnd : m_Grid->GetBoundaryConditions())
        {
            if (get<0>(bnd.second) == 1)
                for (auto row : m_Grid->GetBoundary())
                {
                    if (bnd.first == row->GetType())
                    {
                        for (int i = 0; i < row->GetDoF(); ++i)
                        {
                            m_GlobalMatrix->NullRow(row->GetNodes(i));
                            m_rhsvector[row->GetNodes(i)] = get<1>(bnd.second)(m_Grid->GetNodes()[row->GetNodes(i)]);
                        }
                    }
                }
        }*/
    }
    template<class Problem, class Grid, class Matrix>
    void DGMethod<Problem, Grid, Matrix>::SecondConditions()
    {
        double theta = 0;
        int nfem;
        Mesh::Point temp[3];
        std::vector<int> local;
        for (auto bnd : m_Grid->GetBoundaryConditions())
        {
            //if (get<0>(bnd.second) == 2)
            {
                for (auto row : m_Grid->GetBoundary())
                {
                    if (bnd.first == row->GetType())
                    {
                        local.resize(0);
                        int dofs = row->GetDoF();
                        nfem = row->GetNumberOfElement(0);
                        auto elem = m_Grid->GetElements()[nfem];
                        //auto GetBasis = [&](int t, Point p){return elem->GetBasis(t, p); };
                        for (int j = 0; j < dofs; ++j)
                        {
                            temp[j] = m_Grid->GetNodes()[row->GetNodes(j)];
                            for (int i = 0; i < elem->GetDoF(); ++i)
                            {
                                if (row->GetNodes(j) == elem->GetNodes()[i])
                                {
                                    local.push_back(i);
                                    break;
                                }
                            }
                        }
                        for (int i = 0; i < dofs; ++i)
                        {
                            for (int j = 0; j < dofs; ++j)
                            {
                                //theta = get<1>(bnd.second)(m_Grid->GetNodes()[row->GetNodes(i)]);
                                theta = 0;
                                auto GetMass = [&](const Mesh::Point& p) {return elem->GetBasis(local[j], p) * elem->GetBasis(local[i], p); };
                                auto GetBBasis = [&](const Mesh::Point& p) {return row->GetBasis(j, p)*row->GetBasis(i, p); };
                                //if (i < 2 || j < 2)
                                m_rhsvector[row->GetNodes(i)] += theta * row->Integrate(GetMass, temp);

                                //if (i < 3 || j < 3)
```

```
                                    //  m_rhsvector[row[i + 1]] += theta * row->Integrate(GetBBasis, temp);
                                }
                            }
                        }
                    }
                }
            }
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethod<Problem, Grid, Matrix>::StefanConditions()
        {
            double dest{ 0. }, lat{ 0 };
            int nfem;
            Mesh::Point temp[3];
            std::vector<int> local;
            for (auto bnd : m_Grid->GetBoundaryConditions())
            {
                //if (get<0>(bnd.second) == 4)
                {
                    lat = 0;
                    //lat = get<2>(bnd.second);
                    for (auto row : m_Grid->GetBoundary())
                    {
                        if (bnd.first == row->GetType())
                        {
                            local.resize(0);
                            int dofs = row->GetDoF();
                            nfem = row->GetNumberOfElement(0);
                            auto elem = m_Grid->GetElements()[nfem];
                            //auto GetBasis = [&](int t, Point p){return elem->GetBasis(t, p); };
                            for (int j = 0; j < dofs; ++j)
                            {
                                temp[j] = m_Grid->GetNodes()[row->GetNodes(j)];
                                for (int i = 0; i < elem->GetDoF(); ++i)
                                {
                                    if (row->GetNodes(j) == elem->GetNodes()[i])
                                    {
                                        local.push_back(i);
                                        break;
                                    }
                                }
                            }
                            for (int i = 0; i < dofs; ++i)
                            {
                                for (int j = 0; j < dofs; ++j)
                                {
                                    dest = 0;
                                    //dest = get<1>(bnd.second)(m_Grid->GetNodes()[row->GetNodes(i)]);
                                     auto GetBBasis = [&](const Mesh::Point& p) {return row->GetBasis(j, p)*row->GetBasis(i, p); };
                                     //if (i < 2 || j < 2)
                                     m_rhsvector[row->GetNodes(i)] += dest * lat * row->Integrate(GetBBasis, temp);

                                     //if (i < 3 || j < 3)
                                     //  m_rhsvector[row[i + 1]] += theta * row->Integrate(GetBBasis, temp);
                                 }
                             }
                         }
                     }
                 }
             }
         }
         template<class Problem, class Grid, class Matrix>
         void DGMethod<Problem, Grid, Matrix>::ThirdConditions()
         {
             double param{ 0 }, beta{ 0 };
             int nfem;
             Mesh::Point temp[6];
             std::vector<int> local;
             auto fxy = [&](const Mesh::Point& p) {return (10 * p.y*m_time + m_time) / 10; };
             //auto fxy = [&](const Point& p){return 10 * p.y + 10 * m_time; };
             for (auto bnd : m_Grid->GetBoundaryConditions())
             {
                 //if (get<0>(bnd.second) == 3)
                 {

                     for (auto row : m_Grid->GetBoundary())
                     {
                         if (bnd.first == row->GetType())
                         {
                             local.resize(0);
                             int dofs = row->GetDoF();
                             nfem = row->GetNumberOfElement(0);
                             auto elem = m_Grid->GetElements()[nfem];
```

```
                            //auto GetBasis = [&](int t, Point p){return elem->GetBasis(t, p); };
                            auto order = elem->GetDoF();
                            for (int j = 0; j < dofs; ++j)
                            {
                                temp[j] = m_Grid->GetNodes()[row->GetNodes(j)];
                                for (int i = 0; i < order; ++i)
                                {
                                    if (row->GetNodes(j) == elem->GetNodes()[i])
                                    {
                                        local.push_back(i);
                                        break;
                                    }
                                }
                            }
                            double val{ 0 };
                            for (int i = 0; i < dofs; ++i)
                            {
                                for (int j = 0; j < dofs; ++j)
                                {
                                    param = 0;
                                    beta = 0;
                                    //beta = get<2>(bnd.second);
                                    //param = get<1>(bnd.second)(m_Grid->GetNodes()[row->GetNodes(i)]);
                                    //param = fxy(temp[j]);
                                    auto GetBBasis = [&](const Mesh::Point& p) {return elem->GetBasis(local[j], p)*elem->GetBasis(local[i], p); };
                                    //val = row->GetElement(GetBBasis, temp);
                                    val = row->Integrate(GetBBasis, temp);
                                    m_GlobalMatrix->operator()(row->GetNodes(i), row->GetNodes(j)) += beta * val;
                                    m_rhsvector[row->GetNodes(i)] += beta * param * val;
                                }
                            }
                        }
                    }
                }
            }
        }
        template<class Problem, class Grid, class Matrix>
        Matrix* DGMethod<Problem, Grid, Matrix>::GetGlobalMatrix() const
        {
            return m_GlobalMatrix;
        }
        template<class Problem, class Grid, class Matrix>
        const double DGMethod<Problem, Grid, Matrix>::GetValue(const Mesh::Point& p) const
        {
            if (!m_solution.size())
                return -1;
            double val = 0;
            int nfem = -1;
            nfem = m_Grid->FindElement(p);
            if (nfem == -1)
                return -1;
            auto elem = m_Grid->GetElements()[nfem];
            for (int i = 0; i < elem->GetDoF(); ++i)
                val += m_solution[elem->GetNodes()[i]] * elem->GetBasis(i, p);
            return val;
        }
        template<class Problem, class Grid, class Matrix>
        const double DGMethod<Problem, Grid, Matrix>::GetValue(const Mesh::Point& p, const std::vector<double>& vec) const
        {
            if (!vec.size())
                return -1;
            double val{ 0 };
            int nfem{ -1 };
            nfem = m_Grid->FindElement(p);
            if (nfem == -1)
                return -1;
            auto elem = m_Grid->GetElements()[nfem];
            for (int i = 0; i < elem->GetDoFs(); ++i)
                val += vec[elem->GetNode(i)] * elem->GetShapeFunction(i, p);
            return val;
        }
        template<class Problem, class Grid, class Matrix>
        const double DGMethod<Problem, Grid, Matrix>::GetValue(const Mesh::Point& p, const std::vector<double>& vec, const int num) const
        {
            if (!vec.size() || num < 0)
                return -1;
            double val{ 0 };
            auto elem = m_Grid->GetElements()[num];
            for (int i = 0; i < elem->GetDoF(); ++i)
                val += vec[elem->GetNodes()[i]] * elem->GetBasis(i, p);
            return val;
        }
        //template<class Problem, class Grid, class Matrix>
```

```
    //const Mesh::Point DGMethod<Problem, Grid, Matrix>::GetGradValue(const Mesh::Point& p, const std::vector<double>& vec) const
    //{
    //  Mesh::Point val{ 0, 0 };
    //  int nfem{ -1 };
    //  nfem = m_Grid->FindElement(p);
    //  if (nfem == -1)
    //      return val;
    //  auto elem = m_Grid->GetElements()[nfem];
    //  for (int i = 0; i < elem->GetDoF(); ++i)
    //  {
    //      val.x += vec[elem->GetNodes()[i]] * elem->GetGradBasis(i, p).x;
    //      val.y += vec[elem->GetNodes()[i]] * elem->GetGradBasis(i, p).y;
    //      val.z += vec[elem->GetNodes()[i]] * elem->GetGradBasis(i, p).z;
    //  }
    //  return val;
    //}
    template<class Problem, class Grid, class Matrix>
    const double DGMethod<Problem, Grid, Matrix>::GetEffective(const std::vector<double>& vec) const
    {
        double sum = 0;
        //std::vector<int> dofs;
        //Mesh::Point points[10];
        //for (int i = 0; i < m_Grid->GetElements().size(); ++i)
        //{
            //auto mb = [&](const Mesh::Point& b) {return GetGradValue(b, vec)*GetGradValue(b, vec); };
            //dofs.resize(0);
            //auto elem = m_Grid->GetElements()[i];
            //int order = elem->GetDoF();
            //double diff = std::get<0>(m_Grid->GetDiffusion().find(elem->GetType())->second);
            //for (int j = 0; j < order; ++j)
            //{
                //dofs.push_back(elem->GetNodes()[j]);
                //points[j] = m_Grid->GetNodes()[dofs[j]];
            //}
            //sum += diff * elem->Integrate(mb, points);
        //}
        //std::cout « "Effect (local): " « sum « std::endl;
        //std::cout « "Effect (local) sqrt: " « sqrt(sum) « std::endl;
        return sum;
    }
    //template<class Problem, class Grid, class Matrix>
    //const Mesh::Point DGMethod<Problem, Grid, Matrix>::GetLambdaGrad(const Mesh::Point& p, const std::vector<double>& vec) const
    //{
    //  Mesh::Point val{ 0, 0, 0 };
    //  //double val{ 0 };
    //  double diff{ 0 };
    //  Mesh::Point temp{ 0, 0, 0 };
    //  int nfem{ -1 };
    //  nfem = m_Grid->FindElement(p);
    //  if (nfem == -1)
    //      return val;
    //  auto elem = m_Grid->GetElements()[nfem];
    //  diff = std::get<0>(m_Grid->GetDiffusion().find(elem->GetType())->second);
    //  for (int i = 0; i < elem->GetDoF(); ++i)
    //  {
    //      //val += elem->GetGradBasis(i, p) * elem->GetGradBasis(i, p) * vec[elem->GetNodes()[i]] * vec[elem->GetNodes()[i]] * diff;
    //      //val += elem->GetBasis(i, p) * vec[elem->GetNodes()[i]] * diff;
    //      temp = elem->GetGradBasis(i, p);
    //      val.x += temp.x * vec[elem->GetNodes()[i]] * (diff);
    //      val.y += temp.y * vec[elem->GetNodes()[i]] * (diff);
    //      val.z += temp.z * vec[elem->GetNodes()[i]] * (diff);
    //  }
    //  return val;
    //}
    template<class Problem, class Grid, class Matrix>
    const std::vector<double> DGMethod<Problem, Grid, Matrix>::GetRightVector() const
    {
        return *m_rhsvector;
    }
    template<class Problem, class Grid, class Matrix>
    void DGMethod<Problem, Grid, Matrix>::OutDatFormat(const Mesh::Point& mn, const Mesh::Point& mx, const std::string& file_name, const std::vector<double>& vec) const
    {
        std::ofstream of(file_name + "z.dat");
        std::streambuf *buf = std::cout.rdbuf();
        std::cout.rdbuf(of.rdbuf());
        std::cout « "TITLE = FE-METHOD\n";
        std::cout « "VARIABLES = \"dx1\", \"dx2\", \"u\"\n";
        std::cout « "ZONE i=51, j=51, F=POINT\n";
        double stepx = (mx.x - mn.x) / 51;
        double stepy = (mx.y - mn.y) / 51;
        for (int i = 0; i < 51; ++i)
```

```
                for (int j = 0; j < 51; ++j)
                    std::cout « mn.x + j * stepx « "\t" « mn.y + stepy * i « "\t" « GetValue(Mesh::Point(mn.x + j * stepx, mn.y + i * stepy, mn.z), vec) « std::endl;
            std::cout.rdbuf(buf);
            of.close();
            of.open(file_name + "x.dat");
            buf = std::cout.rdbuf();
            std::cout.rdbuf(of.rdbuf());
            std::cout « "TITLE = FE-METHOD\n";
            std::cout « "VARIABLES = \"dx1\", \"dx2\", \"u\"\n";
            std::cout « "ZONE i=51, j=51, F=POINT\n";
            for (int i = 0; i < 51; ++i)
                for (int j = 0; j < 51; ++j)
                    std::cout « mn.x + j * stepx « "\t" « mn.y + stepy * i « "\t" « GetValue(Mesh::Point(mn.z, mn.x + j * stepx, mn.y + i * stepy), vec) « std::endl;
            std::cout.rdbuf(buf);
            of.close();
            of.open(file_name + "y.dat");
            buf = std::cout.rdbuf();
            std::cout.rdbuf(of.rdbuf());
            std::cout « "TITLE = FE-METHOD\n";
            std::cout « "VARIABLES = \"dx1\", \"dx2\", \"u\"\n";
            std::cout « "ZONE i=51, j=51, F=POINT\n";
            for (int i = 0; i < 51; ++i)
                for (int j = 0; j < 51; ++j)
                    std::cout « mn.x + j * stepx « "\t" « mn.y + stepy * i « "\t" « GetValue(Mesh::Point(mn.x + j * stepx, mn.z, mn.y + i * stepy), vec) « std::endl;
            std::cout.rdbuf(buf);
            of.close();
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethod<Problem, Grid, Matrix>::ApplySources()
        {
            int nfem = -1;
            auto total = m_problem->get_total_sources();
            for (int i = 0; i < total; ++i)
            {
                auto src = m_problem->get_point_source(i);
                auto point = src.get_point();
                nfem = m_Grid->FindElement(point);
                if (nfem != -1)
                {
                    auto val = src.get_value();
                    auto elem = m_Grid->GetElement(nfem);
                    for (int j = 0; j < 3; ++j)
                        m_rhsvector->operator[](elem->GetNode(j)) += val * elem->GetShapeFunction(j, point);
                }
                nfem = -1;
            }
            /*for (auto srd : m_Grid->GetDottedSources())
            {
                nfem = m_Grid->FindElement(srd.first);
                if (nfem != -1)
                {
                    auto elem = m_Grid->GetElements()[nfem];
                    for (int i = 0; i < elem->GetDoF(); ++i)
                    {
                        m_rhsvector[elem->GetNodes()[i]] += srd.second * elem->GetBasis(i, srd.first);
                    }
                }
                nfem = -1;
            }*/
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethod<Problem, Grid, Matrix>::Rediscretization(const std::shared_ptr<Grid>& grid)
        {
            m_GlobalMatrix->NullMatrix();
            for (unsigned int i = 0; i < m_rhsvector->size(); ++i)
                (*m_rhsvector)[i] = 0;
            AssemblGlobal();
            //SecondConditions();
            //ApplySources();
            //StefanConditions();
            MainConditions();
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethod<Problem, Grid, Matrix>::Rediscretization()
        {
            m_time += m_step;
            m_GlobalMatrix->NullMatrix();
            for (unsigned int i = 0; i < m_rhsvector->size(); ++i)
                (*m_rhsvector)[i] = 0;
            AssemblGlobal();
            SecondConditions();
            ThirdConditions();
            StefanConditions();
```

```
            //ApplySources();
            MainConditions();
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethod<Problem, Grid, Matrix>::GetSolution(std::vector<double>& vec)
        {
            int size = vec.size();
            //Translation(vec);
            for (int i = 0; i < size; ++i)
                vec[i] = m_solution[i];
        }
        template<class Problem, class Grid, class Matrix>
        const double DGMethod<Problem, Grid, Matrix>::GetSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p)
        {
            double sum{ 0 };
            auto nfem{ g.FindElement(p) };
            if (nfem < 0)
                return 0.;
            auto elem{ g.GetElement(nfem) };
            auto dofs{ elem->GetDoFs() };
            for (auto i{ 0 }; i < dofs; ++i)
                sum += weights[elem->GetNode(i)] * elem->GetShapeFunction(i, p);
            return sum;
        }
        template<class Problem, class Grid, class Matrix>
        const double DGMethod<Problem, Grid, Matrix>::GetSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p, const int nfem)
        {
            double sum{ 0 };
            //if (nfem < 0)
            //    return 0.;
            auto elem{ g.GetElement(nfem) };
            auto dofs{ elem->GetDoFs() };
            //std::cout « nfem « std::endl;
            for (auto i{ 0 }; i < dofs; ++i)
                sum += weights[elem->GetNode(i)] * elem->GetShapeFunction(i, p);
            return sum;
        }
        template<class Problem, class Grid, class Matrix>
        const Mesh::Point DGMethod<Problem, Grid, Matrix>::GetGradSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p)
        {
            Mesh::Point sum{ 0, 0, 0 };
            auto nfem{ g.FindElement(p) };
            auto elem{ g.GetElement(nfem) };
            auto dofs{ elem->GetDoFs() };
            for (auto i{ 0 }; i < dofs; ++i)
                sum += weights[elem->GetNode(i)] * elem->GetGradShapeFunction(i, p);
            return sum;
        }
        template<class Problem, class Grid, class Matrix>
        const Mesh::Point DGMethod<Problem, Grid, Matrix>::GetGradSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p, const int nfem)
        {
            Mesh::Point sum{ 0, 0, 0 };
            auto elem{ g.GetElement(nfem) };
            auto dofs{ elem->GetDoFs() };
            for (auto i{ 0 }; i < dofs; ++i)
                sum += weights[elem->GetNode(i)] * elem->GetGradShapeFunction(i, p);
            return sum;
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethod<Problem, Grid, Matrix>::LoadSolution(const std::vector<double>& vec)
        {
            m_solution.resize(vec.size());
            for (unsigned int i = 0; i < vec.size(); ++i)
                m_solution[i] = vec[i];
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethod<Problem, Grid, Matrix>::OutMeshFormat(const std::string& file_name, const std::vector<double>& vec)
        {
            const int size{ (int)m_Grid->GetNodes().size() };
            const int number{ (int)m_Grid->GetElements().size() };
            //const int size{ number * 4 };
            std::ofstream ofs(file_name + ".dat", std::ios::out);
            std::string title("TITLE = \"Mesh data\"\n Variables = \"X\", \"Y\", \"Z\", \"U\"\n Zone N = " + std::to_string(size) + ", E = " + std::to_string(number) + ", DATAPACKING = POINT, ZONETYPE = FETETRAHEDRON\n");
            ofs « title;
            Mesh::Point p;
            for (int i = 0; i < size; ++i)
            {
                p = m_Grid->GetNodes()[i];
                ofs « p.x « "\t" « p.y « "\t" « p.z « "\t" « GetValue(p, vec, 1) « std::endl;
            }
```

```
            for (int i = 0; i < number; ++i)
            {
                auto elem = m_Grid->GetElements()[i];
                for (int k = 0; k < 4; ++k)
                {
                    ofs « elem->GetNodes()[k] + 1 « "\t";
                }
                ofs « std::endl;
            }
            ofs.close();
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethod<Problem, Grid, Matrix>::OutMeshTimeFormat(const std::string& file_name, const std::vector<double>& vec)
        {
            const int size{ (int)m_Grid->GetNodes().size() };
            const int number{ (int)m_Grid->GetElements().size() };
            //const int size{ number * 4 };
            std::ofstream ofs(file_name + ".dat", std::ios::out | std::ios::app);
            std::string title("TITLE = \"Mesh data\"\n Variables = \"X\", \"Y\", \"Z\", \"U\"\n Zone N = " + std::to_string(size) + ", E = " + std::to_string(number) + ", DATAPACKING = POINT, ZONETYPE = FETETRAHEDRON\n");
            ofs « title;
            Mesh::Point p;
            for (int i = 0; i < size; ++i)
            {
                p = m_Grid->GetNodes()[i];
                ofs « p.x « "\t" « p.y « "\t" « p.z « "\t" « GetValue(p, vec, 1) « std::endl;
            }
            for (int i = 0; i < number; ++i)
            {
                auto elem = m_Grid->GetElements()[i];
                for (int k = 0; k < 4; ++k)
                {
                    ofs « elem->GetNodes()[k] + 1 « "\t";
                }
                ofs « std::endl;
            }
            ofs.close();
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethod<Problem, Grid, Matrix>::ProjectSolution(std::vector<double>& sol, std::function<const double(const Mesh::Point&, const std::vector<double>&, const int)> GetVal, std::vector<double>& vec)
        {
            for (int i = 0; i < m_Grid->GetElements().size(); ++i)
            {
                auto elem = m_Grid->GetElements()[i];
                int order = elem->GetDoF();
                for (int j = 0; j < order; ++j)
                    sol[elem->GetNodes(j)] = GetVal(m_Grid->GetNodes()[elem->GetNodes(j)], vec, i);
            }
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethod<Problem, Grid, Matrix>::ProjectSolution(std::vector<double>& sol, std::function<const double(const Mesh::Point&, const std::vector<double>&)> GetVal, std::vector<double>& vec, const int)
        {
            for (int i = 0; i < m_Grid->GetElements().size(); ++i)
            {
                auto elem = m_Grid->GetElements()[i];
                int order = elem->GetDoF();
                for (int j = 0; j < order; ++j)
                    sol[elem->GetNodes(j)] = GetVal(m_Grid->GetNodes()[elem->GetNodes(j)], vec);
            }
        }
        template<class Problem, class Grid, class Matrix>
        const std::vector<double> DGMethod<Problem, Grid, Matrix>::SetSolution(const int sol, const int liq, const double s, const double l, const double m)
        {
            int i;
            m_solution.resize(m_Grid->GetNodes().size());
            for (i = 0; i < m_Grid->GetElements().size(); ++i)
            {
                auto elem = m_Grid->GetElements()[i];
                int order = elem->GetDoF();
                if (m_Grid->GetElements()[i]->GetType() == liq)
                    for (int j = 0; j < order; ++j)
                        m_solution[elem->GetNodes()[j]] = l;
                else
                    for (int j = 0; j < order; ++j)
                        m_solution[elem->GetNodes()[j]] = s;
            }

            for (auto bnd : m_Grid->GetBoundaryConditions())
            {
                //if (get<0>(bnd.second) == 4)
```

```
                {
                    for (auto row : m_Grid->GetBoundary())
                    {
                        if (bnd.first == row->GetType())
                        {
                            int dofs = row->GetDoF();
                            for (int i = 0; i < dofs; ++i)
                            {
                                m_solution[row->GetNodes(i)] = m;
                            }
                        }
                    }
                }
            }
            return m_solution;
        }
        template<class Problem, class Grid, class Matrix>
        DGMethod<Problem, Grid, Matrix>::~DGMethod()
        {
            delete m_Grid;
        }
    }
}

#endif // !CORENC_METHODS_DGMethod_h
```

7.73 CoreNCFEM/Methods/DGMethodZero.h File Reference

```
#include <functional>
#include <set>
#include "../Point.h"
#include "../Parameter.h"
#include "CSMethod.h"
#include <memory>
#include <cmath>
#include <map>
#include <algorithm>
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
```

Classes

- class corenc::method::CDGMethodZero< Type >
- class corenc::method::DGMethodZero< Problem, Grid, Matrix >

Namespaces

- namespace corenc
- namespace corenc::Mesh
- namespace corenc::method

7.74 DGMethodZero.h

Go to the documentation of this file.

```
#ifndef DGMETHODZERO_H
#define DGMETHODZERO_H

// DGMethodZero.h describes an abstract interface and functions for a DG method with zero Dirichlet boundaries
#include <functional>
#include <set>
#include "../Point.h"
#include "../Parameter.h"
#include "CSMethod.h"
#include <memory>
#include <cmath>
#include <map>
#include <algorithm>
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
namespace corenc
{
    namespace Mesh
    {
        class Point;
    }
    namespace method
    {
        // class Type = Type of the solution, for ex vector or double, or even more specific


        template<class Type>
        class CDGMethodZero
        {
        public:
            CDGMethodZero() {};
            virtual ~CDGMethodZero() {};
            virtual const int                          Assemble() = 0;
            virtual const Type                         GetSolution(const std::vector<double>& point) const = 0;
            virtual const std::vector<Type>            GetSolution() const = 0;
            virtual const Type                         GetMaxSolution() const = 0;
            virtual const Type                         GetMinSolution() const = 0;
        };

        template<class Problem, class Grid, class Matrix>
        class DGMethodZero
        {
        public:
            DGMethodZero() :
                m_problem{nullptr},
                m_Grid{nullptr},
                m_GlobalMatrix{nullptr},
                m_RightMatrix{nullptr},
                m_rhsvector{nullptr}
            {}
            DGMethodZero(
                Problem* p,
                Grid* g,
                Matrix* m,
                std::vector<double>* rhs):
                m_problem{ p },
                m_Grid{ g->Clone() },
                m_GlobalMatrix{ m },
                m_N{ g->GetNumberOfElements() },
                m_Ns{ g->GetNumberOfBoundaries() },
                m_rhsvector{ rhs }{
                //GeneratePortrait();
            }
            DGMethodZero(
                Problem* p,
                Grid* g,
                Matrix* m,
                Matrix* rm,
                std::vector<double>* rhs):
                m_problem{ p },
                m_Grid{ g->Clone() },
                m_GlobalMatrix{ m },
                m_RightMatrix{ rm },
                m_N{ g->GetNumberOfElements() },
                m_Ns{ g->GetNumberOfBoundaries() },
                m_rhsvector{ rhs }{
                //GeneratePortrait();
            }
```

```
        DGMethodZero(const std::shared_ptr<Grid>& grid) :m_Grid{ grid->Clone() } {}
        DGMethodZero(Grid* grid) :m_Grid{ grid->Clone() } {}
        DGMethodZero(const DGMethodZero& meth) :
            m_Grid{ meth.m_Grid->Clone() },
            //m_GlobalMatrix{ meth.m_GlobalMatrix->Clone() },
            //m_rhsvector{ meth.m_rhsvector },
            //m_problem{ meth.m_problem },
            m_time{ meth.m_time },
            //m_solution{ meth.m_solution },
            m_size{ meth.m_size },
            m_N{ meth.m_N },
            m_Ns{ meth.m_Ns },
            m_nums{ meth.m_nums }
        {};
        void                        Discretization();
        const double                GetValue(const Mesh::Point&) const;
        const double                GetValue(const Mesh::Point&, const std::vector<double>& vec) const;
        const double                GetValue(const Mesh::Point&, const std::vector<double>& vec, const int num) const;
        //const Mesh::Point         GetGradValue(const Mesh::Point&, const std::vector<double>& vec) const;
         //const Mesh::Point         GetLambdaGrad(const Mesh::Point&, const std::vector<double>& vec) const;
         const double                GetEffective(const std::vector<double>& vec) const;
         void                        ProjectSolution(std::vector<double>&, std::function<const double(const Mesh::Point&, const std::vector<double>&, const int)> GetValue, std::vector<double>& sol);
         void                        ProjectSolution(std::vector<double>&, std::function<const double(const Mesh::Point&, const std::vector<double>&)> GetValue, std::vector<double>& sol, const int);
         void                        LoadSolution(const std::vector<double>& vec);
         const std::vector<double>   SetSolution(const int sol, const int liq, const double, const double, const double);
         void                        GetSolution(std::vector<double>& vec);
         void                        Rediscretization(const std::shared_ptr<Grid>&);
         void                        Rediscretization();
         void                        SetTimeStep(const double& step) { m_step = step; m_time = step; }
         Matrix*                     GetGlobalMatrix() const;
         Grid*                       GetMesh() { return m_Grid; }
         const std::vector<double>   GetRightVector() const;
         void                        OutDatFormat(const Mesh::Point& min, const Mesh::Point& max, const std::string& file_name, const std::vector<double>& vec) const;
         void                        OutMeshFormat(const std::string& file_name, const std::vector<double>& vec);
         void                        OutMeshTimeFormat(const std::string& file_name, const std::vector<double>& vec);
         static const double         GetSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p);
         static const double         GetSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p, const int nfem);
         static const Mesh::Point GetGradSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p);
         static const Mesh::Point GetGradSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p, const int n);
         ~DGMethodZero();
     private:
         void                        GeneratePortrait();
         void                        AssemblGlobal();
         void                        MainConditions();
         void                        SecondConditions();
         void                        ThirdConditions();
         void                        StefanConditions();
         void                        ApplySources();
         const int                   AssembleLocalMatrix(const int);
         const int                   AssembleIDUDVMatrix(const int);
         const int                   AssembleIDUVMatrix(const int);
         const int                   AssembleIUDVMatrix(const int);
         const int                   AssembleRUVMatrix(const int);
         const int                   AssembleSUPGMatrix(const int);
         const int                   AssembleLocalMatrix(const int, const int);
         const int                   AssembleInter();
         Grid*                       m_Grid = nullptr;
         Matrix*                     m_GlobalMatrix = nullptr;
         Matrix*                 m_RightMatrix = nullptr;
         Problem*                    m_problem = nullptr;
         std::vector<double>         m_solution;
         std::vector<double>*        m_rhsvector;
         unsigned int                m_size;
         double                      m_step{ 0.1 };
         double                      m_time{ 0.1 };
         unsigned int                m_N;
         unsigned int                m_Ns;
         std::vector<unsigned int>   m_nums;
         // interpolation nodes
         std::vector<std::vector<int»   m_inums;
```

```

    };

    template<class Problem, class Grid, class Matrix>
    void DGMethodZero<Problem, Grid, Matrix>::Discretization()
    {
        GeneratePortrait();
        AssemblGlobal();
        AssembleInter();
        //ApplySources();
        //SecondConditions();
        //ThirdConditions();
        //MainConditions();
        //StefanConditions();
    }
    template<class Problem, class Grid, class Matrix>
    void DGMethodZero<Problem, Grid, Matrix>::GeneratePortrait()
    {
        const auto& el = m_Grid->GetElement(0);
        int order = m_Grid->GetElement(0)->GetDoFs();
        std::vector<std::set<unsigned int» temp;
        //m_Ns = m_Grid->GetNumberOfINodes();
        m_Ns = m_Grid->GetNumberOfBoundaries();
        m_N = m_Grid->GetNumberOfElements();
        //temp.resize(m_Grid->GetNumberOfINodes());
        unsigned i, j, k;
        m_nums.resize(m_N);
        m_inums.resize(m_N);
        int size;
        m_size = 0;
        std::cout « "nums" « std::endl;
        for (k = 0; k < m_N; ++k)
        {
            const auto& elem{ m_Grid->GetElement(k) };
            size = 0;
            m_inums[k].resize(order);
            for (i = 0; i < order; ++i)
            {
                if (m_Grid->interpolate(elem->GetNode(i)) > -1)
                {
                    m_inums[k][i] = size;
                    ++size;
                }
            }
            m_nums[k] = m_size;
            m_size += size;
            std::cout « k « "\t" « m_nums[k] « std::endl;
        }
        int sz = m_Ns;
        int nk, ne;
        int sizej = 0;
        int sizei = 0;
        temp.resize(m_size);
        for (k = 0; k < sz; ++k)
        {
            auto bound = m_Grid->GetBoundary(k);
            nk = bound->GetNeighbour(0);
            ne = bound->GetNeighbour(1);
            std::cout « nk « ne « std::endl;
            sizei = 0;
            sizej = 0;
            if (ne != -1)
            {
                auto elemk = m_Grid->GetElement(nk);
                auto eleme = m_Grid->GetElement(ne);
                size = 0;
                for (i = 0; i < order; ++i)
                {
                    int inode = m_Grid->interpolate(elemk->GetNode(i));
                    if (inode > -1)
                    {
                        //sizej = sizei + 1;
                        for (j = i + 1; j < order; ++j)
                        {
                            int jnode = m_Grid->interpolate(elemk->GetNode(j));
                            if (jnode > - 1)
                            {
                                temp[m_nums[nk] + m_inums[nk][j]].insert(m_nums[nk] +
m_inums[nk][i]);
                                //temp[m_nums[nk] + sizej].insert(m_nums[nk] + sizei);
                                //++sizej;
                                //std::cout « "k";
                            }
                        }
                        //++sizei;
                    }
                }
```

```
                sizei = 0;
                sizej = 0;
                for (i = 0; i < order; ++i)
                {
                    int inode = m_Grid->interpolate(elemk->GetNode(i));
                    if (inode > -1)
                    {
                        sizej = 0;
                        for (j = 0; j < order; ++j)
                        {
                            int jnode = m_Grid->interpolate(eleme->GetNode(j));
                            if (jnode > - 1)
                            {
                                temp[m_nums[ne] + m_inums[ne][j]].insert(m_nums[nk] + m_inums[nk][i]);
                                //temp[m_nums[ne] + sizej].insert(m_nums[nk] + sizei);
                                ++sizej;
                                std::cout « "k";
                            }
                        }
                        ++sizei;
                    }
                }
            }
            else
            {
                sizei = 0;
                sizej = 0;
                auto elemk = m_Grid->GetElement(nk);
                size = 0;
                for (i = 0; i < order; ++i)
                {
                    int inode = m_Grid->interpolate(elemk->GetNode(i));
                    if (inode > -1)
                    {
                        sizej = sizei + 1;
                        for (j = i + 1; j < order; ++j)
                        {
                            int jnode = m_Grid->interpolate(elemk->GetNode(j));
                            if (jnode > - 1)
                            {
                                temp[m_nums[nk] + m_inums[nk][j]].insert(m_nums[nk] + m_inums[nk][i]);
                                //temp[m_nums[nk] + sizej].insert(m_nums[nk] + sizei);
                                ++sizej;
                                std::cout « "k";
                            }
                        }
                        ++sizei;
                    }
                }
            }
        }
        if(m_problem->findTerm(Terms::RUV))
            m_RightMatrix->Create(temp.size(), temp);

        for (auto & it : temp)
        {
            for (auto& it2 : it)
                std::cout « it2 « "\t";
            std::cout « std::endl;
        }
        //m_GlobalMatrix = std::shared_ptr<Matrix>(new Matrix(m_Grid->GetNumberOfNodes(), temp));
        //m_rhsvector.resize(m_Grid->GetNumberOfNodes());
        //std::cout « temp.size() « std::endl;
        m_GlobalMatrix->Create(temp.size(), temp);
        m_rhsvector->resize(temp.size());
        //m_solution.resize(m_Grid->GetNumberOfNodes());
        //for (int l = 0; l < m_Grid->GetNumberOfNodes(); ++l)
        //  m_solution[l] = 20;
    }
    template<class Problem, class Grid, class Matrix>
    void DGMethodZero<Problem, Grid, Matrix>::AssemblGlobal()
    {
        int l;
        //std::vector<std::future<int» futures;
        int i, j, k, nodes;
        double mij;
        const int terms{ (int)m_problem->getNumberOfTerms() };
        for (k = 0; k < terms; ++k)
        {
            switch (m_problem->getTerm(k))
            {
                case Terms::IDUDV:
                    for (l = 0; l < m_N; ++l)
                    {
                        std::cout « "IDUDV: " « l « std::endl;
```

```
                            AssembleIDUDVMatrix(l);
                        }
                        break;
                    case Terms::IDUV:
                        for (l = 0; l < m_N; ++l)
                            AssembleIDUVMatrix(l);
                        break;
                    case Terms::IUDV:
                        for (l = 0; l < m_N; ++l)
                            AssembleIUDVMatrix(l);
                        break;
                    case Terms::SUPG:
                        for (l = 0; l < m_N; ++l)
                            AssembleSUPGMatrix(l);
                        break;
                    case Terms::RUV:
                        for (l = 0; l < m_N; ++l)
                            AssembleRUVMatrix(l);
                        break;
                    default:
                        break;
                }
            }
            //for (l = 0; l < m_N; ++l)
                //futures.push_back(async(&DGMethod<Problem, Grid, Matrix>::AssembleLocalMatrix, this, l));
             //   AssembleLocalMatrix(l, 0);
            //for (auto &it : futures)
            //it.get();
        }

        template<class Problem, class Grid, class Matrix>
        const int DGMethodZero<Problem, Grid, Matrix>::AssembleIDUDVMatrix(const int l)
        {
            int i, j, k, nodes;
            double mij;
            const auto& elem{ m_Grid->GetElement(l) };
            const int dofs{ (int)elem->GetDoFs() };
            const int terms{ (int)m_problem->getNumberOfTerms() };
            nodes = elem->GetNumberOfNodes();
            std::vector<Mesh::Point> points(nodes);
            for (i = 0; i < nodes; ++i)
                points[i] = m_Grid->GetNode(elem->GetNode(i));
            int sizei = 0, sizej = 0;
            for (i = 0; i < (int)dofs; ++i)
            {
                auto inode = m_Grid->interpolate(elem->GetNode(i));
                if (inode == -1)
                    continue;
                sizej = 0;
                for (j = 0; j < (int)dofs; ++j)
                {
                    auto jnode = m_Grid->interpolate(elem->GetNode(j));
                    if (jnode == -1)
                        continue;
                    auto M = [&](const Mesh::Point& p)
                    {
                        //auto m = elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                        return m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p) * elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                    };
                    //mij = m_Grid->getParameter(Parameters::DIFFUSION, l, j) * elem->Integrate(M, points);
                    mij = elem->Integrate(M, points);
                    //m_GlobalMatrix->AddElement(inode, jnode, mij);
                    m_GlobalMatrix->AddElement(m_nums[l] + sizei, m_nums[l] + sizej, mij);
                    ++sizej;
                }
                ++sizei;
            }
            return 0;
        }

        template<class Problem, class Grid, class Matrix>
        const int DGMethodZero<Problem, Grid, Matrix>::AssembleIDUVMatrix(const int l)
        {
            int i, j, k, nodes;
            double mij;
            const auto& elem{ m_Grid->GetElement(l) };
            const int dofs{ (int)elem->GetDoFs() };
            const int terms{ (int)m_problem->getNumberOfTerms() };
            nodes = elem->GetNumberOfNodes();
            std::vector<Mesh::Point> points(nodes);
            for (i = 0; i < nodes; ++i)
                points[i] = m_Grid->GetNode(elem->GetNode(i));
            int sizei = 0, sizej = 0;
            for (i = 0; i < (int)dofs; ++i)
```

```
            {
                auto inode = m_Grid->interpolate(elem->GetNode(i));
                if (inode == -1)
                    continue;
                sizej = 0;
                for (j = 0; j < (int)dofs; ++j)
                {
                    auto jnode = m_Grid->interpolate(elem->GetNode(j));
                    if (jnode == -1)
                        continue;
                    auto M = [&](const Mesh::Point& p)
                    {
                        return m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * elem->GetShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                    };
                    auto _mij = elem->Integrate(M, points);
                    //m_GlobalMatrix->AddElement(inode, jnode, _mij);
                    m_GlobalMatrix->AddElement(m_nums[l] + sizei, m_nums[l] + sizej, _mij);
                    ++sizej;
                }
                ++sizei;
            }
            return 0;
        }

        template<class Problem, class Grid, class Matrix>
        const int DGMethodZero<Problem, Grid, Matrix>::AssembleIUDVMatrix(const int l)
        {
            int i, j, k, nodes;
            double mij;
            const auto& elem{ m_Grid->GetElement(l) };
            const int dofs{ (int)elem->GetDoFs() };
            const int terms{ (int)m_problem->getNumberOfTerms() };
            nodes = elem->GetNumberOfNodes();
            std::vector<Mesh::Point> points(nodes);
            for (i = 0; i < nodes; ++i)
                points[i] = m_Grid->GetNode(elem->GetNode(i));
            int sizei = 0, sizej = 0;
            for (i = 0; i < dofs; ++i)
            {
                auto inode = m_Grid->interpolate(elem->GetNode(i));
                if (inode == -1)
                    continue;
                sizej = 0;
                for (j = 0; j < dofs; ++j)
                {
                    auto jnode = m_Grid->interpolate(elem->GetNode(j));
                    if (jnode == -1)
                        continue;
                    auto M = [&](const Mesh::Point& p)
                    {
                        return elem->GetGradShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                    };
                    //mij = m_CoarseGrid->getParameter(Parameters::ADVECTION, l, j) * m_flux(m_CoarseGrid->getSolution(l, j)) * elem->Integrate(M, points).x;
                    mij = elem->Integrate(M, points).x;
                    //m_GlobalMatrix->AddElement(inode, jnode, mij);
                    m_GlobalMatrix->AddElement(m_nums[l] + sizei, m_nums[l] + sizej, mij);
                    ++sizej;
                }
                ++sizei;
            }
            return 0;
        }


        template<class Problem, class Grid, class Matrix>
        const int DGMethodZero<Problem, Grid, Matrix>::AssembleRUVMatrix(const int l)
        {
            int i, j, k, nodes;
            double mij;
            const auto& elem{ m_Grid->GetElement(l) };
            const int dofs{ (int)elem->GetDoFs() };
            const int terms{ (int)m_problem->getNumberOfTerms() };
            nodes = elem->GetNumberOfNodes();
            std::vector<Mesh::Point> points(nodes);
            for (i = 0; i < nodes; ++i)
                points[i] = m_Grid->GetNode(elem->GetNode(i));
            int sizei = 0, sizej = 0;
            for (i = 0; i < (int)dofs; ++i)
            {
                auto inode = m_Grid->interpolate(elem->GetNode(i));
                if (inode == -1)
                    continue;
                sizej = 0;
                for (j = 0; j < (int)dofs; ++j)
                {
```

```
                auto M = [&](const Mesh::Point& p)
                {
                    double vel = sqrt(m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0));
                    double h = elem->GetMeasure();
                    double Pe = vel * h / 6. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                    double tau = 0.;
                    //double Pe = vel * h / 2. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);

                    if (Pe >= 1)
                        tau = h / 2. / vel;
                    else
                        tau = h * h / 12. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                    auto supg = tau * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * elem->GetGradShapeFunction(i, p) * elem->GetShapeFunction(j, p) * elem->GetShapeFunction(i, p);
                    return elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);// + supg;
                };
                mij = elem->Integrate(M, points);

                auto jnode = m_Grid->interpolate(elem->GetNode(j));
                if (inode > -1 && jnode > -1)
                {
                    m_RightMatrix->AddElement(m_nums[l] + sizei, m_nums[l] + sizej, mij);
                    ++sizej;
                }
            }
            ++sizei;
        }
        return 0;
    }

    template<class Problem, class Grid, class Matrix>
    const int DGMethodZero<Problem, Grid, Matrix>::AssembleSUPGMatrix(const int l)
    {
        int i, j, k, nodes;
        double mij;
        const auto& elem{ m_Grid->GetElement(l) };
        const int dofs{ (int)elem->GetDoFs() };
        const int terms{ (int)m_problem->getNumberOfTerms() };
        nodes = elem->GetNumberOfNodes();
        std::vector<Mesh::Point> points(nodes);
        for (i = 0; i < nodes; ++i)
            points[i] = m_Grid->GetNode(elem->GetNode(i));
        for (i = 0; i < (int)dofs; ++i)
        {
            for (j = 0; j < (int)dofs; ++j)
            {
                auto inode = m_Grid->interpolate(elem->GetNode(i));
                auto jnode = m_Grid->interpolate(elem->GetNode(j));
                if (inode == -1 || jnode == -1)
                    continue;
                auto M = [&](const Mesh::Point& p)
                {
                    double vel = sqrt(m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0));
                    double h = elem->GetMeasure();
                    //double Pe = vel * h / 6. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                    double tau = 0.;
                    double Pe = vel * h / 2. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                    //double beta = h / 2. / vel * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                    //double beta = h / std::sqrt(3.) * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                    //double beta = h / 2. * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                    //double beta = h / 2. * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                    //beta = 0.;
                    //for (int ii = 0; ii < (int)dofs; ++ii)
                        //beta += m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * elem->GetGradShapeFunction(ii, p);
                    //return beta * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) *
                    //        elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                    if (Pe >= 1)
                        tau = h / 2. / vel;
                    else
                        tau = h * h / 12. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                    //return 0.;
                    return tau * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) *
                            elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                };
```

```

                    //double tau =
                    auto _mij = elem->Integrate(M, points);
                    m_GlobalMatrix->AddElement(inode, jnode, _mij);
                }
            }
            return 0;
        }


        template<class Problem, class Grid, class Matrix>
        const int DGMethodZero<Problem, Grid, Matrix>::AssembleInter()
        {
            for (int l = 0; l < m_Ns; ++l)
            {
                const auto& bound{ m_Grid->GetBoundary(l) };
                const auto& nk{ bound->GetNeighbour(0) };
                const auto& ne{ bound->GetNeighbour(1) };
                const auto& elemk{ m_Grid->GetElement(nk) };
                const auto& dofs{ bound->GetDoFs() };
                const auto& dofsk{ elemk->GetDoFs() };
                std::vector<Mesh::Point> points(dofs);
                for (int i = 0; i < dofs; ++i)
                {
                    points[i] = m_Grid->GetNode(bound->GetNode(i));
                }
                if (ne < 0)
                    continue;
                const auto& eleme{ m_Grid->GetElement(ne) };
                for (int i = 0; i < dofsk; ++i)
                {
                    int inode = m_Grid->interpolate(elemk->GetNode(i));
                    if (inode == -1)
                        continue;
                    for (int j = 0; j < dofsk; ++j)
                    {
                        int jnode = m_Grid->interpolate(elemk->GetNode(j));
                        if (jnode == -1)
                            continue;
                        auto Tkk = [&](const Mesh::Point& p)
                        {
                            auto kappa = m_problem->get_parameter(Terms::IDUDV, l, elemk->GetType(), p);
                            auto val1 = bound->GetNormal() * elemk->GetShapeFunction(j, p)  * elemk->GetGradShapeFunction(i, p);
                            auto val2 = bound->GetNormal() * elemk->GetShapeFunction(i, p)  * elemk->GetGradShapeFunction(j, p);
                            return 0.5 * kappa * (val2 - val1);
                        };
                        auto mj = bound->Integrate(Tkk, points);
                        m_GlobalMatrix->AddElement(m_nums[nk] + m_inums[nk][i], m_nums[nk] + m_inums[nk][j], mj);
                    }
                }

                for (int i = 0; i < dofsk; ++i)
                {
                    int inode = m_Grid->interpolate(elemk->GetNode(i));
                    if (inode == -1)
                        continue;
                    for (int j = 0; j < dofsk; ++j)
                    {
                        int jnode = m_Grid->interpolate(elemk->GetNode(j));
                        if (jnode == -1)
                            continue;
                        auto Tkk = [&](const Mesh::Point& p)
                        {
                            auto kappa = m_problem->get_parameter(Terms::IDUDV, l, eleme->GetType(), p);
                            auto val1 = bound->GetNormal() * eleme->GetShapeFunction(j, p)  * elemk->GetGradShapeFunction(i, p);
                            auto val2 = bound->GetNormal() * elemk->GetShapeFunction(i, p)  * eleme->GetGradShapeFunction(j, p);
                            return 0.5 * kappa * (val2 + val1);
                        };
                        auto mj = bound->Integrate(Tkk, points);
                        m_GlobalMatrix->AddElement(m_nums[nk] + m_inums[nk][i], m_nums[ne] + m_inums[ne][j], mj);
                    }
                }
            }
            return 0;
        }

        template<class Problem, class Grid, class Matrix>
        const int DGMethodZero<Problem, Grid, Matrix>::AssembleLocalMatrix(const int l, const int old)
        {
            int i, j, k, nodes;
            double mij;
```

```
        const auto& elem{ m_Grid->GetElement(l) };
        const int dofs{ (int)elem->GetDoFs() };
        const int terms{ (int)m_problem->getNumberOfTerms() };
        nodes = elem->GetNumberOfNodes();
        std::vector<Mesh::Point> points(nodes);
        for (i = 0; i < nodes; ++i)
            points[i] = m_Grid->GetNode(elem->GetNode(i));
        for (k = 0; k < terms; ++k)
        {
            switch (m_problem->getTerm(k))
            {
            case Terms::IUV:
                for (i = 0; i < (int)dofs; ++i)
                {
                    for (j = 0; j < (int)dofs; ++j)
                    {
                        auto M = [&](const Mesh::Point& p)
                        {
                            return m_problem->get_parameter(Terms::IUV, l, elem->GetType(), p) * elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                        };
                        mij = elem->Integrate(M, points);
                        auto inode = m_Grid->interpolate(elem->GetNode(i));
                        auto jnode = m_Grid->interpolate(elem->GetNode(j));
                        if (inode > -1 && jnode > -1)
                            m_GlobalMatrix->AddElement(inode, jnode, mij);
                    }
                }
                break;
            case Terms::IDUDV:
                for (i = 0; i < (int)dofs; ++i)
                {
                    for (j = 0; j < (int)dofs; ++j)
                    {
                        auto inode = m_Grid->interpolate(elem->GetNode(i));
                        auto jnode = m_Grid->interpolate(elem->GetNode(j));
                        if (inode == -1 || jnode == -1)
                            continue;
                        auto M = [&](const Mesh::Point& p)
                        {
                            //auto m = elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                            return m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p) * elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                        };
                        //mij = m_Grid->getParameter(Parameters::DIFFUSION, l, j) * elem->Integrate(M, points);
                        mij = elem->Integrate(M, points);
                        m_GlobalMatrix->AddElement(inode, jnode, mij);
                    }
                }
                break;
            case Terms::IDUV:
                for (i = 0; i < (int)dofs; ++i)
                {
                    for (j = 0; j < (int)dofs; ++j)
                    {
                        auto inode = m_Grid->interpolate(elem->GetNode(i));
                        auto jnode = m_Grid->interpolate(elem->GetNode(j));
                        if (inode == -1 || jnode == -1)
                            continue;
                        auto M = [&](const Mesh::Point& p)
                        {
                            return m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * elem->GetShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                        };
                        auto _mij = elem->Integrate(M, points);
                        m_GlobalMatrix->AddElement(inode, jnode, _mij);
                    }
                }
                break;
            case Terms::IUDV:
                for (i = 0; i < dofs; ++i)
                {
                    for (j = 0; j < dofs; ++j)
                    {
                        auto inode = m_Grid->interpolate(elem->GetNode(i));
                        auto jnode = m_Grid->interpolate(elem->GetNode(j));
                        if (inode == -1 || jnode == -1)
                            continue;
                        auto M = [&](const Mesh::Point& p)
                        {
                            return elem->GetGradShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                        };
                        //mij = m_CoarseGrid->getParameter(Parameters::ADVECTION, l, j) * m_flux(m_CoarseGrid->getSolution(l, j)) * elem->Integrate(M, points).x;
                        mij = elem->Integrate(M, points).x;
```

```
                            m_GlobalMatrix->AddElement(inode, jnode, mij);
                        }
                    }
                    break;
                case Terms::EUV:
                    for (i = 0; i < dofs; ++i)
                    {
                        for (j = 0; j < dofs; ++j)
                        {
                            auto M = [&](const Mesh::Point& p)
                            {
                                return elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                            };
                            mij = elem->Integrate(M, points);
                            m_rhsvector->operator[](elem->GetNode(i)) += m_Grid->getParameter(Parameters::MASS, l, j) * m_Grid->getSolution(l, j) * mij;
                            //m_rhsvector->operator[](m_nums[l] + i) += m_CoarseGrid->getParameter(Parameters::MASS, l, points[j]) * elem->GetValue(j) * mij;
                        }
                    }
                    break;
                case Terms::EDUDV:
                    for (i = 0; i < dofs; ++i)
                    {
                        for (j = 0; j < dofs; ++j)
                        {
                            auto M = [&](const Mesh::Point& p)
                            {
                                return elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                            };
                            mij = elem->Integrate(M, points);
                            m_rhsvector->operator[](elem->GetNode(i)) += m_Grid->getParameter(Parameters::DIFFUSION, l, j) * m_Grid->getSolution(l, j) * mij;
                        }
                    }
                    break;
                case Terms::EDUV:
                    for (i = 0; i < dofs; ++i)
                    {
                        for (j = 0; j < dofs; ++j)
                        {
                            auto M = [&](const Mesh::Point& p)
                            {
                                return elem->GetShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                            };
                            mij = elem->Integrate(M, points).x;
                            m_rhsvector->operator[](elem->GetNode(i)) += m_Grid->getParameter(Parameters::ADVECTION, l, j) * mij;
                        }
                    }
                    break;
                case Terms::EUDV:
                    for (i = 0; i < dofs; ++i)
                    {
                        for (j = 0; j < dofs; ++j)
                        {
                            auto M = [&](const Mesh::Point& p)
                            {
                                return elem->GetGradShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                            };
                            mij = elem->Integrate(M, points).x;
                            m_rhsvector->operator[](elem->GetNode(i)) += m_Grid->getParameter(Parameters::ADVECTION, l, j) * mij;// *mij;
                        }
                    }
                    break;
                case Terms::EFV:
                    for (i = 0; i < dofs; ++i)
                    {
                        /*for (j = 0; j < dofs; ++j)
                        {
                            auto M = [&](const Mesh::Point& p)
                            {
                                return m_problem->get_parameter(Terms::EFV, elem->GetType(), l, j, p) * elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                            };
                            mij = elem->Integrate(M, points);
                            m_rhsvector->operator[](elem->GetNode(i)) += mij;
                        }*/
                        auto M = [&](const Mesh::Point& p)
                        {
                            return m_problem->get_parameter(Terms::EFV, elem->GetType(), l, i, p) * elem->GetShapeFunction(i, p);
                        };
                        mij = elem->Integrate(M, points);
                        m_rhsvector->operator[](elem->GetNode(i)) += mij;
```

```
            }
            break;
        case Terms::RUV:
            for (i = 0; i < (int)dofs; ++i)
            {
                for (j = 0; j < (int)dofs; ++j)
                {
                    auto M = [&](const Mesh::Point& p)
                    {
                        return elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                    };
                    mij = elem->Integrate(M, points);
                    auto inode = m_Grid->interpolate(elem->GetNode(i));
                    auto jnode = m_Grid->interpolate(elem->GetNode(j));
                    if (inode > -1 && jnode > -1)
                        m_RightMatrix->AddElement(inode, jnode, mij);
                }
            }
            break;
        case Terms::SUPG:
            for (i = 0; i < (int)dofs; ++i)
            {
                for (j = 0; j < (int)dofs; ++j)
                {
                    auto inode = m_Grid->interpolate(elem->GetNode(i));
                    auto jnode = m_Grid->interpolate(elem->GetNode(j));
                    if (inode == -1 || jnode == -1)
                        continue;
                    auto M = [&](const Mesh::Point& p)
                    {
                        double vel = sqrt(m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0));
                        double h = elem->GetMeasure();
                        //double Pe = vel * h / 6. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                        double tau = 0.;
                        double Pe = vel * h / 2. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                        //double beta = h / 2. / vel * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                        double beta = h / std::sqrt(3.) * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                        //double beta = h / 2. * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                        //beta = 0.;
                        //for (int ii = 0; ii < (int)dofs; ++ii)
                            //beta += m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * elem->GetGradShapeFunction(ii, p);
                        return beta * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) *
                                elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                        if (Pe >= 1)
                            tau = h / 2. / vel;
                        else
                            tau = h * h / 12. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                        //return 0.;
                        return tau * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) *
                                elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                    };

                    //double tau =
                    auto _mij = elem->Integrate(M, points);
                    m_GlobalMatrix->AddElement(inode, jnode, _mij);
                }
            }
            break;
        default:
            break;
        }
    }
    return 0;
}
template<class Problem, class Grid, class Matrix>
void DGMethodZero<Problem, Grid, Matrix>::MainConditions()
{
    double mu{ 1e10 };
    const auto n = m_problem->get_number_of_boundaries();
    const auto m = m_Grid->GetNumberOfBoundaries();
    for (int i = 0; i < n; ++i)
    {
        const auto& type = m_problem->get_boundary_type(i);
        for (int j = 0; j < m; ++j)
        {
```

```
                    const auto& row = m_Grid->GetBoundary(j);
                    if (row->GetType() == type)
                    {
                        const int dofs = (int)row->GetDoFs();
                        const int dofs2 = 2;
                        const auto& elem_num = row->GetNeighbour(0);
                        const auto& elem = m_Grid->GetElement(elem_num);
                        const int dofs_elem = elem->GetDoFs();
                        std::vector<Mesh::Point> points(dofs_elem);
                        for (int k = 0; k < dofs_elem; ++k)
                            points[k] = m_Grid->GetNode(elem->GetNode(k));
                        for (int k = 0; k < dofs; ++k)
                        {
                            int l = 0;
                            for (; l < dofs_elem; ++l)
                            {
                                if (elem->GetNode(l) == row->GetNode(k))
                                    break;
                            }
                            m_GlobalMatrix->NullRow(row->GetNode(k));
                            //m_GlobalMatrix->operator()(row->GetNode(k), row->GetNode(k)) *= mu;
                            //m_rhsvector->operator[](row->GetNode(k)) = m_problem->get_boundary_parameter(0, type, m_Grid->GetNode(row->GetNode(k)));
                            //m_rhsvector->operator[](row->GetNode(k)) = m_problem->get_boundary_parameter(0, type, elem_num, l, m_Grid->GetNode(row->GetNode(k)));
                            m_rhsvector->operator[](row->GetNode(k)) = elem->GetWeight(l, points, [=](const Mesh::Point& p) { return m_problem->get_boundary_parameter(0, type, p); });
                            if(m_problem->findTerm(Terms::RUV))
                                {
                                    m_RightMatrix->NullRow(row->GetNode(k));
                                    //m_RightMatrix->operator()(row->GetNode(k), row->GetNode(k)) *= mu;
                                }
                        }
                        /*for (int k = dofs2; k < dofs; ++k)
                        {
                            m_GlobalMatrix->NullRow(row->GetNode(k));
                            m_rhsvector->operator[](row->GetNode(k)) = 0;
                        }*/
                    }
                }
            }
            /*for (auto bnd : m_Grid->GetBoundaryConditions())
            {
                if (get<0>(bnd.second) == 1)
                    for (auto row : m_Grid->GetBoundary())
                    {
                        if (bnd.first == row->GetType())
                        {
                            for (int i = 0; i < row->GetDoF(); ++i)
                            {
                                m_GlobalMatrix->NullRow(row->GetNodes(i));
                                m_rhsvector[row->GetNodes(i)] = get<1>(bnd.second)(m_Grid->GetNodes()[row->GetNodes(i)]);
                            }
                        }
                    }
            }*/
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethodZero<Problem, Grid, Matrix>::SecondConditions()
        {
            double theta = 0;
            int nfem;
            Mesh::Point temp[3];
            std::vector<int> local;
            for (auto bnd : m_Grid->GetBoundaryConditions())
            {
                //if (get<0>(bnd.second) == 2)
                {
                    for (auto row : m_Grid->GetBoundary())
                    {
                        if (bnd.first == row->GetType())
                        {
                            local.resize(0);
                            int dofs = row->GetDoF();
                            nfem = row->GetNumberOfElement(0);
                            auto elem = m_Grid->GetElements()[nfem];
                            //auto GetBasis = [&](int t, Point p){return elem->GetBasis(t, p); };
                            for (int j = 0; j < dofs; ++j)
                            {
                                temp[j] = m_Grid->GetNodes()[row->GetNodes(j)];
                                for (int i = 0; i < elem->GetDoF(); ++i)
                                {
                                    if (row->GetNodes(j) == elem->GetNodes()[i])
                                    {
                                        local.push_back(i);
                                        break;
```

```
                                }
                            }
                        }
                        for (int i = 0; i < dofs; ++i)
                        {
                            for (int j = 0; j < dofs; ++j)
                            {
                                //theta = get<1>(bnd.second)(m_Grid->GetNodes()[row->GetNodes(i)]);
                                theta = 0;
                                auto GetMass = [&](const Mesh::Point& p) {return elem->GetBasis(local[j], p) * elem->GetBasis(local[i], p); };
                                auto GetBBasis = [&](const Mesh::Point& p) {return row->GetBasis(j, p)*row->GetBasis(i, p); };
                                //if (i < 2 || j < 2)
                                m_rhsvector[row->GetNodes(i)] += theta * row->Integrate(GetMass, temp);

                                //if (i < 3 || j < 3)
                                //  m_rhsvector[row[i + 1]] += theta * row->Integrate(GetBBasis, temp);
                            }
                        }
                    }
                }
            }
        }
    }
    template<class Problem, class Grid, class Matrix>
    void DGMethodZero<Problem, Grid, Matrix>::StefanConditions()
    {
        double dest{ 0. }, lat{ 0 };
        int nfem;
        Mesh::Point temp[3];
        std::vector<int> local;
        for (auto bnd : m_Grid->GetBoundaryConditions())
        {
            //if (get<0>(bnd.second) == 4)
            {
                lat = 0;
                //lat = get<2>(bnd.second);
                for (auto row : m_Grid->GetBoundary())
                 {
                     if (bnd.first == row->GetType())
                     {
                         local.resize(0);
                         int dofs = row->GetDoF();
                         nfem = row->GetNumberOfElement(0);
                         auto elem = m_Grid->GetElements()[nfem];
                         //auto GetBasis = [&](int t, Point p){return elem->GetBasis(t, p); };
                         for (int j = 0; j < dofs; ++j)
                         {
                             temp[j] = m_Grid->GetNodes()[row->GetNodes(j)];
                             for (int i = 0; i < elem->GetDoF(); ++i)
                             {
                                 if (row->GetNodes(j) == elem->GetNodes()[i])
                                 {
                                     local.push_back(i);
                                     break;
                                 }
                             }
                         }
                         for (int i = 0; i < dofs; ++i)
                         {
                             for (int j = 0; j < dofs; ++j)
                             {
                                 dest = 0;
                                 //dest = get<1>(bnd.second)(m_Grid->GetNodes()[row->GetNodes(i)]);
                                 auto GetBBasis = [&](const Mesh::Point& p) {return row->GetBasis(j, p)*row->GetBasis(i, p); };
                                 //if (i < 2 || j < 2)
                                 m_rhsvector[row->GetNodes(i)] += dest * lat * row->Integrate(GetBBasis, temp);

                                 //if (i < 3 || j < 3)
                                 //  m_rhsvector[row[i + 1]] += theta * row->Integrate(GetBBasis, temp);
                             }
                         }
                     }
                 }
             }
         }
     }
     template<class Problem, class Grid, class Matrix>
     void DGMethodZero<Problem, Grid, Matrix>::ThirdConditions()
     {
         double param{ 0 }, beta{ 0 };
```

```
            int nfem;
            Mesh::Point temp[6];
            std::vector<int> local;
            auto fxy = [&](const Mesh::Point& p) {return (10 * p.y*m_time + m_time) / 10; };
            //auto fxy = [&](const Point& p){return 10 * p.y + 10 * m_time; };
            for (auto bnd : m_Grid->GetBoundaryConditions())
            {
                //if (get<0>(bnd.second) == 3)
                {

                    for (auto row : m_Grid->GetBoundary())
                    {
                        if (bnd.first == row->GetType())
                        {
                            local.resize(0);
                            int dofs = row->GetDoF();
                            nfem = row->GetNumberOfElement(0);
                            auto elem = m_Grid->GetElements()[nfem];
                            //auto GetBasis = [&](int t, Point p){return elem->GetBasis(t, p); };
                            auto order = elem->GetDoF();
                            for (int j = 0; j < dofs; ++j)
                            {
                                temp[j] = m_Grid->GetNodes()[row->GetNodes(j)];
                                for (int i = 0; i < order; ++i)
                                {
                                    if (row->GetNodes(j) == elem->GetNodes()[i])
                                    {
                                        local.push_back(i);
                                        break;
                                    }
                                }
                            }
                            double val{ 0 };
                            for (int i = 0; i < dofs; ++i)
                            {
                                for (int j = 0; j < dofs; ++j)
                                {
                                    param = 0;
                                    beta = 0;
                                    //beta = get<2>(bnd.second);
                                    //param = get<1>(bnd.second)(m_Grid->GetNodes()[row->GetNodes(i)]);
                                    //param = fxy(temp[j]);
                                    auto GetBBasis = [&](const Mesh::Point& p) {return
      elem->GetBasis(local[j], p)*elem->GetBasis(local[i], p); };
                                    //val = row->GetElement(GetBBasis, temp);
                                    val = row->Integrate(GetBBasis, temp);
                                    m_GlobalMatrix->operator()(row->GetNodes(i), row->GetNodes(j)) +=
      beta * val;
                                    m_rhsvector[row->GetNodes(i)] += beta * param * val;
                                }
                            }
                        }
                    }
                }
            }
        }
        template<class Problem, class Grid, class Matrix>
        Matrix* DGMethodZero<Problem, Grid, Matrix>::GetGlobalMatrix() const
        {
            return m_GlobalMatrix;
        }
        template<class Problem, class Grid, class Matrix>
        const double DGMethodZero<Problem, Grid, Matrix>::GetValue(const Mesh::Point& p) const
        {
            if (!m_solution.size())
                return -1;
            double val = 0;
            int nfem = -1;
            nfem = m_Grid->FindElement(p);
            if (nfem == -1)
                return -1;
            auto elem = m_Grid->GetElements()[nfem];
            for (int i = 0; i < elem->GetDoF(); ++i)
                val += m_solution[elem->GetNodes()[i]] * elem->GetBasis(i, p);
            return val;
        }
        template<class Problem, class Grid, class Matrix>
        const double DGMethodZero<Problem, Grid, Matrix>::GetValue(const Mesh::Point& p, const
      std::vector<double>& vec) const
        {
            if (!vec.size())
                return -1;
            double val{ 0 };
            int nfem{ -1 };
            nfem = m_Grid->FindElement(p);
            if (nfem == -1)
                return -1;
```

```
        auto elem = m_Grid->GetElements()[nfem];
        for (int i = 0; i < elem->GetDoFs(); ++i)
            val += vec[elem->GetNode(i)] * elem->GetShapeFunction(i, p);
        return val;
    }
    template<class Problem, class Grid, class Matrix>
    const double DGMethodZero<Problem, Grid, Matrix>::GetValue(const Mesh::Point& p, const std::vector<double>& vec, const int num) const
    {
        if (!vec.size() || num < 0)
            return -1;
        double val{ 0 };
        auto elem = m_Grid->GetElements()[num];
        for (int i = 0; i < elem->GetDoF(); ++i)
            val += vec[elem->GetNodes()[i]] * elem->GetBasis(i, p);
        return val;
    }
    //template<class Problem, class Grid, class Matrix>
    //const Mesh::Point DGMethodZero<Problem, Grid, Matrix>::GetGradValue(const Mesh::Point& p, const std::vector<double>& vec) const
    //{
    //  Mesh::Point val{ 0, 0 };
    //  int nfem{ -1 };
    //  nfem = m_Grid->FindElement(p);
    //  if (nfem == -1)
    //      return val;
    //  auto elem = m_Grid->GetElements()[nfem];
    //  for (int i = 0; i < elem->GetDoF(); ++i)
    //  {
    //      val.x += vec[elem->GetNodes()[i]] * elem->GetGradBasis(i, p).x;
    //      val.y += vec[elem->GetNodes()[i]] * elem->GetGradBasis(i, p).y;
    //      val.z += vec[elem->GetNodes()[i]] * elem->GetGradBasis(i, p).z;
    //  }
    //  return val;
    //}
    template<class Problem, class Grid, class Matrix>
    const double DGMethodZero<Problem, Grid, Matrix>::GetEffective(const std::vector<double>& vec) const
    {
        double sum = 0;
        //std::vector<int> dofs;
        //Mesh::Point points[10];
        //for (int i = 0; i < m_Grid->GetElements().size(); ++i)
        //{
            //auto mb = [&](const Mesh::Point& b) {return GetGradValue(b, vec)*GetGradValue(b, vec); };
            //dofs.resize(0);
            //auto elem = m_Grid->GetElements()[i];
            //int order = elem->GetDoF();
            //double diff = std::get<0>(m_Grid->GetDiffusion().find(elem->GetType())->second);
            //for (int j = 0; j < order; ++j)
            //{
                //dofs.push_back(elem->GetNodes()[j]);
                //points[j] = m_Grid->GetNodes()[dofs[j]];
            //}
            //sum += diff * elem->Integrate(mb, points);
        //}
        //std::cout « "Effect (local): " « sum « std::endl;
        //std::cout « "Effect (local) sqrt: " « sqrt(sum) « std::endl;
        return sum;
    }
    //template<class Problem, class Grid, class Matrix>
    //const Mesh::Point DGMethodZero<Problem, Grid, Matrix>::GetLambdaGrad(const Mesh::Point& p, const std::vector<double>& vec) const
    //{
    //  Mesh::Point val{ 0, 0, 0 };
    //  //double val{ 0 };
    //  double diff{ 0 };
    //  Mesh::Point temp{ 0, 0, 0 };
    //  int nfem{ -1 };
    //  nfem = m_Grid->FindElement(p);
    //  if (nfem == -1)
    //      return val;
    //  auto elem = m_Grid->GetElements()[nfem];
    //  diff = std::get<0>(m_Grid->GetDiffusion().find(elem->GetType())->second);
    //  for (int i = 0; i < elem->GetDoF(); ++i)
    //  {
    //      //val += elem->GetGradBasis(i, p) * elem->GetGradBasis(i, p) * vec[elem->GetNodes()[i]] * vec[elem->GetNodes()[i]] * diff;
    //      //val += elem->GetBasis(i, p) * vec[elem->GetNodes()[i]] * diff;
    //      temp = elem->GetGradBasis(i, p);
    //      val.x += temp.x * vec[elem->GetNodes()[i]] * (diff);
    //      val.y += temp.y * vec[elem->GetNodes()[i]] * (diff);
    //      val.z += temp.z * vec[elem->GetNodes()[i]] * (diff);
    //  }
    //  return val;
    //}
```

```
        template<class Problem, class Grid, class Matrix>
        const std::vector<double> DGMethodZero<Problem, Grid, Matrix>::GetRightVector() const
        {
            return *m_rhsvector;
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethodZero<Problem, Grid, Matrix>::OutDatFormat(const Mesh::Point& mn, const Mesh::Point& mx, const std::string& file_name, const std::vector<double>& vec) const
        {
            std::ofstream of(file_name + "z.dat");
            std::streambuf *buf = std::cout.rdbuf();
            std::cout.rdbuf(of.rdbuf());
            std::cout « "TITLE = FE-METHOD\n";
            std::cout « "VARIABLES = \"dx1\", \"dx2\", \"u\"\n";
            std::cout « "ZONE i=51, j=51, F=POINT\n";
            double stepx = (mx.x - mn.x) / 51;
            double stepy = (mx.y - mn.y) / 51;
            for (int i = 0; i < 51; ++i)
                for (int j = 0; j < 51; ++j)
                    std::cout « mn.x + j * stepx « "\t" « mn.y + stepy * i « "\t" « GetValue(Mesh::Point(mn.x + j * stepx, mn.y + i * stepy, mn.z), vec) « std::endl;
            std::cout.rdbuf(buf);
            of.close();
            of.open(file_name + "x.dat");
            buf = std::cout.rdbuf();
            std::cout.rdbuf(of.rdbuf());
            std::cout « "TITLE = FE-METHOD\n";
            std::cout « "VARIABLES = \"dx1\", \"dx2\", \"u\"\n";
            std::cout « "ZONE i=51, j=51, F=POINT\n";
            for (int i = 0; i < 51; ++i)
                for (int j = 0; j < 51; ++j)
                    std::cout « mn.x + j * stepx « "\t" « mn.y + stepy * i « "\t" « GetValue(Mesh::Point(mn.z, mn.x + j * stepx, mn.y + i * stepy), vec) « std::endl;
            std::cout.rdbuf(buf);
            of.close();
            of.open(file_name + "y.dat");
            buf = std::cout.rdbuf();
            std::cout.rdbuf(of.rdbuf());
            std::cout « "TITLE = FE-METHOD\n";
            std::cout « "VARIABLES = \"dx1\", \"dx2\", \"u\"\n";
            std::cout « "ZONE i=51, j=51, F=POINT\n";
            for (int i = 0; i < 51; ++i)
                for (int j = 0; j < 51; ++j)
                    std::cout « mn.x + j * stepx « "\t" « mn.y + stepy * i « "\t" « GetValue(Mesh::Point(mn.x + j * stepx, mn.z, mn.y + i * stepy), vec) « std::endl;
            std::cout.rdbuf(buf);
            of.close();
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethodZero<Problem, Grid, Matrix>::ApplySources()
        {
            int nfem = -1;
            auto total = m_problem->get_total_sources();
            for (int i = 0; i < total; ++i)
            {
                auto src = m_problem->get_point_source(i);
                auto point = src.get_point();
                nfem = m_Grid->FindElement(point);
                if (nfem != -1)
                {
                    auto val = src.get_value();
                    auto elem = m_Grid->GetElement(nfem);
                    for (int j = 0; j < 3; ++j)
                        m_rhsvector->operator[](elem->GetNode(j)) += val * elem->GetShapeFunction(j, point);
                }
                nfem = -1;
            }
            /*for (auto srd : m_Grid->GetDottedSources())
            {
                nfem = m_Grid->FindElement(srd.first);
                if (nfem != -1)
                {
                    auto elem = m_Grid->GetElements()[nfem];
                    for (int i = 0; i < elem->GetDoF(); ++i)
                    {
                        m_rhsvector[elem->GetNodes()[i]] += srd.second * elem->GetBasis(i, srd.first);
                    }
                }
                nfem = -1;
            }*/
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethodZero<Problem, Grid, Matrix>::Rediscretization(const std::shared_ptr<Grid>& grid)
        {
            m_GlobalMatrix->NullMatrix();
            for (unsigned int i = 0; i < m_rhsvector->size(); ++i)
```

```
                (*m_rhsvector)[i] = 0;
            AssemblGlobal();
            //SecondConditions();
            //ApplySources();
            //StefanConditions();
            MainConditions();
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethodZero<Problem, Grid, Matrix>::Rediscretization()
        {
            m_time += m_step;
            m_GlobalMatrix->NullMatrix();
            for (unsigned int i = 0; i < m_rhsvector->size(); ++i)
                (*m_rhsvector)[i] = 0;
            AssemblGlobal();
            SecondConditions();
            ThirdConditions();
            StefanConditions();
            //ApplySources();
            MainConditions();
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethodZero<Problem, Grid, Matrix>::GetSolution(std::vector<double>& vec)
        {
            int size = vec.size();
            //Translation(vec);
            for (int i = 0; i < size; ++i)
                vec[i] = m_solution[i];
        }
        template<class Problem, class Grid, class Matrix>
        const double DGMethodZero<Problem, Grid, Matrix>::GetSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p)
        {
            double sum{ 0 };
            auto nfem{ g.FindElement(p) };
            if (nfem < 0)
                return 0.;
            auto elem{ g.GetElement(nfem) };
            auto dofs{ elem->GetDoFs() };
            for (auto i{ 0 }; i < dofs; ++i)
                sum += weights[elem->GetNode(i)] * elem->GetShapeFunction(i, p);
            return sum;
        }
        template<class Problem, class Grid, class Matrix>
        const double DGMethodZero<Problem, Grid, Matrix>::GetSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p, const int nfem)
        {
            double sum{ 0 };
            //if (nfem < 0)
            //    return 0.;
            auto elem{ g.GetElement(nfem) };
            auto dofs{ elem->GetDoFs() };
            //std::cout « nfem « std::endl;
            for (auto i{ 0 }; i < dofs; ++i)
                sum += weights[elem->GetNode(i)] * elem->GetShapeFunction(i, p);
            return sum;
        }
        template<class Problem, class Grid, class Matrix>
        const Mesh::Point DGMethodZero<Problem, Grid, Matrix>::GetGradSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p)
        {
            Mesh::Point sum{ 0, 0, 0 };
            auto nfem{ g.FindElement(p) };
            auto elem{ g.GetElement(nfem) };
            auto dofs{ elem->GetDoFs() };
            for (auto i{ 0 }; i < dofs; ++i)
                sum += weights[elem->GetNode(i)] * elem->GetGradShapeFunction(i, p);
            return sum;
        }
        template<class Problem, class Grid, class Matrix>
        const Mesh::Point DGMethodZero<Problem, Grid, Matrix>::GetGradSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p, const int nfem)
        {
            Mesh::Point sum{ 0, 0, 0 };
            auto elem{ g.GetElement(nfem) };
            auto dofs{ elem->GetDoFs() };
            for (auto i{ 0 }; i < dofs; ++i)
                sum += weights[elem->GetNode(i)] * elem->GetGradShapeFunction(i, p);
            return sum;
        }
        template<class Problem, class Grid, class Matrix>
        void DGMethodZero<Problem, Grid, Matrix>::LoadSolution(const std::vector<double>& vec)
        {
            m_solution.resize(vec.size());
            for (unsigned int i = 0; i < vec.size(); ++i)
                m_solution[i] = vec[i];
        }
```

```
    template<class Problem, class Grid, class Matrix>
    void DGMethodZero<Problem, Grid, Matrix>::OutMeshFormat(const std::string& file_name, const std::vector<double>& vec)
    {
        const int size{ (int)m_Grid->GetNodes().size() };
        const int number{ (int)m_Grid->GetElements().size() };
        //const int size{ number * 4 };
        std::ofstream ofs(file_name + ".dat", std::ios::out);
        std::string title("TITLE = \"Mesh data\"\n Variables = \"X\", \"Y\", \"Z\", \"U\"\n Zone N = " + std::to_string(size) + ", E = " + std::to_string(number) + ", DATAPACKING = POINT, ZONETYPE = FETETRAHEDRON\n");
        ofs « title;
        Mesh::Point p;
        for (int i = 0; i < size; ++i)
        {
            p = m_Grid->GetNodes()[i];
            ofs « p.x « "\t" « p.y « "\t" « p.z « "\t" « GetValue(p, vec, 1) « std::endl;
        }
        for (int i = 0; i < number; ++i)
        {
            auto elem = m_Grid->GetElements()[i];
            for (int k = 0; k < 4; ++k)
            {
                ofs « elem->GetNodes()[k] + 1 « "\t";
            }
            ofs « std::endl;
        }
        ofs.close();
    }
    template<class Problem, class Grid, class Matrix>
    void DGMethodZero<Problem, Grid, Matrix>::OutMeshTimeFormat(const std::string& file_name, const std::vector<double>& vec)
    {
        const int size{ (int)m_Grid->GetNodes().size() };
        const int number{ (int)m_Grid->GetElements().size() };
        //const int size{ number * 4 };
        std::ofstream ofs(file_name + ".dat", std::ios::out | std::ios::app);
        std::string title("TITLE = \"Mesh data\"\n Variables = \"X\", \"Y\", \"Z\", \"U\"\n Zone N = " + std::to_string(size) + ", E = " + std::to_string(number) + ", DATAPACKING = POINT, ZONETYPE = FETETRAHEDRON\n");
        ofs « title;
        Mesh::Point p;
        for (int i = 0; i < size; ++i)
        {
            p = m_Grid->GetNodes()[i];
            ofs « p.x « "\t" « p.y « "\t" « p.z « "\t" « GetValue(p, vec, 1) « std::endl;
        }
        for (int i = 0; i < number; ++i)
        {
            auto elem = m_Grid->GetElements()[i];
            for (int k = 0; k < 4; ++k)
            {
                ofs « elem->GetNodes()[k] + 1 « "\t";
            }
            ofs « std::endl;
        }
        ofs.close();
    }
    template<class Problem, class Grid, class Matrix>
    void DGMethodZero<Problem, Grid, Matrix>::ProjectSolution(std::vector<double>& sol, std::function<const double(const Mesh::Point&, const std::vector<double>&, const int)> GetVal, std::vector<double>& vec)
    {
        for (int i = 0; i < m_Grid->GetElements().size(); ++i)
        {
            auto elem = m_Grid->GetElements()[i];
            int order = elem->GetDoF();
            for (int j = 0; j < order; ++j)
                sol[elem->GetNodes(j)] = GetVal(m_Grid->GetNodes()[elem->GetNodes(j)], vec, i);
        }
    }
    template<class Problem, class Grid, class Matrix>
    void DGMethodZero<Problem, Grid, Matrix>::ProjectSolution(std::vector<double>& sol, std::function<const double(const Mesh::Point&, const std::vector<double>&)> GetVal, std::vector<double>& vec, const int)
    {
        for (int i = 0; i < m_Grid->GetElements().size(); ++i)
        {
            auto elem = m_Grid->GetElements()[i];
            int order = elem->GetDoF();
            for (int j = 0; j < order; ++j)
                sol[elem->GetNodes(j)] = GetVal(m_Grid->GetNodes()[elem->GetNodes(j)], vec);
        }
    }
    template<class Problem, class Grid, class Matrix>
    const std::vector<double> DGMethodZero<Problem, Grid, Matrix>::SetSolution(const int sol, const int liq, const double s, const double l, const double m)
```

```
        {
            int i;
            m_solution.resize(m_Grid->GetNodes().size());
            for (i = 0; i < m_Grid->GetElements().size(); ++i)
            {
                auto elem = m_Grid->GetElements()[i];
                int order = elem->GetDoF();
                if (m_Grid->GetElements()[i]->GetType() == liq)
                    for (int j = 0; j < order; ++j)
                        m_solution[elem->GetNodes()[j]] = l;
                else
                    for (int j = 0; j < order; ++j)
                        m_solution[elem->GetNodes()[j]] = s;
            }

            for (auto bnd : m_Grid->GetBoundaryConditions())
            {
                //if (get<0>(bnd.second) == 4)
                {
                    for (auto row : m_Grid->GetBoundary())
                    {
                        if (bnd.first == row->GetType())
                        {
                            int dofs = row->GetDoF();
                            for (int i = 0; i < dofs; ++i)
                            {
                                m_solution[row->GetNodes(i)] = m;
                            }
                        }
                    }
                }
            }
            return m_solution;
        }
        template<class Problem, class Grid, class Matrix>
        DGMethodZero<Problem, Grid, Matrix>::~DGMethodZero()
        {
            delete m_Grid;
        }
    }
}

#endif // !CORENC_METHODS_DGMethodZero_h
```

7.75 CoreNCFEM/Methods/DGSolution.h File Reference

```
#include "DGMethod.h"
```

Classes

- class corenc::method::DGSolution< Grid >
- class corenc::method::STSolution< Grid >

Namespaces

- namespace corenc
- namespace corenc::method

7.76 DGSolution.h

Go to the documentation of this file.

```
#ifndef CORENC_METHODS_DGSOLUTION_H_
#define CORENC_METHODS_DGSOLUTION_H_

#include "DGMethod.h"
namespace corenc
{
    namespace method
    {
        template<class Grid>
         class DGSolution
         {
         public:
             DGSolution() {};
             DGSolution(const std::vector<double>& w) :m_w{ w } {}
             DGSolution(const DGSolution<Grid>& dg) :m_w{ dg.m_w } {}
             DGSolution<Grid>& operator=(const DGSolution<Grid>& dg)
             {
                 m_w = dg.m_w;
                 return *this;
             }
             ~DGSolution()
             {
                 if (m_w.size() > 0)
                     std::vector<double>().swap(m_w);
             }
             const double                getWeight(const Grid& g, const Mesh::Point& p) const
             {
                 if (m_w.size() > 0)
                     return DGMethod<int, Grid, int>::GetSolution(g, m_w, p);
                 return 0.;
             };
             const std::vector<double>   getWeights() const { return m_w; }
             const int                   updateWeight(const unsigned int i, const double val)
             {
                 if (i < m_w.size())
                 {
                     m_w[i] = val;
                     return 0;
                 }
                 return 1;
             }
         private:
             std::vector<double>         m_w;
         };
         template<class Grid>
         class STSolution
         {
         public:
             STSolution() {};
             STSolution(const Grid& g):m_grid{g}{}
             STSolution(
                 const std::vector<DGSolution<Grid>>& w,
                 const std::vector<double> time,
                 const Grid& g) : m_w{ w }, m_time{ time }, m_grid{g} {}
             STSolution(const STSolution<Grid>& st) :m_w{ st.m_w }, m_time{ st.m_time }, m_grid{st.m_grid} {}
             STSolution<Grid>& operator=(const STSolution<Grid>& st)
             {
                 m_w = st.m_w;
                 m_time = st.m_time;
                 m_grid = st.m_grid;
                 return *this;
             }
             ~STSolution()
             {
                 if (m_w.size() > 0)
                     std::vector<DGSolution<Grid»().swap(m_w);
                 if (m_time.size() > 0)
                     std::vector<double>().swap(m_time);
             }
             const double                getWeight(const Mesh::Point& p, const double time) const
             {
                 int i = 0;
                 auto sz = m_time.size();
                 if (fabs(time) < 1e-14)
                     return DGMethod<Grid>::GetSolution(m_grid, m_w[0].getWeights(), p);
                 for (; i < sz; ++i)
                 {
                     if (time < m_time[i])
                         break;
                 }
                 if (i == sz)
```

```
                    --i;
                double dt = m_time[i] - m_time[i - 1];
                auto temp = DGMethod<Grid>::GetSolution(m_grid, m_w[i - 1].getWeights(), p);
                double du = DGMethod<Grid>::GetSolution(m_grid, m_w[i].getWeights(), p) - temp;
                return temp + du * (time - m_time[i - 1]) / dt;
            };
            const int            updateWeight(
                const std::vector<double> time,
                const std::vector<DGSolution<Grid>> w
            )
            {
                m_time = time;
                m_w = w;
            }
            const int            addTimeLayer(
                const double time,
                const DGSolution<Grid> w
            )
             {
                 m_time.push_back(time);
                 m_w.push_back(w);
                 return 0;
             }
             const std::vector<DGSolution<Grid»     getWeights() const { return m_w; }
        private:
             std::vector<DGSolution<Grid»                          m_w;
             std::vector<double>                                    m_time;
             Grid                                                   m_grid;
        };
    }
}
#endif // !CORENC_METHODS_DGSOLUTION_H_
```

7.77 CoreNCFEM/Methods/FEAnalysis.h File Reference

```
#include <vector>
#include "../Point.h"
```

Classes

- class corenc::method::FEAnalysis< Method1, Method2, Mesh1, Mesh2 >

Namespaces

- namespace corenc
- namespace corenc::method

Macros

- #define CORENC_METHODS_FEANALYSIS_H_

7.77.1 Macro Definition Documentation

7.77.1.1 CORENC_METHODS_FEANALYSIS_H_

```
#define CORENC_METHODS_FEANALYSIS_H_
```

7.78 FEAnalysis.h

Go to the documentation of this file.

```
#pragma once
#ifndef CORENC_METHODS_FEANALYSIS_H_
#define CORENC_METHODS_FEANALYSIS_H_
#include <vector>
#include "../Point.h"
namespace corenc
{
    namespace method
    {
        template<class Method1, class Method2, class Mesh1, class Mesh2>
        class FEAnalysis
        {
        public:
            FEAnalysis() {};
            ~FEAnalysis() {};
            const double                    L2Norm(   const Method1& method1,
                                                    const Method2& method2,
                                                    const Mesh1& mesh1,
                                                    const Mesh2& mesh2,
                                                    const std::vector<double>& w1,
                                                    const std::vector<double>& w2) const;
        };
        template<class Method1, class Method2, class Mesh1, class Mesh2>
        const double FEAnalysis<Method1, Method2, Mesh1, Mesh2>::L2Norm(
            const Method1& method1,
            const Method2& method2,
            const Mesh1& mesh1,
            const Mesh2& mesh2,
            const std::vector<double>& w1,
            const std::vector<double>& w2) const
        {
            double sum{ 0 }, sum2{0};
            double res, res2;
            int j;
            std::vector<int> dofs;
            int order = mesh1.GetElement(0)->GetDoFs();
            dofs.resize(order);
            std::vector<Mesh::Point> points(order);
            auto sub = [&](const Mesh::Point& p)
            {
                return (method1.GetValue(p, w1) - method2.GetValue(p, w2)) * (method1.GetValue(p, w1) -
      method2.GetValue(p, w2));
            };
            auto r = [&](const Mesh::Point& p)
            {
                return method1.GetValue(p, w1);
            };
            const int n = (int)mesh1.GetNumberOfElements();
            for (int i = 0; i < n; ++i)
            {
                const auto& elem = mesh1.GetElement(i);
                for (j = 0; j < order; ++j)
                    points[j] = mesh1.GetNode(elem->GetNode(j));
                res = elem->Integrate(sub, points);
                res2 = elem->Integrate(r, points);
                sum += res;
                sum2 += res2;
            }
            if (dofs.size() > 0)
                std::vector<int>().swap(dofs);
            if (points.size() > 0)
                std::vector<Mesh::Point>().swap(points);
            return sqrt(sum/sum2);
        }
    }
}

#endif // !CORENC_METHODS_FEANALYSIS_H_

```

7.79 CoreNCFEM/Methods/FEMethod.h File Reference

```
#include <functional>
#include <set>
#include "../Point.h"
```

```
#include "../Parameter.h"
#include "CSMethod.h"
#include <memory>
#include <cmath>
#include <map>
#include <algorithm>
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
```

Classes

- class corenc::method::CFEMethod< Type >
- class corenc::method::FEMethod< Problem, Grid, Matrix >

Namespaces

- namespace corenc
- namespace corenc::Mesh
- namespace corenc::method

Enumerations

- enum class corenc::method::BoundaryType { corenc::method::MAIN , corenc::method::SECOND , corenc::method::THIRD , corenc::method::FREE }

7.80 FEMethod.h

Go to the documentation of this file.

```
// FEMethod.h describes an abstract interface and functions for a general finite element method

#ifndef CORENC_METHODS_FEMethod_h
#define CORENC_METHODS_FEMethod_h
#include <functional>
#include <set>
#include "../Point.h"
#include "../Parameter.h"
#include "CSMethod.h"
#include <memory>
#include <cmath>
#include <map>
#include <algorithm>
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
namespace corenc
{
    namespace Mesh
    {
        class Point;
    }
    namespace method
    {
        enum class BoundaryType
        {
            MAIN,
            SECOND,
            THIRD,
            FREE
```

```
    };
    // class Type = Type of the solution, for ex vector or double, or even more specific


    template<class Type>
    class CFEMethod
    {
    public:
        CFEMethod() {};
        virtual ~CFEMethod() {};
        virtual const int                           Assemble() = 0;
        virtual const Type                          GetSolution(const std::vector<double>& point) const = 0;
        virtual const std::vector<Type>             GetSolution() const = 0;
        virtual const Type                          GetMaxSolution() const = 0;
        virtual const Type                          GetMinSolution() const = 0;
    };

    template<class Problem, class Grid, class Matrix>
    class FEMethod
    {
    public:
        FEMethod() :
            m_problem{nullptr},
            m_Grid{nullptr},
            m_GlobalMatrix{nullptr},
            m_RightMatrix{nullptr},
            m_rhsvector{nullptr}
        {}
        FEMethod(
            Problem* p,
            Grid* g,
            Matrix* m,
            std::vector<double>* rhs):
            m_problem{ p },
            m_Grid{ g->Clone() },
            m_GlobalMatrix{ m },
            m_N{ g->GetNumberOfElements() },
            m_Ns{ g->GetNumberOfBoundaries() },
            m_rhsvector{ rhs }{
            //GeneratePortrait();
        }
        FEMethod(
            Problem* p,
            Grid* g,
            Matrix* m,
            Matrix* rm,
            std::vector<double>* rhs):
            m_problem{ p },
            m_Grid{ g->Clone() },
            m_GlobalMatrix{ m },
            m_RightMatrix{ rm },
            m_N{ g->GetNumberOfElements() },
            m_Ns{ g->GetNumberOfBoundaries() },
            m_rhsvector{ rhs }{
            //GeneratePortrait();
        }
        FEMethod(const std::shared_ptr<Grid>& grid) :m_Grid{ grid->Clone() } {}
        FEMethod(Grid* grid) :m_Grid{ grid->Clone() } {}
        FEMethod(const FEMethod& meth) :
            m_Grid{ meth.m_Grid->Clone() },
            //m_GlobalMatrix{ meth.m_GlobalMatrix->Clone() },
            //m_rhsvector{ meth.m_rhsvector },
            //m_problem{ meth.m_problem },
            m_time{ meth.m_time },
            //m_solution{ meth.m_solution },
            m_size{ meth.m_size },
            m_N{ meth.m_N },
            m_Ns{ meth.m_Ns },
             m_nums{ meth.m_nums }
         {};
         FEMethod&                   operator=(const FEMethod& fem)
         {
             m_Grid = fem.m_Grid->Clone();
             m_time = fem.m_time;
             m_size = fem.m_size;
             m_N = fem.m_N;
             m_Ns = fem.m_Ns;
             m_nums = fem.m_nums;
             return *this;
         }
         void                        Discretization();
         const double                GetValue(const Mesh::Point&) const;
         const double                GetValue(const Mesh::Point&, const std::vector<double>& vec) const;
         const double                GetValue(const Mesh::Point&, const std::vector<double>& vec, const int num) const;
```

```
        //const Mesh::Point         GetGradValue(const Mesh::Point&, const std::vector<double>& vec) const;
        //const Mesh::Point         GetLambdaGrad(const Mesh::Point&, const std::vector<double>& vec) const;
        const double                GetEffective(const std::vector<double>& vec) const;
        void                        ProjectSolution(std::vector<double>&, std::function<const double(const Mesh::Point&, const std::vector<double>&, const int)> GetValue, std::vector<double>& sol);
        void                        ProjectSolution(std::vector<double>&, std::function<const double(const Mesh::Point&, const std::vector<double>&)> GetValue, std::vector<double>& sol, const int);
        void                        LoadSolution(const std::vector<double>& vec);
        const std::vector<double>   SetSolution(const int sol, const int liq, const double, const double, const double);
        void                        GetSolution(std::vector<double>& vec);
        void                        Rediscretization(const std::shared_ptr<Grid>&);
        void                        Rediscretization();
        void                        SetTimeStep(const double& step) { m_step = step; m_time = step; }
        Matrix*                     GetGlobalMatrix() const;
        Grid*                       GetMesh() { return m_Grid; }
        const std::vector<double>   GetRightVector() const;
        void                        OutDatFormat(const Mesh::Point& min, const Mesh::Point& max, const std::string& file_name, const std::vector<double>& vec) const;
        void                        OutMeshFormat(const std::string& file_name, const std::vector<double>& vec);
        void                        OutMeshTimeFormat(const std::string& file_name, const std::vector<double>& vec);
        static const double         GetSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p);
        static const double         GetSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p, const int nfem);
        static const Mesh::Point GetGradSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p);
        static const Mesh::Point GetGradSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p, const int n);
        ~FEMethod();
    private:
        void                        GeneratePortrait();
        void                        AssemblGlobal();
        void                        MainConditions();
        void                        SecondConditions();
        void                        ThirdConditions();
        void                        StefanConditions();
        void                        ApplySources();
        const int                   AssembleLocalMatrix(const int);
        Grid*                       m_Grid = nullptr;
        Matrix*                     m_GlobalMatrix = nullptr;
        Matrix*                 m_RightMatrix = nullptr;
        Problem*                    m_problem = nullptr;
        std::vector<double>         m_solution;
        std::vector<double>*        m_rhsvector;
        unsigned int                m_size;
        double                      m_step{ 0.1 };
        double                      m_time{ 0.1 };
        unsigned int                m_N;
        unsigned int                m_Ns;
        std::vector<unsigned int>   m_nums;

    };

    template<class Problem, class Grid, class Matrix>
    void FEMethod<Problem, Grid, Matrix>::Discretization()
    {
        GeneratePortrait();
        AssemblGlobal();
        //ApplySources();
        //SecondConditions();
        //ThirdConditions();
        MainConditions();
        //StefanConditions();
    }
    template<class Problem, class Grid, class Matrix>
    void FEMethod<Problem, Grid, Matrix>::GeneratePortrait()
    {
        const auto& el = m_Grid->GetElement(0);
        int order = m_Grid->GetElement(0)->GetDoFs();
        std::vector<std::set<unsigned int» temp;
        m_Ns = m_Grid->GetNumberOfNodes();
        m_N = m_Grid->GetNumberOfElements();
        temp.resize(m_Grid->GetNumberOfNodes());
        unsigned i, j, k;
        for (k = 0; k < m_N; ++k)
        {
            const auto& elem{ m_Grid->GetElement(k) };
            for (i = 0; i < order; ++i)
                for (j = 0; j < order; ++j)
```

```
                        if (elem->GetNode(j) > elem->GetNode(i))
                            temp[elem->GetNode(j)].insert(elem->GetNode(i));
            }
            if(m_problem->findTerm(Terms::RUV))
                m_RightMatrix->Create(temp.size(), temp);

            //m_GlobalMatrix = std::shared_ptr<Matrix>(new Matrix(m_Grid->GetNumberOfNodes(), temp));
            //m_rhsvector.resize(m_Grid->GetNumberOfNodes());
            //std::cout « temp.size() « std::endl;
            m_GlobalMatrix->Create(temp.size(), temp);
            m_rhsvector->resize(temp.size());
            //m_solution.resize(m_Grid->GetNumberOfNodes());
            //for (int l = 0; l < m_Grid->GetNumberOfNodes(); ++l)
            //  m_solution[l] = 20;
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethod<Problem, Grid, Matrix>::AssemblGlobal()
        {
            int l;
            //std::vector<std::future<int» futures;
            for (l = 0; l < m_N; ++l)
                //futures.push_back(async(&DGMethod<Problem, Grid, Matrix>::AssembleLocalMatrix, this, l));
                AssembleLocalMatrix(l);
            //for (auto &it : futures)
            //it.get();
        }
        template<class Problem, class Grid, class Matrix>
        const int FEMethod<Problem, Grid, Matrix>::AssembleLocalMatrix(const int l)
        {
            int i, j, k, nodes;
            double mij;
            const auto& elem{ m_Grid->GetElement(l) };
            const int dofs{ (int)elem->GetDoFs() };
            const int terms{ (int)m_problem->getNumberOfTerms() };
            nodes = elem->GetNumberOfNodes();
            std::vector<Mesh::Point> points(nodes);
            for (i = 0; i < nodes; ++i)
                points[i] = m_Grid->GetNode(elem->GetNode(i));
            for (k = 0; k < terms; ++k)
            {
                switch (m_problem->getTerm(k))
                {
                case Terms::IUV:
                    for (i = 0; i < (int)dofs; ++i)
                    {
                        for (j = 0; j < (int)dofs; ++j)
                        {
                            auto M = [&](const Mesh::Point& p)
                            {
                                return m_problem->get_parameter(Terms::IUV, l, elem->GetType(), p) * elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                            };
                            mij = elem->Integrate(M, points);
                            m_GlobalMatrix->AddElement(elem->GetNode(i), elem->GetNode(j), mij);
                        }
                    }
                    break;
                case Terms::IDUDV:
                    for (i = 0; i < (int)dofs; ++i)
                    {
                        for (j = 0; j < (int)dofs; ++j)
                        {
                            auto M = [&](const Mesh::Point& p)
                            {
                                //auto m = elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                                return m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p) * elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                            };
                            //mij = m_Grid->getParameter(Parameters::DIFFUSION, l, j) * elem->Integrate(M, points);
                            mij = elem->Integrate(M, points);
                            m_GlobalMatrix->AddElement(elem->GetNode(i), elem->GetNode(j), mij);
                        }
                    }
                    break;
                case Terms::IDUV:
                    for (i = 0; i < (int)dofs; ++i)
                    {
                        for (j = 0; j < (int)dofs; ++j)
                        {
                            auto M = [&](const Mesh::Point& p)
                            {
                                return m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * elem->GetShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                            };
```

```
                        auto _mij = elem->Integrate(M, points);
                        m_GlobalMatrix->AddElement(elem->GetNode(i), elem->GetNode(j), _mij);
                    }
                }
                break;
            case Terms::IUDV:
                for (i = 0; i < dofs; ++i)
                {
                    for (j = 0; j < dofs; ++j)
                    {
                        auto M = [&](const Mesh::Point& p)
                        {
                            return elem->GetGradShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                        };
                        //mij = m_CoarseGrid->getParameter(Parameters::ADVECTION, l, j) * m_flux(m_CoarseGrid->getSolution(l, j)) * elem->Integrate(M, points).x;
                        mij = elem->Integrate(M, points).x;
                        m_GlobalMatrix->AddElement(elem->GetNode(i), elem->GetNode(j), mij);
                    }
                }
                break;
            case Terms::EUV:
                for (i = 0; i < dofs; ++i)
                {
                    for (j = 0; j < dofs; ++j)
                    {
                        auto M = [&](const Mesh::Point& p)
                        {
                            return elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                        };
                        mij = elem->Integrate(M, points);
                        m_rhsvector->operator[](elem->GetNode(i)) += m_Grid->getParameter(Parameters::MASS, l, j) * m_Grid->getSolution(l, j) * mij;
                        //m_rhsvector->operator[](m_nums[l] + i) += m_CoarseGrid->getParameter(Parameters::MASS, l, points[j]) * elem->GetValue(j) * mij;
                    }
                }
                break;
            case Terms::EDUDV:
                for (i = 0; i < dofs; ++i)
                {
                    for (j = 0; j < dofs; ++j)
                    {
                        auto M = [&](const Mesh::Point& p)
                        {
                            return elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                        };
                        mij = elem->Integrate(M, points);
                        m_rhsvector->operator[](elem->GetNode(i)) += m_Grid->getParameter(Parameters::DIFFUSION, l, j) * m_Grid->getSolution(l, j) * mij;
                    }
                }
                break;
            case Terms::EDUV:
                for (i = 0; i < dofs; ++i)
                {
                    for (j = 0; j < dofs; ++j)
                    {
                        auto M = [&](const Mesh::Point& p)
                        {
                            return elem->GetShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                        };
                        mij = elem->Integrate(M, points).x;
                        m_rhsvector->operator[](elem->GetNode(i)) += m_Grid->getParameter(Parameters::ADVECTION, l, j) * mij;
                    }
                }
                break;
            case Terms::EUDV:
                for (i = 0; i < dofs; ++i)
                {
                    for (j = 0; j < dofs; ++j)
                    {
                        auto M = [&](const Mesh::Point& p)
                        {
                            return elem->GetGradShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                        };
                        mij = elem->Integrate(M, points).x;
                        m_rhsvector->operator[](elem->GetNode(i)) += m_Grid->getParameter(Parameters::ADVECTION, l, j) * mij;// *mij;
                    }
                }
                break;
            case Terms::EFV:
                for (i = 0; i < dofs; ++i)
                {
```

```
                    /*for (j = 0; j < dofs; ++j)
                    {
                        auto M = [&](const Mesh::Point& p)
                        {
                            return m_problem->get_parameter(Terms::EFV, elem->GetType(), l, j, p) * elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                        };
                        mij = elem->Integrate(M, points);
                        m_rhsvector->operator[](elem->GetNode(i)) += mij;
                    }*/
                    auto M = [&](const Mesh::Point& p)
                    {
                        return m_problem->get_parameter(Terms::EFV, elem->GetType(), l, i, p) * elem->GetShapeFunction(i, p);
                    };
                    mij = elem->Integrate(M, points);
                    m_rhsvector->operator[](elem->GetNode(i)) += mij;
                }
                break;
            case Terms::RUV:
                for (i = 0; i < (int)dofs; ++i)
                {
                    for (j = 0; j < (int)dofs; ++j)
                    {
                        auto M = [&](const Mesh::Point& p)
                        {
                            return elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                        };
                        mij = elem->Integrate(M, points);
                        m_RightMatrix->AddElement(elem->GetNode(i), elem->GetNode(j), mij);
                    }
                }
                break;
            case Terms::SUPG:
            {
                for (i = 0; i < (int)dofs; ++i)
                {
                    for (j = 0; j < (int)dofs; ++j)
                    {
                        /*auto inode = m_Grid->interpolate(elem->GetNode(i));
                        auto jnode = m_Grid->interpolate(elem->GetNode(j));
                        if (inode == -1 || jnode == -1)
                            continue;*/
                        auto M = [&](const Mesh::Point& p)
                        {
                            double vel = sqrt(m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0));
                            double h = elem->GetMeasure();
                            //double Pe = vel * h / 6. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                            double tau = 0.;
                            double Pe = vel * h / 6. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                            //double beta = h / 2. / vel * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                            //double beta = h / std::sqrt(3.) * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                            //double beta = h / 2. * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                            //double beta = h / 2. * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                            //beta = 0.;
                            //for (int ii = 0; ii < (int)dofs; ++ii)
                                //beta += m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * elem->GetGradShapeFunction(ii, p);
                            //return beta * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) *
                            //        elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                            if (Pe >= 1.)
                                tau = h / 2. / vel;
                            else
                                tau = h * h / 12. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                            //return 0.;
                            return tau * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) *
                                elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                        };

                        //double tau =
                        auto _mij = elem->Integrate(M, points);
                        m_GlobalMatrix->AddElement(elem->GetNode(i), elem->GetNode(j), _mij);
                    }
                    auto M = [&](const Mesh::Point& p)
                    {
```

```
                        double vel = sqrt(m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0));
                        double h = elem->GetMeasure();
                        double Pe = vel * h / 6. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                        double tau = 0.;
                        //double Pe = vel * h / 2. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);

                        if (Pe >= 1.)
                            tau = h / 2. / vel;
                        else
                            tau = h * h / 12. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                        auto supg = tau * m_problem->get_parameter(Terms::EFV, elem->GetType(), l, i, p) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * elem->GetGradShapeFunction(i, p);

                        return supg;
                    };
                    mij = elem->Integrate(M, points);
                    m_rhsvector->operator[](elem->GetNode(i)) += mij;
                }
            }
                break;
            default:
                break;
            }
        }
        return 0;
    }
    template<class Problem, class Grid, class Matrix>
    void FEMethod<Problem, Grid, Matrix>::MainConditions()
    {
        double mu{ 1e10 };
        const auto n = m_problem->get_number_of_boundaries();
        const auto m = m_Grid->GetNumberOfBoundaries();
        for (int i = 0; i < n; ++i)
        {
            const auto& type = m_problem->get_boundary_type(i);
            for (int j = 0; j < m; ++j)
            {
                const auto& row = m_Grid->GetBoundary(j);
                if (row->GetType() == type)
                {
                    const int dofs = (int)row->GetDoFs();
                    const int dofs2 = 2;
                    const auto& elem_num = row->GetNeighbour(0);
                    const auto& elem = m_Grid->GetElement(elem_num);
                    const int dofs_elem = elem->GetDoFs();
                    std::vector<Mesh::Point> points(dofs_elem);
                    for (int k = 0; k < dofs_elem; ++k)
                        points[k] = m_Grid->GetNode(elem->GetNode(k));
                    for (int k = 0; k < dofs; ++k)
                    {
                        int l = 0;
                        for (; l < dofs_elem; ++l)
                        {
                            if (elem->GetNode(l) == row->GetNode(k))
                                break;
                        }
                        m_GlobalMatrix->NullRow(row->GetNode(k));
                        m_GlobalMatrix->operator()(row->GetNode(k), row->GetNode(k)) = 0;
                        //m_GlobalMatrix->operator()(row->GetNode(k), row->GetNode(k)) *= mu;
                        //m_rhsvector->operator[](row->GetNode(k)) = m_problem->get_boundary_parameter(0, type, m_Grid->GetNode(row->GetNode(k)));
                        //m_rhsvector->operator[](row->GetNode(k)) = m_problem->get_boundary_parameter(0, type, elem_num, l, m_Grid->GetNode(row->GetNode(k)));
                        m_rhsvector->operator[](row->GetNode(k)) = elem->GetWeight(l, points, [=](const Mesh::Point& p) { return m_problem->get_boundary_parameter(0, type, p); });
                        if(m_problem->findTerm(Terms::RUV))
                            {
                                m_RightMatrix->NullRow(row->GetNode(k));
                                //m_RightMatrix->operator()(row->GetNode(k), row->GetNode(k)) *= mu;
                            }
                    }
                    /*for (int k = dofs2; k < dofs; ++k)
                    {
                        m_GlobalMatrix->NullRow(row->GetNode(k));
                        m_rhsvector->operator[](row->GetNode(k)) = 0;
                    }*/
                }
            }
        }
        /*for (auto bnd : m_Grid->GetBoundaryConditions())
        {
            if (get<0>(bnd.second) == 1)
```

```
                    for (auto row : m_Grid->GetBoundary())
                    {
                        if (bnd.first == row->GetType())
                        {
                            for (int i = 0; i < row->GetDoF(); ++i)
                            {
                                m_GlobalMatrix->NullRow(row->GetNodes(i));
                                m_rhsvector[row->GetNodes(i)] = get<1>(bnd.second)(m_Grid->GetNodes()[row->GetNodes(i)]);
                            }
                        }
                    }
            }*/
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethod<Problem, Grid, Matrix>::SecondConditions()
        {
            double theta = 0;
            int nfem;
            Mesh::Point temp[3];
            std::vector<int> local;
            for (auto bnd : m_Grid->GetBoundaryConditions())
            {
                //if (get<0>(bnd.second) == 2)
                {
                    for (auto row : m_Grid->GetBoundary())
                    {
                        if (bnd.first == row->GetType())
                        {
                            local.resize(0);
                            int dofs = row->GetDoF();
                            nfem = row->GetNumberOfElement(0);
                            auto elem = m_Grid->GetElements()[nfem];
                            //auto GetBasis = [&](int t, Point p){return elem->GetBasis(t, p); };
                            for (int j = 0; j < dofs; ++j)
                            {
                                temp[j] = m_Grid->GetNodes()[row->GetNodes(j)];
                                for (int i = 0; i < elem->GetDoF(); ++i)
                                {
                                    if (row->GetNodes(j) == elem->GetNodes()[i])
                                    {
                                        local.push_back(i);
                                        break;
                                    }
                                }
                            }
                            for (int i = 0; i < dofs; ++i)
                            {
                                for (int j = 0; j < dofs; ++j)
                                {
                                    //theta = get<1>(bnd.second)(m_Grid->GetNodes()[row->GetNodes(i)]);
                                    theta = 0;
                                    auto GetMass = [&](const Mesh::Point& p) {return elem->GetBasis(local[j], p) * elem->GetBasis(local[i], p); };
                                    auto GetBBasis = [&](const Mesh::Point& p) {return row->GetBasis(j, p)*row->GetBasis(i, p); };
                                    //if (i < 2 || j < 2)
                                    m_rhsvector[row->GetNodes(i)] += theta * row->Integrate(GetMass, temp);

                                    //if (i < 3 || j < 3)
                                    //  m_rhsvector[row[i + 1]] += theta * row->Integrate(GetBBasis, temp);
                                }
                            }
                        }
                    }
                }
            }
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethod<Problem, Grid, Matrix>::StefanConditions()
        {
            double dest{ 0. }, lat{ 0 };
            int nfem;
            Mesh::Point temp[3];
            std::vector<int> local;
            for (auto bnd : m_Grid->GetBoundaryConditions())
            {
                //if (get<0>(bnd.second) == 4)
                {
                    lat = 0;
                    //lat = get<2>(bnd.second);
                    for (auto row : m_Grid->GetBoundary())
                    {
                        if (bnd.first == row->GetType())
                        {
```

```
                        local.resize(0);
                        int dofs = row->GetDoF();
                        nfem = row->GetNumberOfElement(0);
                        auto elem = m_Grid->GetElements()[nfem];
                        //auto GetBasis = [&](int t, Point p){return elem->GetBasis(t, p); };
                        for (int j = 0; j < dofs; ++j)
                        {
                            temp[j] = m_Grid->GetNodes()[row->GetNodes(j)];
                            for (int i = 0; i < elem->GetDoF(); ++i)
                            {
                                if (row->GetNodes(j) == elem->GetNodes()[i])
                                {
                                    local.push_back(i);
                                    break;
                                }
                            }
                        }
                        for (int i = 0; i < dofs; ++i)
                        {
                            for (int j = 0; j < dofs; ++j)
                            {
                                dest = 0;
                                //dest = get<1>(bnd.second)(m_Grid->GetNodes()[row->GetNodes(i)]);
                                auto GetBBasis = [&](const Mesh::Point& p) {return row->GetBasis(j, p)*row->GetBasis(i, p); };
                                //if (i < 2 || j < 2)
                                m_rhsvector[row->GetNodes(i)] += dest * lat * row->Integrate(GetBBasis, temp);

                                //if (i < 3 || j < 3)
                                //  m_rhsvector[row[i + 1]] += theta * row->Integrate(GetBBasis, temp);
                            }
                        }
                    }
                }
            }
        }
    }
    template<class Problem, class Grid, class Matrix>
    void FEMethod<Problem, Grid, Matrix>::ThirdConditions()
    {
        double param{ 0 }, beta{ 0 };
        int nfem;
        Mesh::Point temp[6];
        std::vector<int> local;
        auto fxy = [&](const Mesh::Point& p) {return (10 * p.y*m_time + m_time) / 10; };
        //auto fxy = [&](const Point& p){return 10 * p.y + 10 * m_time; };
        for (auto bnd : m_Grid->GetBoundaryConditions())
        {
            //if (get<0>(bnd.second) == 3)
            {

                for (auto row : m_Grid->GetBoundary())
                {
                    if (bnd.first == row->GetType())
                    {
                        local.resize(0);
                        int dofs = row->GetDoF();
                        nfem = row->GetNumberOfElement(0);
                        auto elem = m_Grid->GetElements()[nfem];
                        //auto GetBasis = [&](int t, Point p){return elem->GetBasis(t, p); };
                        auto order = elem->GetDoF();
                        for (int j = 0; j < dofs; ++j)
                        {
                            temp[j] = m_Grid->GetNodes()[row->GetNodes(j)];
                            for (int i = 0; i < order; ++i)
                            {
                                if (row->GetNodes(j) == elem->GetNodes()[i])
                                {
                                    local.push_back(i);
                                    break;
                                }
                            }
                        }
                        double val{ 0 };
                        for (int i = 0; i < dofs; ++i)
                        {
                            for (int j = 0; j < dofs; ++j)
                            {
                                param = 0;
                                beta = 0;
                                //beta = get<2>(bnd.second);
                                //param = get<1>(bnd.second)(m_Grid->GetNodes()[row->GetNodes(i)]);
                                //param = fxy(temp[j]);
                                auto GetBBasis = [&](const Mesh::Point& p) {return elem->GetBasis(local[j], p)*elem->GetBasis(local[i], p); };
```

```
                                //val = row->GetElement(GetBBasis, temp);
                                val = row->Integrate(GetBBasis, temp);
                                m_GlobalMatrix->operator()(row->GetNodes(i), row->GetNodes(j)) += beta * val;
                                m_rhsvector[row->GetNodes(i)] += beta * param * val;
                            }
                        }
                    }
                }
            }
        }
    }
    template<class Problem, class Grid, class Matrix>
    Matrix* FEMethod<Problem, Grid, Matrix>::GetGlobalMatrix() const
    {
        return m_GlobalMatrix;
    }
    template<class Problem, class Grid, class Matrix>
    const double FEMethod<Problem, Grid, Matrix>::GetValue(const Mesh::Point& p) const
    {
        if (!m_solution.size())
            return -1;
        double val = 0;
        int nfem = -1;
        nfem = m_Grid->FindElement(p);
        if (nfem == -1)
            return -1;
        auto elem = m_Grid->GetElements()[nfem];
        for (int i = 0; i < elem->GetDoF(); ++i)
            val += m_solution[elem->GetNodes()[i]] * elem->GetBasis(i, p);
        return val;
    }
    template<class Problem, class Grid, class Matrix>
    const double FEMethod<Problem, Grid, Matrix>::GetValue(const Mesh::Point& p, const std::vector<double>& vec) const
    {
        if (!vec.size())
            return -1;
        double val{ 0 };
        int nfem{ -1 };
        nfem = m_Grid->FindElement(p);
        if (nfem == -1)
            return -1;
        auto elem = m_Grid->GetElements()[nfem];
        for (int i = 0; i < elem->GetDoFs(); ++i)
            val += vec[elem->GetNode(i)] * elem->GetShapeFunction(i, p);
        return val;
    }
    template<class Problem, class Grid, class Matrix>
    const double FEMethod<Problem, Grid, Matrix>::GetValue(const Mesh::Point& p, const std::vector<double>& vec, const int num) const
    {
        if (!vec.size() || num < 0)
            return -1;
        double val{ 0 };
        auto elem = m_Grid->GetElements()[num];
        for (int i = 0; i < elem->GetDoF(); ++i)
            val += vec[elem->GetNodes()[i]] * elem->GetBasis(i, p);
        return val;
    }
    //template<class Problem, class Grid, class Matrix>
    //const Mesh::Point FEMethod<Problem, Grid, Matrix>::GetGradValue(const Mesh::Point& p, const std::vector<double>& vec) const
    //{
    //  Mesh::Point val{ 0, 0 };
    //  int nfem{ -1 };
    //  nfem = m_Grid->FindElement(p);
    //  if (nfem == -1)
    //      return val;
    //  auto elem = m_Grid->GetElements()[nfem];
    //  for (int i = 0; i < elem->GetDoF(); ++i)
    //  {
    //      val.x += vec[elem->GetNodes()[i]] * elem->GetGradBasis(i, p).x;
    //      val.y += vec[elem->GetNodes()[i]] * elem->GetGradBasis(i, p).y;
    //      val.z += vec[elem->GetNodes()[i]] * elem->GetGradBasis(i, p).z;
    //  }
    //  return val;
    //}
    template<class Problem, class Grid, class Matrix>
    const double FEMethod<Problem, Grid, Matrix>::GetEffective(const std::vector<double>& vec) const
    {
        double sum = 0;
        //std::vector<int> dofs;
        //Mesh::Point points[10];
        //for (int i = 0; i < m_Grid->GetElements().size(); ++i)
        //{
            //auto mb = [&](const Mesh::Point& b) {return GetGradValue(b, vec)*GetGradValue(b, vec);
```

```
};
                //dofs.resize(0);
                //auto elem = m_Grid->GetElements()[i];
                //int order = elem->GetDoF();
                //double diff = std::get<0>(m_Grid->GetDiffusion().find(elem->GetType())->second);
                //for (int j = 0; j < order; ++j)
                //{
                    //dofs.push_back(elem->GetNodes()[j]);
                    //points[j] = m_Grid->GetNodes()[dofs[j]];
                //}
                //sum += diff * elem->Integrate(mb, points);
            //}
            //std::cout « "Effect (local): " « sum « std::endl;
            //std::cout « "Effect (local) sqrt: " « sqrt(sum) « std::endl;
            return sum;
        }
        //template<class Problem, class Grid, class Matrix>
        //const Mesh::Point FEMethod<Problem, Grid, Matrix>::GetLambdaGrad(const Mesh::Point& p, const std::vector<double>& vec) const
        //{
        //  Mesh::Point val{ 0, 0, 0 };
        //  //double val{ 0 };
        //  double diff{ 0 };
        //  Mesh::Point temp{ 0, 0, 0 };
        //  int nfem{ -1 };
        //  nfem = m_Grid->FindElement(p);
        //  if (nfem == -1)
        //      return val;
        //  auto elem = m_Grid->GetElements()[nfem];
        //  diff = std::get<0>(m_Grid->GetDiffusion().find(elem->GetType())->second);
        //  for (int i = 0; i < elem->GetDoF(); ++i)
        //  {
        //      //val += elem->GetGradBasis(i, p) * elem->GetGradBasis(i, p) * vec[elem->GetNodes()[i]] * vec[elem->GetNodes()[i]] * diff;
        //      //val += elem->GetBasis(i, p) * vec[elem->GetNodes()[i]] * diff;
        //      temp = elem->GetGradBasis(i, p);
        //      val.x += temp.x * vec[elem->GetNodes()[i]] * (diff);
        //      val.y += temp.y * vec[elem->GetNodes()[i]] * (diff);
        //      val.z += temp.z * vec[elem->GetNodes()[i]] * (diff);
        //  }
        //  return val;
        //}
        template<class Problem, class Grid, class Matrix>
        const std::vector<double> FEMethod<Problem, Grid, Matrix>::GetRightVector() const
        {
            return *m_rhsvector;
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethod<Problem, Grid, Matrix>::OutDatFormat(const Mesh::Point& mn, const Mesh::Point& mx, const std::string& file_name, const std::vector<double>& vec) const
        {
            std::ofstream of(file_name + "z.dat");
            std::streambuf *buf = std::cout.rdbuf();
            std::cout.rdbuf(of.rdbuf());
            std::cout « "TITLE = FE-METHOD\n";
            std::cout « "VARIABLES = \"dx1\", \"dx2\", \"u\"\n";
            std::cout « "ZONE i=51, j=51, F=POINT\n";
            double stepx = (mx.x - mn.x) / 51;
            double stepy = (mx.y - mn.y) / 51;
            for (int i = 0; i < 51; ++i)
                for (int j = 0; j < 51; ++j)
                    std::cout « mn.x + j * stepx « "\t" « mn.y + stepy * i « "\t" « GetValue(Mesh::Point(mn.x + j * stepx, mn.y + i * stepy, mn.z), vec) « std::endl;
            std::cout.rdbuf(buf);
            of.close();
            of.open(file_name + "x.dat");
            buf = std::cout.rdbuf();
            std::cout.rdbuf(of.rdbuf());
            std::cout « "TITLE = FE-METHOD\n";
            std::cout « "VARIABLES = \"dx1\", \"dx2\", \"u\"\n";
            std::cout « "ZONE i=51, j=51, F=POINT\n";
            for (int i = 0; i < 51; ++i)
                for (int j = 0; j < 51; ++j)
                    std::cout « mn.x + j * stepx « "\t" « mn.y + stepy * i « "\t" « GetValue(Mesh::Point(mn.z, mn.x + j * stepx, mn.y + i * stepy), vec) « std::endl;
            std::cout.rdbuf(buf);
            of.close();
            of.open(file_name + "y.dat");
            buf = std::cout.rdbuf();
            std::cout.rdbuf(of.rdbuf());
            std::cout « "TITLE = FE-METHOD\n";
            std::cout « "VARIABLES = \"dx1\", \"dx2\", \"u\"\n";
            std::cout « "ZONE i=51, j=51, F=POINT\n";
            for (int i = 0; i < 51; ++i)
                for (int j = 0; j < 51; ++j)
                    std::cout « mn.x + j * stepx « "\t" « mn.y + stepy * i « "\t" « GetValue(Mesh::Point(mn.x + j * stepx, mn.z, mn.y + i * stepy), vec) « std::endl;
```

```
            std::cout.rdbuf(buf);
            of.close();
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethod<Problem, Grid, Matrix>::ApplySources()
        {
            int nfem = -1;
            auto total = m_problem->get_total_sources();
            for (int i = 0; i < total; ++i)
            {
                auto src = m_problem->get_point_source(i);
                auto point = src.get_point();
                nfem = m_Grid->FindElement(point);
                if (nfem != -1)
                {
                    auto val = src.get_value();
                    auto elem = m_Grid->GetElement(nfem);
                    for (int j = 0; j < 3; ++j)
                        m_rhsvector->operator[](elem->GetNode(j)) += val * elem->GetShapeFunction(j,
    point);
                }
                nfem = -1;
            }
            /*for (auto srd : m_Grid->GetDottedSources())
            {
                nfem = m_Grid->FindElement(srd.first);
                if (nfem != -1)
                {
                    auto elem = m_Grid->GetElements()[nfem];
                    for (int i = 0; i < elem->GetDoF(); ++i)
                    {
                        m_rhsvector[elem->GetNodes()[i]] += srd.second * elem->GetBasis(i, srd.first);
                    }
                }
                nfem = -1;
            }*/
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethod<Problem, Grid, Matrix>::Rediscretization(const std::shared_ptr<Grid>& grid)
        {
            m_GlobalMatrix->NullMatrix();
            for (unsigned int i = 0; i < m_rhsvector->size(); ++i)
                (*m_rhsvector)[i] = 0;
            AssemblGlobal();
            //SecondConditions();
            //ApplySources();
            //StefanConditions();
            MainConditions();
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethod<Problem, Grid, Matrix>::Rediscretization()
        {
            m_time += m_step;
            m_GlobalMatrix->NullMatrix();
            for (unsigned int i = 0; i < m_rhsvector->size(); ++i)
                (*m_rhsvector)[i] = 0;
            AssemblGlobal();
            SecondConditions();
            ThirdConditions();
            StefanConditions();
            //ApplySources();
            MainConditions();
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethod<Problem, Grid, Matrix>::GetSolution(std::vector<double>& vec)
        {
            int size = vec.size();
            //Translation(vec);
            for (int i = 0; i < size; ++i)
                vec[i] = m_solution[i];
        }
        template<class Problem, class Grid, class Matrix>
        const double FEMethod<Problem, Grid, Matrix>::GetSolution(const Grid& g, const
    std::vector<double> &weights, const Mesh::Point& p)
        {
            double sum{ 0 };
            auto nfem{ g.FindElement(p) };
            if (nfem < 0)
                return 0.;
            auto elem{ g.GetElement(nfem) };
            auto dofs{ elem->GetDoFs() };
            for (auto i{ 0 }; i < dofs; ++i)
                sum += weights[elem->GetNode(i)] * elem->GetShapeFunction(i, p);
            return sum;
        }
        template<class Problem, class Grid, class Matrix>
        const double FEMethod<Problem, Grid, Matrix>::GetSolution(const Grid& g, const
```

```
std::vector<double> &weights, const Mesh::Point& p, const int nfem)
    {
        double sum{ 0 };
        //if (nfem < 0)
        //    return 0.;
        auto elem{ g.GetElement(nfem) };
        auto dofs{ elem->GetDoFs() };
        //std::cout « nfem « std::endl;
        for (auto i{ 0 }; i < dofs; ++i)
            sum += weights[elem->GetNode(i)] * elem->GetShapeFunction(i, p);
        return sum;
    }
    template<class Problem, class Grid, class Matrix>
    const Mesh::Point FEMethod<Problem, Grid, Matrix>::GetGradSolution(const Grid& g, const
std::vector<double> &weights, const Mesh::Point& p)
    {
        Mesh::Point sum{ 0, 0, 0 };
        auto nfem{ g.FindElement(p) };
        auto elem{ g.GetElement(nfem) };
        auto dofs{ elem->GetDoFs() };
        for (auto i{ 0 }; i < dofs; ++i)
            sum += weights[elem->GetNode(i)] * elem->GetGradShapeFunction(i, p);
        return sum;
    }
    template<class Problem, class Grid, class Matrix>
    const Mesh::Point FEMethod<Problem, Grid, Matrix>::GetGradSolution(const Grid& g, const
std::vector<double> &weights, const Mesh::Point& p, const int nfem)
    {
        Mesh::Point sum{ 0, 0, 0 };
        auto elem{ g.GetElement(nfem) };
        auto dofs{ elem->GetDoFs() };
        for (auto i{ 0 }; i < dofs; ++i)
            sum += weights[elem->GetNode(i)] * elem->GetGradShapeFunction(i, p);
        return sum;
    }
    template<class Problem, class Grid, class Matrix>
    void FEMethod<Problem, Grid, Matrix>::LoadSolution(const std::vector<double>& vec)
    {
        m_solution.resize(vec.size());
        for (unsigned int i = 0; i < vec.size(); ++i)
            m_solution[i] = vec[i];
    }
    template<class Problem, class Grid, class Matrix>
    void FEMethod<Problem, Grid, Matrix>::OutMeshFormat(const std::string& file_name, const
std::vector<double>& vec)
    {
        const int size{ (int)m_Grid->GetNodes().size() };
        const int number{ (int)m_Grid->GetElements().size() };
        //const int size{ number * 4 };
        std::ofstream ofs(file_name + ".dat", std::ios::out);
        std::string title("TITLE = \"Mesh data\"\n Variables = \"X\", \"Y\", \"Z\", \"U\"\n Zone N =
" + std::to_string(size) + ", E = " + std::to_string(number) + ", DATAPACKING = POINT, ZONETYPE =
FETETRAHEDRON\n");
        ofs « title;
        Mesh::Point p;
        for (int i = 0; i < size; ++i)
        {
            p = m_Grid->GetNodes()[i];
            ofs « p.x « "\t" « p.y « "\t" « p.z « "\t" « GetValue(p, vec, 1) « std::endl;
        }
        for (int i = 0; i < number; ++i)
        {
            auto elem = m_Grid->GetElements()[i];
            for (int k = 0; k < 4; ++k)
            {
                ofs « elem->GetNodes()[k] + 1 « "\t";
            }
            ofs « std::endl;
        }
        ofs.close();
    }
    template<class Problem, class Grid, class Matrix>
    void FEMethod<Problem, Grid, Matrix>::OutMeshTimeFormat(const std::string& file_name, const
std::vector<double>& vec)
    {
        const int size{ (int)m_Grid->GetNodes().size() };
        const int number{ (int)m_Grid->GetElements().size() };
        //const int size{ number * 4 };
        std::ofstream ofs(file_name + ".dat", std::ios::out | std::ios::app);
        std::string title("TITLE = \"Mesh data\"\n Variables = \"X\", \"Y\", \"Z\", \"U\"\n Zone N =
" + std::to_string(size) + ", E = " + std::to_string(number) + ", DATAPACKING = POINT, ZONETYPE =
FETETRAHEDRON\n");
        ofs « title;
        Mesh::Point p;
        for (int i = 0; i < size; ++i)
        {
            p = m_Grid->GetNodes()[i];
```

```
                ofs « p.x « "\t" « p.y « "\t" « p.z « "\t" « GetValue(p, vec, 1) « std::endl;
            }
            for (int i = 0; i < number; ++i)
            {
                auto elem = m_Grid->GetElements()[i];
                for (int k = 0; k < 4; ++k)
                {
                    ofs « elem->GetNodes()[k] + 1 « "\t";
                }
                ofs « std::endl;
            }
            ofs.close();
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethod<Problem, Grid, Matrix>::ProjectSolution(std::vector<double>& sol, std::function<const double(const Mesh::Point&, const std::vector<double>&, const int)> GetVal, std::vector<double>& vec)
        {
            for (int i = 0; i < m_Grid->GetElements().size(); ++i)
            {
                auto elem = m_Grid->GetElements()[i];
                int order = elem->GetDoF();
                for (int j = 0; j < order; ++j)
                    sol[elem->GetNodes(j)] = GetVal(m_Grid->GetNodes()[elem->GetNodes(j)], vec, i);
            }
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethod<Problem, Grid, Matrix>::ProjectSolution(std::vector<double>& sol, std::function<const double(const Mesh::Point&, const std::vector<double>&)> GetVal, std::vector<double>& vec, const int)
        {
            for (int i = 0; i < m_Grid->GetElements().size(); ++i)
            {
                auto elem = m_Grid->GetElements()[i];
                int order = elem->GetDoF();
                for (int j = 0; j < order; ++j)
                    sol[elem->GetNodes(j)] = GetVal(m_Grid->GetNodes()[elem->GetNodes(j)], vec);
            }
        }
        template<class Problem, class Grid, class Matrix>
        const std::vector<double> FEMethod<Problem, Grid, Matrix>::SetSolution(const int sol, const int liq, const double s, const double l, const double m)
        {
            int i;
            m_solution.resize(m_Grid->GetNodes().size());
            for (i = 0; i < m_Grid->GetElements().size(); ++i)
            {
                auto elem = m_Grid->GetElements()[i];
                int order = elem->GetDoF();
                if (m_Grid->GetElements()[i]->GetType() == liq)
                    for (int j = 0; j < order; ++j)
                        m_solution[elem->GetNodes()[j]] = l;
                else
                    for (int j = 0; j < order; ++j)
                        m_solution[elem->GetNodes()[j]] = s;
            }

            for (auto bnd : m_Grid->GetBoundaryConditions())
            {
                //if (get<0>(bnd.second) == 4)
                {
                    for (auto row : m_Grid->GetBoundary())
                    {
                        if (bnd.first == row->GetType())
                        {
                            int dofs = row->GetDoF();
                            for (int i = 0; i < dofs; ++i)
                            {
                                m_solution[row->GetNodes(i)] = m;
                            }
                        }
                    }
                }
            }
            return m_solution;
        }
        template<class Problem, class Grid, class Matrix>
        FEMethod<Problem, Grid, Matrix>::~FEMethod()
        {
            delete m_Grid;
        }
    }
}

#endif // !CORENC_METHODS_FEMethod_h
```

7.81 CoreNCFEM/Methods/FEMethodZero.h File Reference

```
#include <functional>
#include <set>
#include "../Point.h"
#include "../Parameter.h"
#include "CSMethod.h"
#include <memory>
#include <cmath>
#include <map>
#include <algorithm>
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
```

Classes

- class corenc::method::CFEMethodZero< Type >
- class corenc::method::FEMethodZero< Problem, Grid, Matrix >

Namespaces

- namespace corenc
- namespace corenc::Mesh
- namespace corenc::method

7.82 FEMethodZero.h

Go to the documentation of this file.

```
// FEMethodZero.h describes an abstract interface and functions for a general finite element method with zero Dirichlet boundaries
#ifndef CORENC_METHODS_FEMethodZeroZero_h
#define CORENC_METHODS_FEMethodZeroZero_h
#include <functional>
#include <set>
#include "../Point.h"
#include "../Parameter.h"
#include "CSMethod.h"
#include <memory>
#include <cmath>
#include <map>
#include <algorithm>
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
namespace corenc
{
    namespace Mesh
    {
        class Point;
    }
    namespace method
    {
        // class Type = Type of the solution, for ex vector or double, or even more specific


        template<class Type>
        class CFEMethodZero
        {
        public:
```

```
        CFEMethodZero() {};
        virtual ~CFEMethodZero() {};
        virtual const int                     Assemble() = 0;
        virtual const Type                    GetSolution(const std::vector<double>& point) const = 0;
        virtual const std::vector<Type>       GetSolution() const = 0;
        virtual const Type                    GetMaxSolution() const = 0;
        virtual const Type                    GetMinSolution() const = 0;
    };

    template<class Problem, class Grid, class Matrix>
    class FEMethodZero
    {
    public:
        FEMethodZero() :
            m_problem{nullptr},
            m_Grid{nullptr},
            m_GlobalMatrix{nullptr},
            m_RightMatrix{nullptr},
            m_rhsvector{nullptr}
        {}
        FEMethodZero(
            Problem* p,
            Grid* g,
            Matrix* m,
            std::vector<double>* rhs):
            m_problem{ p },
            m_Grid{ g->Clone() },
            m_GlobalMatrix{ m },
            m_N{ g->GetNumberOfElements() },
            m_Ns{ g->GetNumberOfBoundaries() },
            m_rhsvector{ rhs }{
            //GeneratePortrait();
        }
        FEMethodZero(
            Problem* p,
            Grid* g,
            Matrix* m,
            Matrix* rm,
            std::vector<double>* rhs):
            m_problem{ p },
            m_Grid{ g->Clone() },
            m_GlobalMatrix{ m },
            m_RightMatrix{ rm },
            m_N{ g->GetNumberOfElements() },
            m_Ns{ g->GetNumberOfBoundaries() },
            m_rhsvector{ rhs }{
            //GeneratePortrait();
        }
        FEMethodZero(const std::shared_ptr<Grid>& grid) :m_Grid{ grid->Clone() } {}
        FEMethodZero(Grid* grid) :m_Grid{ grid->Clone() } {}
        FEMethodZero(const FEMethodZero& meth) :
            m_Grid{ meth.m_Grid->Clone() },
            //m_GlobalMatrix{ meth.m_GlobalMatrix->Clone() },
            //m_rhsvector{ meth.m_rhsvector },
            //m_problem{ meth.m_problem },
            m_time{ meth.m_time },
            //m_solution{ meth.m_solution },
            m_size{ meth.m_size },
            m_N{ meth.m_N },
            m_Ns{ meth.m_Ns },
            m_nums{ meth.m_nums }
        {};
        void                        Discretization();
        const double                GetValue(const Mesh::Point&) const;
        const double                GetValue(const Mesh::Point&, const std::vector<double>& vec) const;
        const double                GetValue(const Mesh::Point&, const std::vector<double>& vec, const int num) const;
        //const Mesh::Point         GetGradValue(const Mesh::Point&, const std::vector<double>& vec) const;
        //const Mesh::Point         GetLambdaGrad(const Mesh::Point&, const std::vector<double>& vec) const;
         const double                GetEffective(const std::vector<double>& vec) const;
         void                        ProjectSolution(std::vector<double>&, std::function<const double(const Mesh::Point&, const std::vector<double>&, const int)> GetValue, std::vector<double>& sol);
         void                        ProjectSolution(std::vector<double>&, std::function<const double(const Mesh::Point&, const std::vector<double>&)> GetValue, std::vector<double>& sol, const int);
         void                        LoadSolution(const std::vector<double>& vec);
         const std::vector<double>   SetSolution(const int sol, const int liq, const double, const double, const double);
         void                        GetSolution(std::vector<double>& vec);
         void                        Rediscretization(const std::shared_ptr<Grid>&);
         void                        Rediscretization();
         void                        SetTimeStep(const double& step) { m_step = step; m_time = step;
```

```
    }
        Matrix*                     GetGlobalMatrix() const;
        Grid*                       GetMesh() { return m_Grid; }
        const std::vector<double>   GetRightVector() const;
        void                        OutDatFormat(const Mesh::Point& min, const Mesh::Point& max, const std::string& file_name, const std::vector<double>& vec) const;
        void                        OutMeshFormat(const std::string& file_name, const std::vector<double>& vec);
        void                        OutMeshTimeFormat(const std::string& file_name, const std::vector<double>& vec);
        static const double         GetSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p);
        static const double         GetSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p, const int nfem);
        static const Mesh::Point GetGradSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p);
        static const Mesh::Point GetGradSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p, const int n);
        ~FEMethodZero();
    private:
        void                        GeneratePortrait();
        void                        AssemblGlobal();
        void                        MainConditions();
        void                        SecondConditions();
        void                        ThirdConditions();
        void                        StefanConditions();
        void                        ApplySources();
        const int                   AssembleLocalMatrix(const int);
        const int                   AssembleIDUDVMatrix(const int);
        const int                   AssembleIDUVMatrix(const int);
        const int                   AssembleIUDVMatrix(const int);
        const int                   AssembleRUVMatrix(const int);
        const int                   AssembleSUPGMatrix(const int);
        const int                   AssembleLocalMatrix(const int, const int);
        Grid*                       m_Grid = nullptr;
        Matrix*                     m_GlobalMatrix = nullptr;
        Matrix*                 m_RightMatrix = nullptr;
        Problem*                    m_problem = nullptr;
        std::vector<double>         m_solution;
        std::vector<double>*        m_rhsvector;
        unsigned int                m_size;
        double                      m_step{ 0.1 };
        double                      m_time{ 0.1 };
        unsigned int                m_N;
        unsigned int                m_Ns;
        std::vector<unsigned int>   m_nums;

    };

    template<class Problem, class Grid, class Matrix>
    void FEMethodZero<Problem, Grid, Matrix>::Discretization()
    {
        GeneratePortrait();
        AssemblGlobal();
        //ApplySources();
        //SecondConditions();
        //ThirdConditions();
        //MainConditions();
        //StefanConditions();
    }
    template<class Problem, class Grid, class Matrix>
    void FEMethodZero<Problem, Grid, Matrix>::GeneratePortrait()
    {
        const auto& el = m_Grid->GetElement(0);
        int order = m_Grid->GetElement(0)->GetDoFs();
        std::vector<std::set<unsigned int» temp;
        m_Ns = m_Grid->GetNumberOfINodes();
        m_N = m_Grid->GetNumberOfElements();
        temp.resize(m_Grid->GetNumberOfINodes());
        unsigned i, j, k;
        for (k = 0; k < m_N; ++k)
        {
            const auto& elem{ m_Grid->GetElement(k) };
            for (i = 0; i < order; ++i)
                for (j = 0; j < order; ++j)
                {
                    //std::cout « "inside" « std::endl;
                    int jnode = m_Grid->interpolate(elem->GetNode(j));
                    int inode = m_Grid->interpolate(elem->GetNode(i));
                    //std::cout « jnode « "\t" « inode « std::endl;
                    //std::cout « "outside" « std::endl;
                    if (jnode > -1 && inode > -1)
                        if (jnode > inode)
                        {
                            temp[jnode].insert(inode);
                        }
                }
```

```
        }
        if(m_problem->findTerm(Terms::RUV))
            m_RightMatrix->Create(temp.size(), temp);

        //m_GlobalMatrix = std::shared_ptr<Matrix>(new Matrix(m_Grid->GetNumberOfNodes(), temp));
        //m_rhsvector.resize(m_Grid->GetNumberOfNodes());
        //std::cout « temp.size() « std::endl;
        m_GlobalMatrix->Create(temp.size(), temp);
        m_rhsvector->resize(temp.size());
        //m_solution.resize(m_Grid->GetNumberOfNodes());
        //for (int l = 0; l < m_Grid->GetNumberOfNodes(); ++l)
        //  m_solution[l] = 20;
    }
    template<class Problem, class Grid, class Matrix>
    void FEMethodZero<Problem, Grid, Matrix>::AssemblGlobal()
    {
        int l;
        //std::vector<std::future<int» futures;
        int i, j, k, nodes;
        double mij;
        const int terms{ (int)m_problem->getNumberOfTerms() };
        for (k = 0; k < terms; ++k)
        {
            switch (m_problem->getTerm(k))
            {
                case Terms::IDUDV:
                    for (l = 0; l < m_N; ++l)
                        AssembleIDUDVMatrix(l);
                    break;
                case Terms::IDUV:
                    for (l = 0; l < m_N; ++l)
                        AssembleIDUVMatrix(l);
                    break;
                case Terms::IUDV:
                    for (l = 0; l < m_N; ++l)
                        AssembleIUDVMatrix(l);
                    break;
                case Terms::SUPG:
                    for (l = 0; l < m_N; ++l)
                        AssembleSUPGMatrix(l);
                    break;
                case Terms::RUV:
                    for (l = 0; l < m_N; ++l)
                        AssembleRUVMatrix(l);
                    break;
                default:
                    break;
            }
        }
        //for (l = 0; l < m_N; ++l)
            //futures.push_back(async(&DGMethod<Problem, Grid, Matrix>::AssembleLocalMatrix, this, l));
         //   AssembleLocalMatrix(l, 0);
        //for (auto &it : futures)
        //it.get();
    }

    template<class Problem, class Grid, class Matrix>
    const int FEMethodZero<Problem, Grid, Matrix>::AssembleIDUDVMatrix(const int l)
    {
        int i, j, k, nodes;
        double mij;
        const auto& elem{ m_Grid->GetElement(l) };
        const int dofs{ (int)elem->GetDoFs() };
        const int terms{ (int)m_problem->getNumberOfTerms() };
        nodes = elem->GetNumberOfNodes();
        std::vector<Mesh::Point> points(nodes);
        for (i = 0; i < nodes; ++i)
            points[i] = m_Grid->GetNode(elem->GetNode(i));
        for (i = 0; i < (int)dofs; ++i)
        {
            for (j = 0; j < (int)dofs; ++j)
            {
                auto inode = m_Grid->interpolate(elem->GetNode(i));
                auto jnode = m_Grid->interpolate(elem->GetNode(j));
                if (inode == -1 || jnode == -1)
                    continue;
                auto M = [&](const Mesh::Point& p)
                {
                    //auto m = elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                    return m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p) * elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                };
                //mij = m_Grid->getParameter(Parameters::DIFFUSION, l, j) * elem->Integrate(M, points);
                mij = elem->Integrate(M, points);
                m_GlobalMatrix->AddElement(inode, jnode, mij);
```

```
            }
        }
        return 0;
    }

    template<class Problem, class Grid, class Matrix>
    const int FEMethodZero<Problem, Grid, Matrix>::AssembleIDUVMatrix(const int l)
    {
        int i, j, k, nodes;
        double mij;
        const auto& elem{ m_Grid->GetElement(l) };
        const int dofs{ (int)elem->GetDoFs() };
        const int terms{ (int)m_problem->getNumberOfTerms() };
        nodes = elem->GetNumberOfNodes();
        std::vector<Mesh::Point> points(nodes);
        for (i = 0; i < nodes; ++i)
            points[i] = m_Grid->GetNode(elem->GetNode(i));
        for (i = 0; i < (int)dofs; ++i)
        {
            for (j = 0; j < (int)dofs; ++j)
            {
                auto inode = m_Grid->interpolate(elem->GetNode(i));
                auto jnode = m_Grid->interpolate(elem->GetNode(j));
                if (inode == -1 || jnode == -1)
                    continue;
                auto M = [&](const Mesh::Point& p)
                {
                    return m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) *
elem->GetShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                };
                auto _mij = elem->Integrate(M, points);
                m_GlobalMatrix->AddElement(inode, jnode, _mij);
            }
        }
        return 0;
    }

    template<class Problem, class Grid, class Matrix>
    const int FEMethodZero<Problem, Grid, Matrix>::AssembleIUDVMatrix(const int l)
    {
        int i, j, k, nodes;
        double mij;
        const auto& elem{ m_Grid->GetElement(l) };
        const int dofs{ (int)elem->GetDoFs() };
        const int terms{ (int)m_problem->getNumberOfTerms() };
        nodes = elem->GetNumberOfNodes();
        std::vector<Mesh::Point> points(nodes);
        for (i = 0; i < nodes; ++i)
            points[i] = m_Grid->GetNode(elem->GetNode(i));
        for (i = 0; i < dofs; ++i)
        {
            for (j = 0; j < dofs; ++j)
            {
                auto inode = m_Grid->interpolate(elem->GetNode(i));
                auto jnode = m_Grid->interpolate(elem->GetNode(j));
                if (inode == -1 || jnode == -1)
                    continue;
                auto M = [&](const Mesh::Point& p)
                {
                    return elem->GetGradShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                };
                //mij = m_CoarseGrid->getParameter(Parameters::ADVECTION, l, j) *
m_flux(m_CoarseGrid->getSolution(l, j)) * elem->Integrate(M, points).x;
                mij = elem->Integrate(M, points).x;
                m_GlobalMatrix->AddElement(inode, jnode, mij);
            }
        }
        return 0;
    }


    template<class Problem, class Grid, class Matrix>
    const int FEMethodZero<Problem, Grid, Matrix>::AssembleRUVMatrix(const int l)
    {
        int i, j, k, nodes;
        double mij;
        const auto& elem{ m_Grid->GetElement(l) };
        const int dofs{ (int)elem->GetDoFs() };
        const int terms{ (int)m_problem->getNumberOfTerms() };
        nodes = elem->GetNumberOfNodes();
        std::vector<Mesh::Point> points(nodes);
        for (i = 0; i < nodes; ++i)
            points[i] = m_Grid->GetNode(elem->GetNode(i));
        double minPec = -1;
        auto MM = [&](const Mesh::Point& p)
        {
            double vel = sqrt(m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) *
```

```
m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0));
        double h = elem->GetMeasure();
        //h = fabs(points[1].x - points[0].x);
        h = sqrt(h);
        //h = fabs(m_Grid->GetNode(0).x - m_Grid->GetNode(1).x);
        double Pe = vel * h / 2. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);

        //double Pe = vel * h / 2. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);

        if (Pe > minPec)
            minPec = Pe;
        return 0.;
    };
    elem->Integrate(MM, points);
    for (i = 0; i < (int)dofs; ++i)
    {
        for (j = 0; j < (int)dofs; ++j)
        {
            auto inode = m_Grid->interpolate(elem->GetNode(i));
            auto jnode = m_Grid->interpolate(elem->GetNode(j));
            if (inode == -1 || jnode == -1)
                continue;
            auto M = [&](const Mesh::Point& p)
            {
                double vel = sqrt(m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0));
                double h = elem->GetMeasure();
                //h = fabs(points[1].x - points[0].x);
                h = sqrt(h);
                //h = fabs(m_Grid->GetNode(0).x - m_Grid->GetNode(1).x);
                //double Pe = vel * h / 6. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                double Pe = minPec;
                double tau = 0.;
                double beta = h / 2. / vel * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                //double Pe = vel * h / 2. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);

                if (Pe >= 1)
                    tau = h / 2. / vel;
                else
                    tau = h * h / 12. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                    //tau = 0.;
                //tau = 1e-7;
                //std::cout << "tau Pe:\t" << Pe << std::endl;
                //std::cout << "tau vel:\t" << vel << std::endl;
                //std::cout << "tau h:\t" << h << std::endl;
                //std::cout << "tau:\t" << tau << std::endl;
                auto supg = tau * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * elem->GetGradShapeFunction(i, p) * elem->GetShapeFunction(j, p) * elem->GetShapeFunction(i, p);
                return elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);// + supg;
            };
            mij = elem->Integrate(M, points);
            m_RightMatrix->AddElement(inode, jnode, mij);
        }
    }
    return 0;
}

template<class Problem, class Grid, class Matrix>
const int FEMethodZero<Problem, Grid, Matrix>::AssembleSUPGMatrix(const int l)
{
    int i, j, k, nodes;
    double mij;
    const auto& elem{ m_Grid->GetElement(l) };
    const int dofs{ (int)elem->GetDoFs() };
    const int terms{ (int)m_problem->getNumberOfTerms() };
    nodes = elem->GetNumberOfNodes();
    std::vector<Mesh::Point> points(nodes);
    for (i = 0; i < nodes; ++i)
        points[i] = m_Grid->GetNode(elem->GetNode(i));
    double minPec = -1;
    auto MM = [&](const Mesh::Point& p)
    {
        double vel = sqrt(m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0));
        double h = elem->GetMeasure();
        //h = fabs(points[1].x - points[0].x);
        h = sqrt(h);
        //h = fabs(m_Grid->GetNode(0).x - m_Grid->GetNode(1).x);
        double Pe = vel * h / 2. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);

```

```
                //double Pe = vel * h / 2. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);

                if (Pe > minPec)
                    minPec = Pe;
                return 0.;
            };
            elem->Integrate(MM, points);
            for (i = 0; i < (int)dofs; ++i)
            {
                for (j = 0; j < (int)dofs; ++j)
                {
                    auto inode = m_Grid->interpolate(elem->GetNode(i));
                    auto jnode = m_Grid->interpolate(elem->GetNode(j));
                    if (inode == -1 || jnode == -1)
                        continue;
                    auto M = [&](const Mesh::Point& p)
                    {
                        double vel = sqrt(m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0));
                        double h = elem->GetMeasure();
                        h = sqrt(h);
                        //h = fabs(points[1].x - points[0].x);
                        //h = fabs(m_Grid->GetNode(0).x - m_Grid->GetNode(1).x);
                        //h *= h;
                        //double Pe = vel * h / 6. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                        double tau = 0.;
                        //double Pe = vel * h / 6. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                        double Pe = minPec;
                        double beta = h / 2. / vel * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                        //double beta = h / std::sqrt(3.) * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                        //double beta = h / 2. * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                        //double beta = h / 2. * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                        //beta = 0.;
                        //for (int ii = 0; ii < (int)dofs; ++ii)
                            //beta += m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * elem->GetGradShapeFunction(ii, p);
                        //return beta * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) *
                        //        elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                        if (Pe >= 1)
                            tau = h / 2. / vel;
                        else
                            tau = h * h / 12. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                            //tau = 0;
                        //return 0.;
                        //tau = 1e-7;
                        //std::cout « "Stau Pe:\t" « Pe « std::endl;
                        //std::cout « "Stau vel:\t" « vel « std::endl;
                        //std::cout « "Stau h:\t" « h « std::endl;
                        //std::cout « "Stau:\t" « tau « std::endl;
                        auto ret = tau * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) *
                                elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                        //std::cout « ret « std::endl;
                        return ret;
                    };

                    //double tau =
                    auto _mij = elem->Integrate(M, points);
                    m_GlobalMatrix->AddElement(inode, jnode, _mij);
                }
            }
            return 0;
        }

        template<class Problem, class Grid, class Matrix>
        const int FEMethodZero<Problem, Grid, Matrix>::AssembleLocalMatrix(const int l, const int old)
        {
            int i, j, k, nodes;
            double mij;
            const auto& elem{ m_Grid->GetElement(l) };
            const int dofs{ (int)elem->GetDoFs() };
            const int terms{ (int)m_problem->getNumberOfTerms() };
            nodes = elem->GetNumberOfNodes();
            std::vector<Mesh::Point> points(nodes);
            for (i = 0; i < nodes; ++i)
                points[i] = m_Grid->GetNode(elem->GetNode(i));
            for (k = 0; k < terms; ++k)
            {
                switch (m_problem->getTerm(k))
                {
```

```
            case Terms::IUV:
                for (i = 0; i < (int)dofs; ++i)
                {
                    for (j = 0; j < (int)dofs; ++j)
                    {
                        auto M = [&](const Mesh::Point& p)
                        {
                            return m_problem->get_parameter(Terms::IUV, l, elem->GetType(), p) * elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                        };
                        mij = elem->Integrate(M, points);
                        auto inode = m_Grid->interpolate(elem->GetNode(i));
                        auto jnode = m_Grid->interpolate(elem->GetNode(j));
                        if (inode > -1 && jnode > -1)
                            m_GlobalMatrix->AddElement(inode, jnode, mij);
                    }
                }
                break;
            case Terms::IDUDV:
                for (i = 0; i < (int)dofs; ++i)
                {
                    for (j = 0; j < (int)dofs; ++j)
                    {
                        auto inode = m_Grid->interpolate(elem->GetNode(i));
                        auto jnode = m_Grid->interpolate(elem->GetNode(j));
                        if (inode == -1 || jnode == -1)
                            continue;
                        auto M = [&](const Mesh::Point& p)
                        {
                            //auto m = elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                            return m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p) * elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                        };
                        //mij = m_Grid->getParameter(Parameters::DIFFUSION, l, j) * elem->Integrate(M, points);
                        mij = elem->Integrate(M, points);
                        m_GlobalMatrix->AddElement(inode, jnode, mij);
                    }
                }
                break;
            case Terms::IDUV:
                for (i = 0; i < (int)dofs; ++i)
                {
                    for (j = 0; j < (int)dofs; ++j)
                    {
                        auto inode = m_Grid->interpolate(elem->GetNode(i));
                        auto jnode = m_Grid->interpolate(elem->GetNode(j));
                        if (inode == -1 || jnode == -1)
                            continue;
                        auto M = [&](const Mesh::Point& p)
                        {
                            return m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * elem->GetShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                        };
                        auto _mij = elem->Integrate(M, points);
                        m_GlobalMatrix->AddElement(inode, jnode, _mij);
                    }
                }
                break;
            case Terms::IUDV:
                for (i = 0; i < dofs; ++i)
                {
                    for (j = 0; j < dofs; ++j)
                    {
                        auto inode = m_Grid->interpolate(elem->GetNode(i));
                        auto jnode = m_Grid->interpolate(elem->GetNode(j));
                        if (inode == -1 || jnode == -1)
                            continue;
                        auto M = [&](const Mesh::Point& p)
                        {
                            return elem->GetGradShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                        };
                        //mij = m_CoarseGrid->getParameter(Parameters::ADVECTION, l, j) * m_flux(m_CoarseGrid->getSolution(l, j)) * elem->Integrate(M, points).x;
                        mij = elem->Integrate(M, points).x;
                        m_GlobalMatrix->AddElement(inode, jnode, mij);
                    }
                }
                break;
            case Terms::EUV:
                for (i = 0; i < dofs; ++i)
                {
                    for (j = 0; j < dofs; ++j)
                    {
                        auto M = [&](const Mesh::Point& p)
                        {
```

```
                             return elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                         };
                         mij = elem->Integrate(M, points);
                         m_rhsvector->operator[](elem->GetNode(i)) += m_Grid->getParameter(Parameters::MASS, l, j) * m_Grid->getSolution(l, j) * mij;
                         //m_rhsvector->operator[](m_nums[l] + i) += m_CoarseGrid->getParameter(Parameters::MASS, l, points[j]) * elem->GetValue(j) * mij;
                     }
                 }
                 break;
             case Terms::EDUDV:
                 for (i = 0; i < dofs; ++i)
                 {
                     for (j = 0; j < dofs; ++j)
                     {
                         auto M = [&](const Mesh::Point& p)
                         {
                             return elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                         };
                         mij = elem->Integrate(M, points);
                         m_rhsvector->operator[](elem->GetNode(i)) += m_Grid->getParameter(Parameters::DIFFUSION, l, j) * m_Grid->getSolution(l, j) * mij;
                     }
                 }
                 break;
             case Terms::EDUV:
                 for (i = 0; i < dofs; ++i)
                 {
                     for (j = 0; j < dofs; ++j)
                     {
                         auto M = [&](const Mesh::Point& p)
                         {
                             return elem->GetShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                         };
                         mij = elem->Integrate(M, points).x;
                         m_rhsvector->operator[](elem->GetNode(i)) += m_Grid->getParameter(Parameters::ADVECTION, l, j) * mij;
                     }
                 }
                 break;
             case Terms::EUDV:
                 for (i = 0; i < dofs; ++i)
                 {
                     for (j = 0; j < dofs; ++j)
                     {
                         auto M = [&](const Mesh::Point& p)
                         {
                             return elem->GetGradShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                         };
                         mij = elem->Integrate(M, points).x;
                         m_rhsvector->operator[](elem->GetNode(i)) += m_Grid->getParameter(Parameters::ADVECTION, l, j) * mij;// *mij;
                     }
                 }
                 break;
             case Terms::EFV:
                 for (i = 0; i < dofs; ++i)
                 {
                     /*for (j = 0; j < dofs; ++j)
                     {
                         auto M = [&](const Mesh::Point& p)
                         {
                             return m_problem->get_parameter(Terms::EFV, elem->GetType(), l, j, p) * elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                         };
                         mij = elem->Integrate(M, points);
                         m_rhsvector->operator[](elem->GetNode(i)) += mij;
                     }*/
                     auto M = [&](const Mesh::Point& p)
                     {
                         return m_problem->get_parameter(Terms::EFV, elem->GetType(), l, i, p) * elem->GetShapeFunction(i, p);
                     };
                     mij = elem->Integrate(M, points);
                     m_rhsvector->operator[](elem->GetNode(i)) += mij;
                 }
                 break;
             case Terms::RUV:
                 for (i = 0; i < (int)dofs; ++i)
                 {
                     for (j = 0; j < (int)dofs; ++j)
                     {
                         auto M = [&](const Mesh::Point& p)
                         {
                             return elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);
                         };
```

```
                        mij = elem->Integrate(M, points);
                        auto inode = m_Grid->interpolate(elem->GetNode(i));
                        auto jnode = m_Grid->interpolate(elem->GetNode(j));
                        if (inode > -1 && jnode > -1)
                            m_RightMatrix->AddElement(inode, jnode, mij);
                    }
                }
                break;
            case Terms::SUPG:
                for (i = 0; i < (int)dofs; ++i)
                {
                    for (j = 0; j < (int)dofs; ++j)
                    {
                        auto inode = m_Grid->interpolate(elem->GetNode(i));
                        auto jnode = m_Grid->interpolate(elem->GetNode(j));
                        if (inode == -1 || jnode == -1)
                            continue;
                        auto M = [&](const Mesh::Point& p)
                        {
                            double vel = sqrt(m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0));
                            double h = elem->GetMeasure();
                            //double Pe = vel * h / 6. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                            double tau = 0.;
                            double Pe = vel * h / 2. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                            //double beta = h / 2. / vel * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                            double beta = h / std::sqrt(3.) * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                            //double beta = h / 2. * ((exp(2. * Pe) + 1.) / (exp(2. * Pe) - 1.) - 1. / Pe);
                            //beta = 0.;
                            //for (int ii = 0; ii < (int)dofs; ++ii)
                                //beta += m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * elem->GetGradShapeFunction(ii, p);
                            return beta * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) *
                                    elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                            if (Pe >= 1)
                                tau = h / 2. / vel;
                            else
                                tau = h * h / 12. / m_problem->get_parameter(Terms::IDUDV, l, elem->GetType(), p);
                            //return 0.;
                            return tau * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) * m_problem->get_parameter(Terms::IDUV, l, elem->GetType(), p, 0) *
                                    elem->GetGradShapeFunction(i, p) * elem->GetGradShapeFunction(j, p);
                        };

                        //double tau =
                        auto _mij = elem->Integrate(M, points);
                        m_GlobalMatrix->AddElement(inode, jnode, _mij);
                    }
                }
                break;
            default:
                break;
            }
        }
        return 0;
    }
    template<class Problem, class Grid, class Matrix>
    void FEMethodZero<Problem, Grid, Matrix>::MainConditions()
    {
        double mu{ 1e10 };
        const auto n = m_problem->get_number_of_boundaries();
        const auto m = m_Grid->GetNumberOfBoundaries();
        for (int i = 0; i < n; ++i)
        {
            const auto& type = m_problem->get_boundary_type(i);
            for (int j = 0; j < m; ++j)
            {
                const auto& row = m_Grid->GetBoundary(j);
                if (row->GetType() == type)
                {
                    const int dofs = (int)row->GetDoFs();
                    const int dofs2 = 2;
                    const auto& elem_num = row->GetNeighbour(0);
                    const auto& elem = m_Grid->GetElement(elem_num);
                    const int dofs_elem = elem->GetDoFs();
                    std::vector<Mesh::Point> points(dofs_elem);
                    for (int k = 0; k < dofs_elem; ++k)
                        points[k] = m_Grid->GetNode(elem->GetNode(k));
```

```
                for (int k = 0; k < dofs; ++k)
                {
                    int l = 0;
                    for (; l < dofs_elem; ++l)
                    {
                        if (elem->GetNode(l) == row->GetNode(k))
                            break;
                    }
                    m_GlobalMatrix->NullRow(row->GetNode(k));
                    //m_GlobalMatrix->operator()(row->GetNode(k), row->GetNode(k)) *= mu;
                    //m_rhsvector->operator[](row->GetNode(k)) = m_problem->get_boundary_parameter(0, type, m_Grid->GetNode(row->GetNode(k)));
                    //m_rhsvector->operator[](row->GetNode(k)) = m_problem->get_boundary_parameter(0, type, elem_num, l, m_Grid->GetNode(row->GetNode(k)));
                    m_rhsvector->operator[](row->GetNode(k)) = elem->GetWeight(l, points, [=](const Mesh::Point& p) { return m_problem->get_boundary_parameter(0, type, p); });
                    if(m_problem->findTerm(Terms::RUV))
                        {
                            m_RightMatrix->NullRow(row->GetNode(k));
                            //m_RightMatrix->operator()(row->GetNode(k), row->GetNode(k)) *= mu;
                        }
                }
                /*for (int k = dofs2; k < dofs; ++k)
                {
                    m_GlobalMatrix->NullRow(row->GetNode(k));
                    m_rhsvector->operator[](row->GetNode(k)) = 0;
                }*/
            }
        }
    }
    /*for (auto bnd : m_Grid->GetBoundaryConditions())
    {
        if (get<0>(bnd.second) == 1)
            for (auto row : m_Grid->GetBoundary())
            {
                if (bnd.first == row->GetType())
                {
                    for (int i = 0; i < row->GetDoF(); ++i)
                    {
                        m_GlobalMatrix->NullRow(row->GetNodes(i));
                        m_rhsvector[row->GetNodes(i)] = get<1>(bnd.second)(m_Grid->GetNodes()[row->GetNodes(i)]);
                    }
                }
            }
    }*/
}
template<class Problem, class Grid, class Matrix>
void FEMethodZero<Problem, Grid, Matrix>::SecondConditions()
{
    double theta = 0;
    int nfem;
    Mesh::Point temp[3];
    std::vector<int> local;
    for (auto bnd : m_Grid->GetBoundaryConditions())
    {
        //if (get<0>(bnd.second) == 2)
        {
            for (auto row : m_Grid->GetBoundary())
            {
                if (bnd.first == row->GetType())
                {
                    local.resize(0);
                    int dofs = row->GetDoF();
                    nfem = row->GetNumberOfElement(0);
                    auto elem = m_Grid->GetElements()[nfem];
                    //auto GetBasis = [&](int t, Point p){return elem->GetBasis(t, p); };
                    for (int j = 0; j < dofs; ++j)
                    {
                        temp[j] = m_Grid->GetNodes()[row->GetNodes(j)];
                        for (int i = 0; i < elem->GetDoF(); ++i)
                        {
                            if (row->GetNodes(j) == elem->GetNodes()[i])
                            {
                                local.push_back(i);
                                break;
                            }
                        }
                    }
                    for (int i = 0; i < dofs; ++i)
                    {
                        for (int j = 0; j < dofs; ++j)
                        {
                            //theta = get<1>(bnd.second)(m_Grid->GetNodes()[row->GetNodes(i)]);
                            theta = 0;
                            auto GetMass = [&](const Mesh::Point& p) {return elem->GetBasis(local[j], p) * elem->GetBasis(local[i], p); };
```

```
                                auto GetBBasis = [&](const Mesh::Point& p) {return row->GetBasis(j, p)*row->GetBasis(i, p); };
                                //if (i < 2 || j < 2)
                                m_rhsvector[row->GetNodes(i)] += theta * row->Integrate(GetMass, temp);

                                //if (i < 3 || j < 3)
                                //  m_rhsvector[row[i + 1]] += theta * row->Integrate(GetBBasis, temp);
                            }
                        }
                    }
                }
            }
        }
    }
    template<class Problem, class Grid, class Matrix>
    void FEMethodZero<Problem, Grid, Matrix>::StefanConditions()
    {
        double dest{ 0. }, lat{ 0 };
        int nfem;
        Mesh::Point temp[3];
        std::vector<int> local;
        for (auto bnd : m_Grid->GetBoundaryConditions())
        {
            //if (get<0>(bnd.second) == 4)
            {
                lat = 0;
                //lat = get<2>(bnd.second);
                for (auto row : m_Grid->GetBoundary())
                {
                    if (bnd.first == row->GetType())
                    {
                        local.resize(0);
                        int dofs = row->GetDoF();
                        nfem = row->GetNumberOfElement(0);
                        auto elem = m_Grid->GetElements()[nfem];
                        //auto GetBasis = [&](int t, Point p){return elem->GetBasis(t, p); };
                        for (int j = 0; j < dofs; ++j)
                        {
                            temp[j] = m_Grid->GetNodes()[row->GetNodes(j)];
                            for (int i = 0; i < elem->GetDoF(); ++i)
                            {
                                if (row->GetNodes(j) == elem->GetNodes()[i])
                                {
                                    local.push_back(i);
                                    break;
                                }
                            }
                        }
                        for (int i = 0; i < dofs; ++i)
                        {
                            for (int j = 0; j < dofs; ++j)
                            {
                                dest = 0;
                                //dest = get<1>(bnd.second)(m_Grid->GetNodes()[row->GetNodes(i)]);
                                auto GetBBasis = [&](const Mesh::Point& p) {return row->GetBasis(j, p)*row->GetBasis(i, p); };
                                //if (i < 2 || j < 2)
                                m_rhsvector[row->GetNodes(i)] += dest * lat * row->Integrate(GetBBasis, temp);

                                //if (i < 3 || j < 3)
                                //  m_rhsvector[row[i + 1]] += theta * row->Integrate(GetBBasis, temp);
                            }
                        }
                    }
                }
            }
        }
    }
    template<class Problem, class Grid, class Matrix>
    void FEMethodZero<Problem, Grid, Matrix>::ThirdConditions()
    {
        double param{ 0 }, beta{ 0 };
        int nfem;
        Mesh::Point temp[6];
        std::vector<int> local;
        auto fxy = [&](const Mesh::Point& p) {return (10 * p.y*m_time + m_time) / 10; };
        //auto fxy = [&](const Point& p){return 10 * p.y + 10 * m_time; };
        for (auto bnd : m_Grid->GetBoundaryConditions())
        {
            //if (get<0>(bnd.second) == 3)
            {

                for (auto row : m_Grid->GetBoundary())
```

```
                {
                    if (bnd.first == row->GetType())
                    {
                        local.resize(0);
                        int dofs = row->GetDoF();
                        nfem = row->GetNumberOfElement(0);
                        auto elem = m_Grid->GetElements()[nfem];
                        //auto GetBasis = [&](int t, Point p){return elem->GetBasis(t, p); };
                        auto order = elem->GetDoF();
                        for (int j = 0; j < dofs; ++j)
                        {
                            temp[j] = m_Grid->GetNodes()[row->GetNodes(j)];
                            for (int i = 0; i < order; ++i)
                            {
                                if (row->GetNodes(j) == elem->GetNodes()[i])
                                {
                                    local.push_back(i);
                                    break;
                                }
                            }
                        }
                        double val{ 0 };
                        for (int i = 0; i < dofs; ++i)
                        {
                            for (int j = 0; j < dofs; ++j)
                            {
                                param = 0;
                                beta = 0;
                                //beta = get<2>(bnd.second);
                                //param = get<1>(bnd.second)(m_Grid->GetNodes()[row->GetNodes(i)]);
                                //param = fxy(temp[j]);
                                auto GetBBasis = [&](const Mesh::Point& p) {return
elem->GetBasis(local[j], p)*elem->GetBasis(local[i], p); };
                                //val = row->GetElement(GetBBasis, temp);
                                val = row->Integrate(GetBBasis, temp);
                                m_GlobalMatrix->operator()(row->GetNodes(i), row->GetNodes(j)) +=
beta * val;
                                m_rhsvector[row->GetNodes(i)] += beta * param * val;
                            }
                        }
                    }
                }
            }
        }
    }
    template<class Problem, class Grid, class Matrix>
    Matrix* FEMethodZero<Problem, Grid, Matrix>::GetGlobalMatrix() const
    {
        return m_GlobalMatrix;
    }
    template<class Problem, class Grid, class Matrix>
    const double FEMethodZero<Problem, Grid, Matrix>::GetValue(const Mesh::Point& p) const
    {
        if (!m_solution.size())
            return -1;
        double val = 0;
        int nfem = -1;
        nfem = m_Grid->FindElement(p);
        if (nfem == -1)
            return -1;
        auto elem = m_Grid->GetElements()[nfem];
        for (int i = 0; i < elem->GetDoF(); ++i)
            val += m_solution[elem->GetNodes()[i]] * elem->GetBasis(i, p);
        return val;
    }
    template<class Problem, class Grid, class Matrix>
    const double FEMethodZero<Problem, Grid, Matrix>::GetValue(const Mesh::Point& p, const
std::vector<double>& vec) const
    {
        if (!vec.size())
            return -1;
        double val{ 0 };
        int nfem{ -1 };
        nfem = m_Grid->FindElement(p);
        if (nfem == -1)
            return -1;
        auto elem = m_Grid->GetElements()[nfem];
        for (int i = 0; i < elem->GetDoFs(); ++i)
            val += vec[elem->GetNode(i)] * elem->GetShapeFunction(i, p);
        return val;
    }
    template<class Problem, class Grid, class Matrix>
    const double FEMethodZero<Problem, Grid, Matrix>::GetValue(const Mesh::Point& p, const
std::vector<double>& vec, const int num) const
    {
        if (!vec.size() || num < 0)
            return -1;
```

```
            double val{ 0 };
            auto elem = m_Grid->GetElements()[num];
            for (int i = 0; i < elem->GetDoF(); ++i)
                val += vec[elem->GetNodes()[i]] * elem->GetBasis(i, p);
            return val;
        }
        //template<class Problem, class Grid, class Matrix>
        //const Mesh::Point FEMethodZero<Problem, Grid, Matrix>::GetGradValue(const Mesh::Point& p,
const std::vector<double>& vec) const
        //{
        //  Mesh::Point val{ 0, 0 };
        //  int nfem{ -1 };
        //  nfem = m_Grid->FindElement(p);
        //  if (nfem == -1)
        //      return val;
        //  auto elem = m_Grid->GetElements()[nfem];
        //  for (int i = 0; i < elem->GetDoF(); ++i)
        //  {
        //      val.x += vec[elem->GetNodes()[i]] * elem->GetGradBasis(i, p).x;
        //      val.y += vec[elem->GetNodes()[i]] * elem->GetGradBasis(i, p).y;
        //      val.z += vec[elem->GetNodes()[i]] * elem->GetGradBasis(i, p).z;
        //  }
        //  return val;
        //}
        template<class Problem, class Grid, class Matrix>
        const double FEMethodZero<Problem, Grid, Matrix>::GetEffective(const std::vector<double>& vec)
const
        {
            double sum = 0;
            //std::vector<int> dofs;
            //Mesh::Point points[10];
            //for (int i = 0; i < m_Grid->GetElements().size(); ++i)
            //{
                //auto mb = [&](const Mesh::Point& b) {return GetGradValue(b, vec)*GetGradValue(b,
vec); };
                //dofs.resize(0);
                //auto elem = m_Grid->GetElements()[i];
                //int order = elem->GetDoF();
                //double diff = std::get<0>(m_Grid->GetDiffusion().find(elem->GetType())->second);
                //for (int j = 0; j < order; ++j)
                //{
                    //dofs.push_back(elem->GetNodes()[j]);
                    //points[j] = m_Grid->GetNodes()[dofs[j]];
                //}
                //sum += diff * elem->Integrate(mb, points);
            //}
            //std::cout « "Effect (local): " « sum « std::endl;
            //std::cout « "Effect (local) sqrt: " « sqrt(sum) « std::endl;
            return sum;
        }
        //template<class Problem, class Grid, class Matrix>
        //const Mesh::Point FEMethodZero<Problem, Grid, Matrix>::GetLambdaGrad(const Mesh::Point& p,
const std::vector<double>& vec) const
        //{
        //  Mesh::Point val{ 0, 0, 0 };
        //  //double val{ 0 };
        //  double diff{ 0 };
        //  Mesh::Point temp{ 0, 0, 0 };
        //  int nfem{ -1 };
        //  nfem = m_Grid->FindElement(p);
        //  if (nfem == -1)
        //      return val;
        //  auto elem = m_Grid->GetElements()[nfem];
        //  diff = std::get<0>(m_Grid->GetDiffusion().find(elem->GetType())->second);
        //  for (int i = 0; i < elem->GetDoF(); ++i)
        //  {
        //      //val += elem->GetGradBasis(i, p) * elem->GetGradBasis(i, p) * vec[elem->GetNodes()[i]]
* vec[elem->GetNodes()[i]] * diff;
        //      //val += elem->GetBasis(i, p) * vec[elem->GetNodes()[i]] * diff;
        //      temp = elem->GetGradBasis(i, p);
        //      val.x += temp.x * vec[elem->GetNodes()[i]] * (diff);
        //      val.y += temp.y * vec[elem->GetNodes()[i]] * (diff);
        //      val.z += temp.z * vec[elem->GetNodes()[i]] * (diff);
        //  }
        //  return val;
        //}
        template<class Problem, class Grid, class Matrix>
        const std::vector<double> FEMethodZero<Problem, Grid, Matrix>::GetRightVector() const
        {
            return *m_rhsvector;
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethodZero<Problem, Grid, Matrix>::OutDatFormat(const Mesh::Point& mn, const
Mesh::Point& mx, const std::string& file_name, const std::vector<double>& vec) const
        {
            std::ofstream of(file_name + "z.dat");
            std::streambuf *buf = std::cout.rdbuf();
```

```
            std::cout.rdbuf(of.rdbuf());
            std::cout « "TITLE = FE-METHOD\n";
            std::cout « "VARIABLES = \"dx1\", \"dx2\", \"u\"\n";
            std::cout « "ZONE i=51, j=51, F=POINT\n";
            double stepx = (mx.x - mn.x) / 51;
            double stepy = (mx.y - mn.y) / 51;
            for (int i = 0; i < 51; ++i)
                for (int j = 0; j < 51; ++j)
                    std::cout « mn.x + j * stepx « "\t" « mn.y + stepy * i « "\t" «
        GetValue(Mesh::Point(mn.x + j * stepx, mn.y + i * stepy, mn.z), vec) « std::endl;
            std::cout.rdbuf(buf);
            of.close();
            of.open(file_name + "x.dat");
            buf = std::cout.rdbuf();
            std::cout.rdbuf(of.rdbuf());
            std::cout « "TITLE = FE-METHOD\n";
            std::cout « "VARIABLES = \"dx1\", \"dx2\", \"u\"\n";
            std::cout « "ZONE i=51, j=51, F=POINT\n";
            for (int i = 0; i < 51; ++i)
                for (int j = 0; j < 51; ++j)
                    std::cout « mn.x + j * stepx « "\t" « mn.y + stepy * i « "\t" «
        GetValue(Mesh::Point(mn.z, mn.x + j * stepx, mn.y + i * stepy), vec) « std::endl;
            std::cout.rdbuf(buf);
            of.close();
            of.open(file_name + "y.dat");
            buf = std::cout.rdbuf();
            std::cout.rdbuf(of.rdbuf());
            std::cout « "TITLE = FE-METHOD\n";
            std::cout « "VARIABLES = \"dx1\", \"dx2\", \"u\"\n";
            std::cout « "ZONE i=51, j=51, F=POINT\n";
            for (int i = 0; i < 51; ++i)
                for (int j = 0; j < 51; ++j)
                    std::cout « mn.x + j * stepx « "\t" « mn.y + stepy * i « "\t" «
        GetValue(Mesh::Point(mn.x + j * stepx, mn.z, mn.y + i * stepy), vec) « std::endl;
            std::cout.rdbuf(buf);
            of.close();
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethodZero<Problem, Grid, Matrix>::ApplySources()
        {
            int nfem = -1;
            auto total = m_problem->get_total_sources();
            for (int i = 0; i < total; ++i)
            {
                auto src = m_problem->get_point_source(i);
                auto point = src.get_point();
                nfem = m_Grid->FindElement(point);
                if (nfem != -1)
                {
                    auto val = src.get_value();
                    auto elem = m_Grid->GetElement(nfem);
                    for (int j = 0; j < 3; ++j)
                        m_rhsvector->operator[](elem->GetNode(j)) += val * elem->GetShapeFunction(j,
        point);
                }
                nfem = -1;
            }
            /*for (auto srd : m_Grid->GetDottedSources())
            {
                nfem = m_Grid->FindElement(srd.first);
                if (nfem != -1)
                {
                    auto elem = m_Grid->GetElements()[nfem];
                    for (int i = 0; i < elem->GetDoF(); ++i)
                    {
                        m_rhsvector[elem->GetNodes()[i]] += srd.second * elem->GetBasis(i, srd.first);
                    }
                }
                nfem = -1;
            }*/
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethodZero<Problem, Grid, Matrix>::Rediscretization(const std::shared_ptr<Grid>& grid)
        {
            m_GlobalMatrix->NullMatrix();
            for (unsigned int i = 0; i < m_rhsvector->size(); ++i)
                (*m_rhsvector)[i] = 0;
            AssemblGlobal();
            //SecondConditions();
            //ApplySources();
            //StefanConditions();
            MainConditions();
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethodZero<Problem, Grid, Matrix>::Rediscretization()
        {
            m_time += m_step;
```

```
            m_GlobalMatrix->NullMatrix();
            for (unsigned int i = 0; i < m_rhsvector->size(); ++i)
                (*m_rhsvector)[i] = 0;
            AssemblGlobal();
            SecondConditions();
            ThirdConditions();
            StefanConditions();
            //ApplySources();
            MainConditions();
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethodZero<Problem, Grid, Matrix>::GetSolution(std::vector<double>& vec)
        {
            int size = vec.size();
            //Translation(vec);
            for (int i = 0; i < size; ++i)
                vec[i] = m_solution[i];
        }
        template<class Problem, class Grid, class Matrix>
        const double FEMethodZero<Problem, Grid, Matrix>::GetSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p)
        {
            double sum{ 0 };
            auto nfem{ g.FindElement(p) };
            if (nfem < 0)
                return 0.;
            auto elem{ g.GetElement(nfem) };
            auto dofs{ elem->GetDoFs() };
            for (auto i{ 0 }; i < dofs; ++i)
                sum += weights[elem->GetNode(i)] * elem->GetShapeFunction(i, p);
            return sum;
        }
        template<class Problem, class Grid, class Matrix>
        const double FEMethodZero<Problem, Grid, Matrix>::GetSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p, const int nfem)
        {
            double sum{ 0 };
            //if (nfem < 0)
            //    return 0.;
            auto elem{ g.GetElement(nfem) };
            auto dofs{ elem->GetDoFs() };
            //std::cout « nfem « std::endl;
            for (auto i{ 0 }; i < dofs; ++i)
                sum += weights[elem->GetNode(i)] * elem->GetShapeFunction(i, p);
            return sum;
        }
        template<class Problem, class Grid, class Matrix>
        const Mesh::Point FEMethodZero<Problem, Grid, Matrix>::GetGradSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p)
        {
            Mesh::Point sum{ 0, 0, 0 };
            auto nfem{ g.FindElement(p) };
            auto elem{ g.GetElement(nfem) };
            auto dofs{ elem->GetDoFs() };
            for (auto i{ 0 }; i < dofs; ++i)
                sum += weights[elem->GetNode(i)] * elem->GetGradShapeFunction(i, p);
            return sum;
        }
        template<class Problem, class Grid, class Matrix>
        const Mesh::Point FEMethodZero<Problem, Grid, Matrix>::GetGradSolution(const Grid& g, const std::vector<double> &weights, const Mesh::Point& p, const int nfem)
        {
            Mesh::Point sum{ 0, 0, 0 };
            auto elem{ g.GetElement(nfem) };
            auto dofs{ elem->GetDoFs() };
            for (auto i{ 0 }; i < dofs; ++i)
                sum += weights[elem->GetNode(i)] * elem->GetGradShapeFunction(i, p);
            return sum;
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethodZero<Problem, Grid, Matrix>::LoadSolution(const std::vector<double>& vec)
        {
            m_solution.resize(vec.size());
            for (unsigned int i = 0; i < vec.size(); ++i)
                m_solution[i] = vec[i];
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethodZero<Problem, Grid, Matrix>::OutMeshFormat(const std::string& file_name, const std::vector<double>& vec)
        {
            const int size{ (int)m_Grid->GetNodes().size() };
            const int number{ (int)m_Grid->GetElements().size() };
            //const int size{ number * 4 };
            std::ofstream ofs(file_name + ".dat", std::ios::out);
            std::string title("TITLE = \"Mesh data\"\n Variables = \"X\", \"Y\", \"Z\", \"U\"\n Zone N = " + std::to_string(size) + ", E = " + std::to_string(number) + ", DATAPACKING = POINT, ZONETYPE = FETETRAHEDRON\n");
```

```
            ofs « title;
            Mesh::Point p;
            for (int i = 0; i < size; ++i)
            {
                p = m_Grid->GetNodes()[i];
                ofs « p.x « "\t" « p.y « "\t" « p.z « "\t" « GetValue(p, vec, 1) « std::endl;
            }
            for (int i = 0; i < number; ++i)
            {
                auto elem = m_Grid->GetElements()[i];
                for (int k = 0; k < 4; ++k)
                {
                    ofs « elem->GetNodes()[k] + 1 « "\t";
                }
                ofs « std::endl;
            }
            ofs.close();
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethodZero<Problem, Grid, Matrix>::OutMeshTimeFormat(const std::string& file_name, const std::vector<double>& vec)
        {
            const int size{ (int)m_Grid->GetNodes().size() };
            const int number{ (int)m_Grid->GetElements().size() };
            //const int size{ number * 4 };
            std::ofstream ofs(file_name + ".dat", std::ios::out | std::ios::app);
            std::string title("TITLE = \"Mesh data\"\n Variables = \"X\", \"Y\", \"Z\", \"U\"\n Zone N = " + std::to_string(size) + ", E = " + std::to_string(number) + ", DATAPACKING = POINT, ZONETYPE = FETETRAHEDRON\n");
            ofs « title;
            Mesh::Point p;
            for (int i = 0; i < size; ++i)
            {
                p = m_Grid->GetNodes()[i];
                ofs « p.x « "\t" « p.y « "\t" « p.z « "\t" « GetValue(p, vec, 1) « std::endl;
            }
            for (int i = 0; i < number; ++i)
            {
                auto elem = m_Grid->GetElements()[i];
                for (int k = 0; k < 4; ++k)
                {
                    ofs « elem->GetNodes()[k] + 1 « "\t";
                }
                ofs « std::endl;
            }
            ofs.close();
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethodZero<Problem, Grid, Matrix>::ProjectSolution(std::vector<double>& sol, std::function<const double(const Mesh::Point&, const std::vector<double>&, const int)> GetVal, std::vector<double>& vec)
        {
            for (int i = 0; i < m_Grid->GetElements().size(); ++i)
            {
                auto elem = m_Grid->GetElements()[i];
                int order = elem->GetDoF();
                for (int j = 0; j < order; ++j)
                    sol[elem->GetNodes(j)] = GetVal(m_Grid->GetNodes()[elem->GetNodes(j)], vec, i);
            }
        }
        template<class Problem, class Grid, class Matrix>
        void FEMethodZero<Problem, Grid, Matrix>::ProjectSolution(std::vector<double>& sol, std::function<const double(const Mesh::Point&, const std::vector<double>&)> GetVal, std::vector<double>& vec, const int)
        {
            for (int i = 0; i < m_Grid->GetElements().size(); ++i)
            {
                auto elem = m_Grid->GetElements()[i];
                int order = elem->GetDoF();
                for (int j = 0; j < order; ++j)
                    sol[elem->GetNodes(j)] = GetVal(m_Grid->GetNodes()[elem->GetNodes(j)], vec);
            }
        }
        template<class Problem, class Grid, class Matrix>
        const std::vector<double> FEMethodZero<Problem, Grid, Matrix>::SetSolution(const int sol, const int liq, const double s, const double l, const double m)
        {
            int i;
            m_solution.resize(m_Grid->GetNodes().size());
            for (i = 0; i < m_Grid->GetElements().size(); ++i)
            {
                auto elem = m_Grid->GetElements()[i];
                int order = elem->GetDoF();
                if (m_Grid->GetElements()[i]->GetType() == liq)
                    for (int j = 0; j < order; ++j)
                        m_solution[elem->GetNodes()[j]] = l;
                else
```

```
                    for (int j = 0; j < order; ++j)
                        m_solution[elem->GetNodes()[j]] = s;
            }

            for (auto bnd : m_Grid->GetBoundaryConditions())
            {
                //if (get<0>(bnd.second) == 4)
                {
                    for (auto row : m_Grid->GetBoundary())
                    {
                        if (bnd.first == row->GetType())
                        {
                            int dofs = row->GetDoF();
                            for (int i = 0; i < dofs; ++i)
                            {
                                m_solution[row->GetNodes(i)] = m;
                            }
                        }
                    }
                }
            }
            return m_solution;
        }
        template<class Problem, class Grid, class Matrix>
        FEMethodZero<Problem, Grid, Matrix>::~FEMethodZero()
        {
            delete m_Grid;
        }
    }
}

#endif // !CORENC_METHODS_FEMethodZeroZero_h

```

7.83 CoreNCFEM/Methods/FVMethod.cpp File Reference

```
#include "FVMethod.h"
```

7.84 CoreNCFEM/Methods/FVMethod.h File Reference

```
#include "../Grids/Mesh1D.h"
```

Classes

- class corenc::method::FVMethod1d

Namespaces

- namespace corenc
- namespace corenc::method

Enumerations

- enum class corenc::method::FVFlux { corenc::method::LaxFriedrichs , corenc::method::Upwind , corenc::method::Central , corenc::method::NOFLUX }

7.85 FVMethod.h

Go to the documentation of this file.

```
#ifndef CORENC_METHODS_FINITEVOLUME_H_
#define CORENC_METHODS_FINITEVOLUME_H_

#include "../Grids/Mesh1D.h"

namespace corenc
{
    namespace method
    {
        enum class FVFlux
        {
            LaxFriedrichs,
            Upwind,
            Central,
            NOFLUX,
        };
        class FVMethod1d
        {
        public:
            FVMethod1d();
            ~FVMethod1d();
            static const int                              Solve(Mesh::CMesh<CFESolution>* mesh,
                                                          const std::function<const double(const double)>&
       flux_func,
                                                          const FVFlux& flux_type,
                                                          std::vector<double>& new_solution,
                                                          const double time_step);
            static const double                           GetSolution(const Mesh::CMesh1D& g, const
       Mesh::Point& p);
        };
    }
}
#endif // CORENC_METHODS_FINITEVOLUME_H_
```

7.86 CoreNCFEM/Methods/RungeKutta.h File Reference

```
#include <memory>
#include "../Point.h"
#include <functional>
```

Classes

- class corenc::method::RungeKutta< Problem, Type >

Namespaces

- namespace corenc
- namespace corenc::method

7.87 RungeKutta.h

Go to the documentation of this file.

```
#ifndef CORENC_METHODS_RUNGEKUTTA
#define CORENC_METHODS_RUNGEKUTTA

#include <memory>
#include "../Point.h"
#include <functional>
namespace corenc
```

```
{
    namespace method
    {
        template<class Problem, class Type>
        class RungeKutta
        {
        public:
            RungeKutta() {};
            RungeKutta(const double step, const double final, Problem* problem, const Type* solution) :
                m_step{ step },
                m_final{ final },
                m_problem{problem} {}
            const Type              discretize(const Type& solution, const std::function<const Type(const double time, const double time_step, const Type& curr_sol, Type* result)>& func);
            const Type              explicitEuler(const Type& solution, const std::function<const Type(const double time, const double time_step, const Type& curr_sol, Type* result)>& func);
            void                    updateTimestep(const double step) { m_step = step; };
            ~RungeKutta() {};
        private:
            double                  m_step;
            double                  m_final;
            double                  m_curr;
            Problem*                m_problem;
            Type*                   m_solution;
            static const std::vector<double> vector_mult(const std::vector<double>& lhs, const double rhs)
            {
                std::vector<double> vc(lhs);
                for (auto &it : vc)
                    it *= rhs;
                return vc;
            }

            static const std::vector<double> vector_mult(const double rhs, const std::vector<double>& lhs)
            {
                std::vector<double> vc(lhs);
                for (auto &it : vc)
                    it *= rhs;
                return vc;
            }

            static const std::vector<double> vector_divide(const std::vector<double>& lhs, const double rhs)
            {
                std::vector<double> vc(lhs);
                for (auto &it : vc)
                    it /= rhs;
                return vc;
            }

            static const std::vector<double> vector_divide(const double rhs, const std::vector<double>& lhs)
            {
                std::vector<double> vc(lhs);
                for (auto &it : vc)
                    it /= rhs;
                return vc;
            }

            static const std::vector<double> vector_add(const std::vector<double>& rhs, const std::vector<double>& lhs)
            {
                std::vector<double> vc(lhs);
                for (unsigned i{ 0 }; i < vc.size(); ++i)
                    vc[i] += rhs[i];
                return vc;
            }
        };



        template<class Problem, class Type>
        const Type RungeKutta<Problem, Type>::discretize(const Type& u_pr, const std::function<const Type(const double time, const double time_step, const Type& curr_sol, Type* result)>& func)
        {
            Type k[4];
            const int n{ int(m_final / m_step) };
            func(m_curr, m_step, u_pr, &k[0]);
            //std::vector<double> tempc(m_curr.size());
            std::vector<double> tempu(u_pr.size());
            std::vector<double> tempk(u_pr.size());
            tempk = vector_divide(k[0], 2);
            tempu = vector_add(u_pr, tempk);
            func(m_curr + m_step / 2, m_step, tempu, &k[1]);
            //func(m_curr + m_step / 2, m_step, u_pr + k[0] / 2, &k[1]);

```

```
            tempk = vector_divide(k[1], 2);
            tempu = vector_add(u_pr, tempk);
            func(m_curr + m_step / 2, m_step, tempu, &k[2]);
            //func(m_curr + m_step / 2, m_step, u_pr + k[1] / 2, &k[2]);

            tempu = vector_add(u_pr, k[2]);
            func(m_curr + m_step, m_step, tempu, &k[3]);
            //func(m_curr + m_step, m_step, u_pr + k[2], &k[3]);

            tempk = vector_mult(k[1], 2);
            tempu = vector_mult(k[2], 2);
            k[3] = vector_add(k[3], tempu);
            k[3] = vector_add(k[3], tempk);
            k[3] = vector_add(k[3], k[0]);
            k[3] = vector_divide(k[3], 6.);
            //k[3] = k[0] + 2 * k[1] + 2 * k[2] + k[3];
            //k[3] = 1. / 6 * k[3];
            m_problem->addTerm(Terms::EUV);
            m_problem->addTerm(Terms::IUV);
            m_curr += m_step;
            return k[3];
        }

        template<class Problem, class Type>
        const Type RungeKutta<Problem, Type>::explicitEuler(const Type& u_pr, const std::function<const Type(const double time, const double time_step, const Type& curr_sol, Type* result)>& func)
        {
            Type k;
            func(m_curr, m_step, u_pr, &k);
            m_problem->addTerm(Terms::EUV);
            m_problem->addTerm(Terms::IUV);
            m_curr += m_step;
            return k;
        }
    }
}
#endif // !CORENC_METHODS_RUNGEKUTTA
```

7.88 CoreNCFEM/Methods/system_dg_method.h File Reference

```
#include <functional>
#include <set>
#include "../Point.h"
#include <memory>
#include <cmath>
#include "FEMethod.h"
#include <map>
#include <algorithm>
#include <vector>
#include "dg_flux.h"
```

Classes

- class corenc::method::system_dg_method< Problem, Grid, Matrix >
- class corenc::method::system_dg_method< Grid, bool, bool >

Namespaces

- namespace corenc
- namespace corenc::method

Macros

- #define CORENC_METHODS_SYSTEM_DG_METHOD_H_

7.88.1 Macro Definition Documentation

7.88.1.1 CORENC_METHODS_SYSTEM_DG_METHOD_H_

```
#define CORENC_METHODS_SYSTEM_DG_METHOD_H_
```

7.89 system_dg_method.h

Go to the documentation of this file.

```
// NO GENERALIZATION HERE
// JUST PLAIN DG FOR SYSTEM IN N - DIMENSIONAL SPACE FOR ONE TIME STEP
// CONSTANT BASIS FUNCTIONS

#pragma once
#ifndef CORENC_METHODS_SYSTEM_DG_METHOD_H_
#define CORENC_METHODS_SYSTEM_DG_METHOD_H_
#include <functional>
#include <set>
#include "../Point.h"
#include <memory>
#include <cmath>
#include "FEMethod.h"
#include <map>
#include <algorithm>
#include <vector>
#include "dg_flux.h"

namespace corenc
{
    namespace method
    {
        template<class Problem, class Grid, class Matrix>
        class system_dg_method
        {
        public:
            system_dg_method() :
                m_problem{ nullptr },
                m_CoarseGrid{ nullptr },
                m_GlobalMatrix{ nullptr },
                m_rhsvector{ nullptr }
            {};
            system_dg_method(
                Problem* p,
                Grid* g,
                Matrix* m,
                //Solution* s,
                const size_t sys_size,
                std::vector<double>* rhs):
                //const std::function<const double(const double)>& flux_function,
                //const DGFlux flux_type) :
                m_problem{ p },
                m_CoarseGrid{ g },
                m_GlobalMatrix{ m },
                m_N{ g->GetNumberOfElements() },
                m_Ns{ g->GetNumberOfBoundaries() },
                m_rhsvector{ rhs },
                //m_flux(flux_function),
                m_sys_size{sys_size}{
                GeneratePortrait();
            }
            ~system_dg_method() {};
            const int                   Assemble();
            const int                   changeFlux(const DGFlux flux_type) { m_fluxtype = flux_type; return 0; };
            const Matrix*               GetGlobalMatrix() const { return m_GlobalMatrix; };
            const std::vector<double>   GetSolution() const { return m_vec; };
            const double                GetSolution(const std::vector<double>& point) const;
            const double                GetMaxSolution() const;
            const double                GetMinSolution() const;
            static const double         GetSolution(const Grid& g, const std::vector<double> &dg_sol, const Mesh::Point& p)
            {
                double sum{ 0 };
```

```
            auto nfem{ g.FindElement(p) };
            auto elem{ g.GetElement(nfem) };
            auto dofs{ elem->GetDoFs() };
            for (auto i{ 0 }; i < dofs; ++i)
            {
                sum += dg_sol[nfem * dofs + i] * elem->GetShapeFunction(i, p);
            }
            return sum;
        }
        const double            GetSolution(const std::vector<double> &dg_sol, const Mesh::Point& p)
        {
            double sum{ 0 };
            auto nfem{ m_CoarseGrid->FindElement(p) };
            auto elem{ m_CoarseGrid->GetElement(nfem) };
            auto dofs{ elem->GetDoFs() };
            for (auto i{ 0 }; i < dofs; ++i)
            {
                sum += dg_sol[nfem * dofs + i] * elem->GetShapeFunction(i, p);
            }
            return sum;
        }
        const int                   toDGSolution(const Grid& g, std::vector<double>& dg_result) const
        {
            //dg_result->resize(m_rhsvector->size());
            dg_result.resize(m_rhsvector->size());
            for (unsigned i{ 0 }; i < g.GetNumberOfElements(); ++i)
            {
                auto elem{ g.GetElement(i) };
                auto dofs{ elem->GetDoFs() };
                for (unsigned j{ 0 }; j < dofs; ++j)
                    //dg_result->operator[](m_nums[i] + j) = g.getSolution(i, j);
                    dg_result[m_nums[i] + j] = g.getSolution(i, j);
            }
            return 0;
        }
        const int                   updateWeights(const std::vector<double>& dg_result)
        {
             for (unsigned int i{ 0 }; i < (unsigned int)m_CoarseGrid->GetNumberOfElements(); ++i)
             {
                 for (unsigned int j{ 0 }; j < (unsigned int)m_CoarseGrid->GetElement(i)->GetDoFs();
    ++i)
                     m_CoarseGrid->updateSolution(i, j, dg_result[m_nums[i] + j]);
             }
             return 0;
         }

         const int                   DGtostandart(const std::vector<double>& dg_result)
         {
             for (unsigned int i{ 0 }; i < (unsigned int)m_CoarseGrid->GetNumberOfElements(); ++i)
             {
                 auto elem{ m_CoarseGrid->GetElement(i) };
                 auto dofs{ elem->GetDoFs() };
                 for (unsigned int j{ 0 }; j < (unsigned int)dofs; ++j)
                     //m_CoarseGrid->updateSolution(i, j, dg_result[m_nums[i] + j]);
                     m_CoarseGrid->updateSolution(i, j, dg_result[m_nums[i] + j]);
             }
             return 0;
         }
     private:
         const int                   GeneratePortrait();
         void                        assembleBoundaries();
         void                        assemble_flux(const unsigned boundary);
         const double                numerical_flux(const double ul, const double ur, const double
    fl, const double fr) const;
         void                        MainConditions();
         void                        SecondConditions();
         void                        ThirdConditions();
         const int                   AssembleGlobal();
         const int                   AssembleFluxMatrix();
         Grid*                       m_CoarseGrid;
         Matrix*                     m_GlobalMatrix;
         std::vector<double>*        m_rhsvector;
         std::vector<unsigned int>   m_nums;
         unsigned int                m_N;  // number of elements
         unsigned int                m_Ns; // number of boundaries
         unsigned int                m_size;
         Problem*                    m_problem;
         std::vector<double>         m_vec;
         std::vector<double>         m_solution;
         //std::function<const Mesh::Point(const Mesh::Point)> m_numflux;
         //std::function<const Mesh::Point(const Mesh::Point)> m_flux;
         DGFlux                        m_fluxtype;
         size_t                      m_sys_size;
         std::function<const double(const double)> m_flux;
         const int                   AssembleLocalMatrix(const int);
     };

```

```
    template<class Grid>
    class system_dg_method<Grid, bool, bool>
    {
    public:
        static const double         GetSolution(const Grid& g, const std::vector<double> &dg_sol, const Mesh::Point& p)
        {
            double sum{ 0 };
            auto nfem{ g.FindElement(p) };
            auto elem{ g.GetElement(nfem) };
            auto dofs{ elem->GetDoFs() };
            for (auto i{ 0 }; i < dofs; ++i)
            {
                sum += dg_sol[nfem * dofs + i] * elem->GetShapeFunction(i, p);
            }
            return sum;
        }
    };

    template<class Problem, class  Grid, class Matrix>
    const int system_dg_method<Problem, Grid, Matrix>::Assemble()
    {
        //GeneratePortrait();
        AssembleGlobal();
        AssembleFluxMatrix();
        MainConditions();
        SecondConditions();
        ThirdConditions();
        return 0;
    }
    template<class Problem, class Grid, class Matrix>
    const int system_dg_method<Problem, Grid, Matrix>::GeneratePortrait()
    {
        int lorder, rorder, order;
        std::vector<std::set<unsigned int» temp;
        unsigned int i, j, nk, ne, k, sz, size;
        m_size = 0;
        m_nums.resize(m_N * m_sys_size);

        nk = 0;
        sz = m_N * m_sys_size;
        for (i = 0, k = 0; k < sz; ++i, k += m_sys_size)
        {
            size = m_CoarseGrid->GetElement(i)->GetDoFs() * m_sys_size;
            for(j = 0; j < m_sys_size; ++j)
                m_nums[k + j] = m_size + j * m_CoarseGrid->GetElement(i)->GetDoFs();
            m_size += size;
        }
        temp.resize(m_size);
        sz = m_Ns;
        for (k = 0; k < sz; k += m_sys_size)
        {
            auto bound = m_CoarseGrid->GetBoundary(k);
            nk = bound->GetNeighbour(0);
            ne = bound->GetNeighbour(1);
            lorder = m_CoarseGrid->GetElement(nk)->GetDoFs();
            if (ne != -1)
            {

                rorder = m_CoarseGrid->GetElement(ne)->GetDoFs();
                for (i = 0; i < lorder; ++i)
                    for (j = 0; j < rorder; ++j)
                        temp[m_nums[ne] + j].insert(m_nums[nk] + i);
                for (i = 0; i < lorder; ++i)
                    for (j = i + 1; j < lorder; ++j)
                        temp[m_nums[nk] + j].insert(m_nums[nk] + i);
            }
            else
            {
                for (i = 0; i < lorder; ++i)
                    for (j = i + 1; j < lorder; ++j)
                        temp[m_nums[nk] + j].insert(m_nums[nk] + i);
            }
        }

        /*temp.resize(m_CoarseGrid->GetNumberOfNodes());
        m_nums.resize(m_CoarseGrid->GetNumberOfNodes());
        lorder = m_CoarseGrid->GetElement(0)->GetDoFs();
        for (k = 0; k < m_CoarseGrid->GetNumberOfNodes(); ++k)
        m_nums[k] = k;
        //for (auto elem : m_CoarseGrid->GetElements())
        for(k = 0; k < m_CoarseGrid->GetNumberOfElements(); ++k)
        {
        auto elem{ m_CoarseGrid->GetElement(k) };
        auto order{ elem->GetDoFs() };
        for (i = 0; i < order; ++i)
        for (j = 0; j < order; ++j)
```

```
    if (elem->GetNode(j) > elem->GetNode(i))
    temp[elem->GetNode(j)].insert(elem->GetNode(i));
    }*/
    m_GlobalMatrix->Create(temp.size(), temp);
    m_rhsvector->resize(temp.size());
    //m_vec.resize(temp.size());
    return 0;
}

template<class Problem, class Grid, class Matrix>
const int system_dg_method<Problem, Grid, Matrix>::AssembleLocalMatrix(const int l)
{
    int i, j, k, nodes;
    double mij;
    const auto& elem{ m_CoarseGrid->GetElement(l) };
    const auto& dofs{ elem->GetDoFs() };
    nodes = elem->GetNumberOfNodes();
    std::vector<Mesh::Point> points(nodes);
    for (i = 0; i < nodes; ++i)
        points[i] = m_CoarseGrid->GetNode(elem->GetNode(i));
    for (k = 0; k < m_problem->getNumberOfTerms(); ++k)
    {
        switch (m_problem->getTerm(k))
        {
        case Terms::EUV:
            //for (i = 0; i < dofs; ++i)
            //{
            //  for (j = 0; j < dofs; ++j)
            //  {
            //      auto M = [&](const Mesh::Point& p)
            //      {
            //          return elem->GetShapeFunction(i, p) * elem->GetShapeFunction(j, p);
            //      };
            //      mij = elem->Integrate(M, points);
            //      m_rhsvector->operator[](m_nums[l] + i) += m_CoarseGrid->getParameter(Parameters::MASS, l, j) * m_CoarseGrid->getSolution(l, j) * mij;
            //  }
            //}
            for(size_t j = 0; j < m_sys_size; ++j)
                m_rhsvector->operator[](m_nums[l] + j) += m_problem->get_solution(j, l, elem->GetType(), points[l]);
            break;
        default:
            break;
        }
    }
    return 0;
}

template<class Problem, class Grid, class Matrix>
const int system_dg_method<Problem, Grid, Matrix>::AssembleGlobal()
{
    for (int l = 0; l < m_N; ++l)
        AssembleLocalMatrix(l);
    return 0;
}

template<class Problem, class Grid, class Matrix>
const int system_dg_method<Problem, Grid, Matrix>::AssembleFluxMatrix()
{
    auto Nb{ m_CoarseGrid->GetNumberOfBoundaries() };
    unsigned int l;
    switch (m_fluxtype)
    {
    case corenc::method::DGFlux::ELaxFriedrichs:

        for (l = 0; l < Nb; ++l)
        {
            const auto& bound{ m_CoarseGrid->GetBoundary(l) };
            const auto& nk{ bound->GetNeighbour(0) };
            const auto& ne{ bound->GetNeighbour(1) };
            const auto& elemk{ m_CoarseGrid->GetElement(nk) };
            const auto& dofs{ bound->GetDoFs() };
            const auto& dofsk{ elemk->GetDoFs() };
            double C{ 0 };
            unsigned int i, j;
            std::vector<Mesh::Point> points(dofs);
            for (i = 0; i < dofs; ++i)
                points[i] = m_CoarseGrid->GetNode(bound->GetNode(i));
            if (ne > -1)
            {
                const auto& eleme{ m_CoarseGrid->GetElement(ne) };
                const auto& dofse{ eleme->GetDoFs() };
                for (i = 0; i < dofsk; ++i)
                {
                    for (j = 0; j < dofsk; ++j)
                    {
```

```
                            auto Mkk = [&](const Mesh::Point& p)
                            {
                                return elemk->GetShapeFunction(j, p) * elemk->GetShapeFunction(i, p);
                            };
                            auto temp{ bound->Integrate(Mkk, points) };
                            C = std::max(fabs(m_CoarseGrid->getSolution(ne, i)), fabs(m_CoarseGrid->getSolution(nk, j)));
                            //m_rhsvector->operator[](m_nums[nk] + i) += -0.5*(m_flux(m_CoarseGrid->getSolution(nk, j)) * temp - C * m_CoarseGrid->getSolution(nk, j) * temp);
                            auto val{ -0.5*(m_flux(m_CoarseGrid->getSolution(nk, j)) + C * m_CoarseGrid->getSolution(nk, j)) * temp };
                            m_rhsvector->operator[](m_nums[nk] + i) += val;
                            //lv[m_nums[nk] + i] += val;
                        }
                    }
                    for (i = 0; i < dofsk; ++i)
                    {
                        for (j = 0; j < dofse; ++j)
                        {
                            auto Mke = [&](const Mesh::Point& p)
                            {
                                return eleme->GetShapeFunction(j, p) * elemk->GetShapeFunction(i, p);
                            };
                            auto temp{ bound->Integrate(Mke, points) };
                            C = std::max(fabs(m_CoarseGrid->getSolution(nk, i)), fabs(m_CoarseGrid->getSolution(ne, j)));
                            //m_rhsvector->operator[](m_nums[nk] +7 i) += -0.5*(m_flux(m_CoarseGrid->getSolution(ne, j)) * temp - C * m_CoarseGrid->getSolution(ne, j) * temp);
                            auto val{ -0.5*(m_flux(m_CoarseGrid->getSolution(ne, j)) - C * m_CoarseGrid->getSolution(ne, j)) * temp };
                            m_rhsvector->operator[](m_nums[nk] + i) += val;
                            //ke[m_nums[nk] + i] += val;
                            //lv[m_nums[nk] + i] += val;
                        }
                    }
                    for (i = 0; i < dofse; ++i)
                    {
                        for (j = 0; j < dofsk; ++j)
                        {
                            auto Mek = [&](const Mesh::Point& p)
                            {
                                return eleme->GetShapeFunction(i, p) * elemk->GetShapeFunction(j, p);
                            };
                            auto temp{ bound->Integrate(Mek, points) };
                            C = std::max(fabs(m_CoarseGrid->getSolution(nk, j)), fabs(m_CoarseGrid->getSolution(ne, i)));
                            //m_rhsvector->operator[](m_nums[ne] + i) += 0.5*(m_flux(m_CoarseGrid->getSolution(nk, j)) * temp - C * m_CoarseGrid->getSolution(nk, j) * temp);
                            auto val{ 0.5*(m_flux(m_CoarseGrid->getSolution(nk, j)) + C * m_CoarseGrid->getSolution(nk, j)) * temp };
                            m_rhsvector->operator[](m_nums[ne] + i) += val;
                            //ek[m_nums[ne] + i] += val;
                            //rv[m_nums[ne] + i] += val;
                        }
                    }
                    for (i = 0; i < dofse; ++i)
                    {
                        for (j = 0; j < dofse; ++j)
                        {
                            auto Mee = [&](const Mesh::Point& p)
                            {
                                return eleme->GetShapeFunction(j, p) * eleme->GetShapeFunction(i, p);
                            };
                            auto temp{ bound->Integrate(Mee, points) };
                            C = std::max(fabs(m_CoarseGrid->getSolution(nk, i)), fabs(m_CoarseGrid->getSolution(ne, j)));
                            //m_rhsvector->operator[](m_nums[ne] + i) += 0.5*(m_flux(m_CoarseGrid->getSolution(ne, j)) * temp - C * m_CoarseGrid->getSolution(ne, j) * temp);
                            auto val{ 0.5*(m_flux(m_CoarseGrid->getSolution(ne, j)) - C * m_CoarseGrid->getSolution(ne, j)) * temp };
                            m_rhsvector->operator[](m_nums[ne] + i) += val;
                            //ee[m_nums[ne] + i] += val;
                            //rv[m_nums[ne] + i] += val;
                        }
                    }
                }
                else
                {
                    //C = m_flux(m_CoarseGrid->getSolution(nk, 0));
                    //m_rhsvector->operator[](m_nums[nk]) = C;
                    if (l == 0)
                    {
                        for (i = 0; i < dofsk; ++i)
```

```
                        {
                            for (j = 0; j < dofsk; ++j)
                            {
                                auto Mkk = [&](const Mesh::Point& p)
                                {
                                    return elemk->GetShapeFunction(j, p) * elemk->GetShapeFunction(i, p);
                                };
                                auto temp{ bound->Integrate(Mkk, points) };
                                //m_rhsvector->operator[](m_nums[nk]+i) -= ((ne+int(l))>0?-1:1)*(m_flux(m_CoarseGrid->getSolution(nk, j)) * temp - C * m_CoarseGrid->getSolution(nk, j) * temp);
                                auto fl = m_flux(m_CoarseGrid->getSolution(nk, j)) * temp;
                                m_rhsvector->operator[](m_nums[nk] + i) += fl * temp;
                                //if(C >= 0)
                                //m_rhsvector->operator[](m_nums[nk] + i) += ((ne + int(l))>0 ? 1 : 0) * C * temp;
                                //m_rhsvector->operator[](m_nums[nk] + i) += 1e10 * C * temp;
                            }
                        }
                    }
                    else
                    {
                        for (i = 0; i < dofsk; ++i)
                        {
                            for (j = 0; j < dofsk; ++j)
                            {
                                auto Mkk = [&](const Mesh::Point& p)
                                {
                                    return elemk->GetShapeFunction(j, p) * elemk->GetShapeFunction(i, p);
                                };
                                auto temp{ bound->Integrate(Mkk, points) };
                                auto fl = m_flux(m_CoarseGrid->getSolution(nk, j)) * temp;
                                m_rhsvector->operator[](m_nums[nk] + i) -= fl * temp;
                            }
                        }
                    }
                }
            }

            // explicit LF flux
            break;
        default:
            break;
        }
        return 0;
    }

    template<class Problem, class Grid, class Matrix>
    void system_dg_method<Problem, Grid, Matrix>::assemble_flux(const unsigned l)
    {
        const auto& bound{ m_CoarseGrid->GetBoundary(l) };
        const auto& nk{ bound->GetNeighbour(0) };
        const auto& ne{ bound->GetNeighbour(1) };
        const auto& elemk{ m_CoarseGrid->GetElement(nk) };
        const auto& dofs{ bound->GetDoFs() };
        const auto& dofsk{ elemk->GetDoFs() };
        double C{ 0 };
        unsigned int i, j;
        std::vector<Mesh::Point> points(dofs);
        for (i = 0; i < dofs; ++i)
            points[i] = m_CoarseGrid->GetNode(bound->GetNode(i));
        C = 2;
        if (ne > -1)
        {
            const auto& eleme{ m_CoarseGrid->GetElement(ne) };
            const auto& dofse{ eleme->GetDoFs() };
            for (i = 0; i < dofsk; ++i)
            {
                for (j = 0; j < dofsk; ++j)
                {
                    auto Mkk = [&](const Mesh::Point& p)
                    {
                        return elemk->GetShapeFunction(j, p) * elemk->GetShapeFunction(i, p);
                    };
                    auto temp{ bound->Integrate(Mkk, points) };
                    C = std::max(m_CoarseGrid->getSolution(nk, i), m_CoarseGrid->getSolution(nk, j));
                    auto val{ -0.5*(m_flux(m_CoarseGrid->getSolution(nk, j)) + C * m_CoarseGrid->getSolution(nk, j)) * temp };
                    m_rhsvector->operator[](m_nums[nk] + i) += val;
                    //kk[m_nums[nk] + i] += val;
                    //lv[m_nums[nk] + i] += val;
                }
            }
            for (i = 0; i < dofsk; ++i)
```

```
                {
                    for (j = 0; j < dofse; ++j)
                    {
                        auto Mke = [&](const Mesh::Point& p)
                        {
                            return eleme->GetShapeFunction(j, p) * elemk->GetShapeFunction(i, p);
                        };
                        auto temp{ bound->Integrate(Mke, points) };
                        C = std::max(m_CoarseGrid->getSolution(nk, i), m_CoarseGrid->getSolution(ne, j));
                        //m_rhsvector->operator[](m_nums[nk] +7 i) += -0.5*(m_flux(m_CoarseGrid->getSolution(ne, j)) * temp - C * m_CoarseGrid->getSolution(ne, j) * temp);
                        auto val{ -0.5*(m_flux(m_CoarseGrid->getSolution(ne, j)) - C * m_CoarseGrid->getSolution(ne, j)) * temp };
                        m_rhsvector->operator[](m_nums[nk] + i) += val;
                        //ke[m_nums[nk] + i] += val;
                        //rv[m_nums[nk] + i] += val;
                    }
                }
                for (i = 0; i < dofse; ++i)
                {
                    for (j = 0; j < dofsk; ++j)
                    {
                        auto Mek = [&](const Mesh::Point& p)
                        {
                            return eleme->GetShapeFunction(i, p) * elemk->GetShapeFunction(j, p);
                        };
                        auto temp{ bound->Integrate(Mek, points) };
                        C = std::max(m_CoarseGrid->getSolution(nk, j), m_CoarseGrid->getSolution(ne, i));
                        //m_rhsvector->operator[](m_nums[ne] + i) += 0.5*(m_flux(m_CoarseGrid->getSolution(nk, j)) * temp - C * m_CoarseGrid->getSolution(nk, j) * temp);
                        auto val{ 0.5*(m_flux(m_CoarseGrid->getSolution(nk, j)) + C * m_CoarseGrid->getSolution(nk, j)) * temp };
                        m_rhsvector->operator[](m_nums[ne] + i) += val;
                        //ek[m_nums[ne] + i] += val;
                        //lv[m_nums[ne] + i] += val;
                    }
                }
                for (i = 0; i < dofse; ++i)
                {
                    for (j = 0; j < dofse; ++j)
                    {
                        auto Mee = [&](const Mesh::Point& p)
                        {
                            return eleme->GetShapeFunction(j, p) * eleme->GetShapeFunction(i, p);
                        };
                        auto temp{ bound->Integrate(Mee, points) };
                        C = std::max(m_CoarseGrid->getSolution(ne, i), m_CoarseGrid->getSolution(ne, j));
                        //m_rhsvector->operator[](m_nums[ne] + i) += 0.5*(m_flux(m_CoarseGrid->getSolution(ne, j)) * temp - C * m_CoarseGrid->getSolution(ne, j) * temp);
                        auto val{ 0.5*(m_flux(m_CoarseGrid->getSolution(ne, j)) - C * m_CoarseGrid->getSolution(ne, j)) * temp };
                        m_rhsvector->operator[](m_nums[ne] + i) += val;
                        //ee[m_nums[ne] + i] += val;
                        //rv[m_nums[ne] + i] += val;
                    }
                }
            }
            else
            {
                for (i = 0; i < dofsk; ++i)
                {
                    for (j = 0; j < dofsk; ++j)
                    {
                        auto Mkk = [&](const Mesh::Point& p)
                        {
                            return elemk->GetShapeFunction(j, p) * elemk->GetShapeFunction(i, p);
                        };
                        auto temp{ bound->Integrate(Mkk, points) };
                        auto fl = m_flux(m_CoarseGrid->getSolution(nk, j));
                        C = m_CoarseGrid->getSolution(nk, j);
                        m_rhsvector->operator[](m_nums[nk] + i) += ((ne + int(1))>0 ? 0 : 1) *fl * temp;
                    }
                }
            }
        }
        template<class Problem, class Grid, class Matrix>
        void system_dg_method<Problem, Grid, Matrix>::MainConditions()
        {

        }

        template<class Problem, class Grid, class Matrix>
        void system_dg_method<Problem, Grid, Matrix>::SecondConditions()
        {
```

```
        }

        template<class Problem, class Grid, class Matrix>
        void system_dg_method<Problem, Grid, Matrix>::ThirdConditions()
        {

        }

        template<class Problem, class Grid, class Matrix>
        const double system_dg_method<Problem, Grid, Matrix>::GetMaxSolution() const
        {
            return 0.;
        }

        template<class Problem, class Grid, class Matrix>
        const double system_dg_method<Problem, Grid, Matrix>::GetMinSolution() const
        {
            return 0.;
        }

        template<class Problem, class Grid, class Matrix>
        const double system_dg_method<Problem, Grid, Matrix>::GetSolution(const std::vector<double>&
    point) const
        {
            return 0.;
        }
    }
}
#endif // !CORENC_METHODS_SYSTEM_DG_METHOD_H_
```

7.90 CoreNCFEM/multi_vector.h File Reference

```
#include <vector>
#include <cstdarg>
#include <cstddef>
```

Classes

- class corenc::multi_vector< T >

Namespaces

- namespace corenc

Macros

- #define CORENC_MULTI_VECTOR_H_

7.90.1 Macro Definition Documentation

7.90.1.1 CORENC_MULTI_VECTOR_H_

```
#define CORENC_MULTI_VECTOR_H_
```

7.91 multi_vector.h

Go to the documentation of this file.

```
#pragma once
#ifndef CORENC_MULTI_VECTOR_H_
#define CORENC_MULTI_VECTOR_H_
#include <vector>
#include <cstdarg>
#include <cstddef>
namespace corenc
{
    template<class T>
     class multi_vector
     {
     public:
         multi_vector();
         // dim = 1 vector, dim = 2 matrix, etc
         // block x ... x block; dim times
         multi_vector(const size_t block, const size_t dim);
         multi_vector(const size_t dim);
         ~multi_vector();
         const T             get(const size_t i...) const;
         const T             get(const std::vector<size_t>& i) const;
         const int           set(const T& element, const std::vector<size_t>& index);
         //const int         set(const T& element, const size_t i...);
         const int           fill_inc();
         void                resize(const size_t block);
         void                resize(const size_t block, const size_t dim);
         const size_t        size() const;
         const size_t        totalsize() const;
     private:
         std::vector<T>      m_vector;
         size_t              m_dim;
         size_t              m_block;
         size_t              m_totalsize;
     };

     template<class T>
     multi_vector<T>::multi_vector()
     {

     }
     template<class T>
     multi_vector<T>::multi_vector(const size_t block, const size_t dim)
     {
         m_block = block;
         m_dim = dim;
         m_totalsize = 1;
         for (size_t i = 0; i < m_dim; ++i, m_totalsize *= block);
         m_vector.resize(m_totalsize);
     }
     template<class T>
     multi_vector<T>::multi_vector(const size_t dim)
     {
         m_block = 0;
         m_dim = dim;
         m_totalsize = 0;
     }
     template<class T>
     multi_vector<T>::~multi_vector()
     {

     }
     template<class T>
     const size_t multi_vector<T>::size() const
     {
         return m_block;
     }
     template<class T>
     const size_t multi_vector<T>::totalsize() const
     {
         return m_totalsize;
     }
     template<class T>
     const T multi_vector<T>::get(const size_t i...) const
     {
         va_list args;
         va_start(args, i);

         va_end(args);
         return m_vector[i];
     }
     template<class T>
     const T multi_vector<T>::get(const std::vector<size_t>& i) const
     {
```

```
        if (i.size() != m_dim)
            return T(0);
        size_t ind = 0;
        for (size_t j = 0; j < m_dim; ++j)
        {
            size_t l = 1;
            const int lim = m_dim - j - 1;
            for (int k = 0; k < lim; ++k, l *= m_block);
            ind += i[j] * l;
        }
        return m_vector[ind];
    }
    template<class T>
    void multi_vector<T>::resize(const size_t block)
    {
        m_block = block;
        m_totalsize = 1;
        for (size_t i = 0; i < m_dim; ++i, m_totalsize *= block);
        m_vector.resize(m_totalsize);
    }
    template<class T>
    void multi_vector<T>::resize(const size_t block, const size_t dim)
    {
        m_block = block;
        m_dim = dim;
        m_totalsize = 1;
        for (size_t i = 0; i < m_dim; ++i, m_totalsize *= block);
        m_vector.resize(m_totalsize);
    }
    template<class T>
    const int multi_vector<T>::fill_inc()
    {
        for (size_t i = 0; i < m_totalsize; ++i)
            m_vector[i] = i;
        return 0;
    }
    template<class T>
    const int multi_vector<T>::set(const T& element, const std::vector<size_t>& i)
    {
        if (i.size() != m_dim)
            return 1;
        size_t ind = 0;
        for (size_t j = 0; j < m_dim; ++j)
        {
            size_t l = 1;
            const int lim = m_dim - j - 1;
            for (int k = 0; k < lim; ++k, l *= m_block);
            ind += i[j] * l;
        }
        m_vector[ind] = element;
        return 0;
    }
}
#endif // !CORENC_MULTI_VECTOR_H_
```

7.92 CoreNCFEM/Parameter.cpp File Reference

```
#include "Parameter.h"
```

7.93 CoreNCFEM/Parameter.h File Reference

```
#include "Point.h"
#include <functional>
```

Classes

- class corenc::Mesh::parameter< T >
- class corenc::Mesh::point_source< T >
- class corenc::Mesh::CParameter

Namespaces

- namespace corenc
- namespace corenc::Mesh

Macros

- #define CORENC_MESH_PARAMETER_H_

7.93.1 Macro Definition Documentation

7.93.1.1 CORENC_MESH_PARAMETER_H_

```
#define CORENC_MESH_PARAMETER_H_
```

7.94 Parameter.h

Go to the documentation of this file.

```
// OK. DESCRIPTION.
// Here the known parameters are described. it is used then with meshes and problems etc.
#pragma once
#ifndef CORENC_MESH_PARAMETER_H_
#define CORENC_MESH_PARAMETER_H_

#include "Point.h"
#include <functional>
namespace corenc
{
    namespace Mesh
    {
        template<class T>
        class parameter
        {
        public:
            using cfunc = std::function<const T(const int, const int, const Point&)>;
            using cfunc_old = std::function<const T(const int, const Point&)>;
            parameter() :m_func{ [=](const int, const int, const Point&) {return T(); } } {};
            parameter(const cfunc& func):m_func{func}{}
            parameter(const cfunc_old& func)
            {
                cfunc f = [=](const int, const int n, const Point& p) {return func(n, p); };
                m_func = f;
            }
            parameter(const double _p) :m_func{ [=](const int, const int, const Point&) {return _p; } } {}
            parameter(const Mesh::Point _p) :m_func{ [=](const int, const int, const Point&) {return _p; } } {}
            parameter(const parameter<T>& _p) :m_func{ _p.m_func } {}
            ~parameter() {};
            const T         get(const Point& p) const { return m_func(0, 0, p); };
            const T         get(const int number, const Point& p) const { return m_func(0, number, p); };
            const T         get(const int element, const int node, const Point& p) const { return m_func(element, node, p); };
            void            set(const cfunc& func) { m_func = func; };
        private:
            cfunc           m_func;
        };

        template<class T>
        class point_source
        {
        public:
            point_source() : m_point(Mesh::Point(0,0,0)), m_value(T(0)) {};
            point_source(const Mesh::Point& p, const T& val) : m_point(p), m_value(val) {};
```

```
            const T             get_value() const { return m_value; };
            const Mesh::Point   get_point() const { return m_point; };
            point_source<T>&    operator=(const point_source<T>& ps)
            {
                m_point = ps.m_point;
                m_value = ps.m_value;
                return *this;
            }
        private:
            Mesh::Point     m_point;
            T               m_value;
        };
        class CParameter
        {
        public:
            CParameter();
            //CParameter(const double _diff, const double _adv, const double _mass);
            CParameter(const parameter<double>& _diff, const parameter<double>& _adv, const parameter<double>& _mass);
            CParameter(const Parameters&, const parameter<double>&);
            ~CParameter();
            const double        GetDiffusion() const;
            const double        GetAdvection() const;
            const double        GetMass() const;
            const double        GetDiffusion(const Point&) const;
            const double        GetAdvection(const Point&) const;
            const double        GetMass(const Point&) const;
        private:
            parameter<double>  m_diffusion;
            parameter<double>  m_advection;
            parameter<double>  m_mass;
        };
    }
}

#endif // !CORENC_MESH_PARAMETER_H_
```

7.95 CoreNCFEM/Point.cpp File Reference

```
#include "Point.h"
#include <cmath>
```

7.96 CoreNCFEM/Point.h File Reference

```
#include <cmath>
#include <vector>
```

Classes

- class corenc::Mesh::Point
- struct corenc::Mesh::GaussTriangle
- struct corenc::Mesh::GaussRectangular
- struct corenc::Mesh::Gauss1dim
- struct corenc::Mesh::Gauss1dimN< N >
- struct corenc::Mesh::GaussTetrahedron
- struct corenc::Mesh::GaussRectangularCubic

Namespaces

- namespace corenc
- namespace corenc::Mesh

Macros

- #define CORENC_MESH_Point_h

Enumerations

- enum class corenc::Terms {
  corenc::IUV , corenc::IDUDV , corenc::IDUV , corenc::IUDV ,
  corenc::EUV , corenc::EDUDV , corenc::EDUV , corenc::EUDV ,
  corenc::EFV , corenc::RUV , corenc::SUPG }
- enum class corenc::Parameters { corenc::DIFFUSION , corenc::MASS , corenc::ADVECTION }

7.96.1 Macro Definition Documentation

7.96.1.1 CORENC_MESH_Point_h

```
#define CORENC_MESH_Point_h
```

7.97 Point.h

Go to the documentation of this file.

```
#pragma once
#ifndef CORENC_MESH_Point_h
#define CORENC_MESH_Point_h
#include <cmath>
#include <vector>
namespace corenc
{
    enum class Terms
    {
        // left-side
        // uv
        IUV,
        // grad u grad v
        IDUDV,
        // grad u v
        IDUV,
        // u grad v
        IUDV,
        // right-side
        EUV,
        EDUDV,
        EDUV,
        EUDV,
        EFV,
        // right-side matrix
        RUV,
        SUPG,
    };

    enum class Parameters
    {
        DIFFUSION,
        MASS,
        ADVECTION
    };

    namespace Mesh
    {
        class Point
        {
        public:
            Point() :x{ 0 }, y{ 0 }, z{ 0 } {}
```

```
        Point(const double _x, const double _y) :
            x{ _x }, y{ _y }, z{ 0 } {}
        Point(const double _x, const double _y, const double _z) :
            x{ _x }, y{ _y }, z{ _z } {}
        Point(const Point& p) :
            x{ p.x }, y{ p.y }, z{ p.z } {}
        double x, y, z;
        const double Jacobian() const { return 1; }
        Point& operator=(const Point& p)
        {
            x = p.x;
            y = p.y;
            z = p.z;
            return *this;
        }
        const bool operator==(const Point& p)
        {
            const double eps{ 1e-13 };
            if (fabs(x - p.x) < eps)
                if (fabs(y - p.y) < eps)
                    if (fabs(z - p.z) < eps)
                        return true;
            return false;
        }
        friend const bool operator!=(const Point& p1, const Point& p2)
        {
            const double eps{ 1e-13 };
            if (fabs(p1.x - p2.x) < eps)
                if (fabs(p1.y - p2.y) < eps)
                    if (fabs(p1.z - p2.z) < eps)
                        return false;
            return true;
        }
        const bool operator<(const Point& p2)
        {
            return (x < p2.x);
        }
        friend const double operator*(const Point& lhs, const Point& rhs)
        {
            return lhs.x * rhs.x + lhs.y * rhs.y + lhs.z * rhs.z;
        }
        const Point operator*(const double rhs)
        {
            return Point{ x * rhs, y * rhs, z * rhs };
        }
        Point& operator+=(const Point& rhs)
        {
            x += rhs.x;
            y += rhs.y;
            z += rhs.z;
            return *this;
        }
        Point& operator*=(const double rhs)
        {
            x *= rhs;
            y *= rhs;
            z *= rhs;
             return *this;
         }
         friend const Point operator*(const Point& lhs, const double rhs)
         {
             return Point{ rhs * lhs.x, rhs * lhs.y, rhs * lhs.z };
         }
         friend const Point operator*(const double lhs, const Point& rhs)
         {
             return Point{ lhs * rhs.x, lhs * rhs.y, lhs * rhs.z };
         }
         friend const Point operator+(const Point& lhs, const Point& rhs)
         {
             return Point{ lhs.x + rhs.x, lhs.y + rhs.y, lhs.z + rhs.z };
         }
         friend const Point operator-(const Point& lhs, const Point& rhs)
         {
             return Point{ lhs.x - rhs.x, lhs.y - rhs.y, lhs.z - rhs.z };
         }
    };

    struct GaussTriangle
    {
        const static double m_tra[];
        const static double m_trb[];
        const static double m_sqrt15;
        const static double m_trw[];
        const static int    m_order;

    };
    struct GaussRectangular
```

```
        {
            const static double m_ra[];
            const static double m_rb[];
            const static double m_rw[];
            const static double m_a;
            const static double m_b;
            const static double m_c;
            const static double m_wa;
            const static double m_wb;
            const static double m_wc;
        };
        struct Gauss1dim
        {
            const static int    m_order;
            const static double m_a[];
            const static double m_sqrt35;
            const static double m_w[];
        };

        template<int N>
        struct Gauss1dimN
        {
            const static int    m_order;
            const static double m_a[];
            const static double m_w[];
        };


        struct GaussTetrahedron
        {
            const    static double    m_la[];
            const    static double    m_lb[];
            const    static double    m_lc[];
            const    static double    m_ld[];
            const    static double    m_w[];
            const    static double    m_psq, m_msq;
        };

        struct GaussRectangularCubic
        {
            const static double m_ra[];
            const static double m_rb[];
            const static double m_rc[];
            const static double m_rw[];
            const static double m_a;
            const static double m_b;
            const static double m_c;
            const static double m_w1;
            const static double m_w2;
            const static double m_w3;
            const static double m_w4;
            const static int m_s{ 34 };
        };
    }
}
#endif /* CORENC_MESH_Point_h */
```

7.98 main.cpp File Reference

```
#include <iostream>
#include "colors.h"
#include "Tests/test_cases.h"
```

Functions

- int main (int argc, char *argv[ ])

7.98.1 Function Documentation

7.98.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

7.99 Problems/BurgersScalar.cpp File Reference

```
#include "BurgersScalar.h"
#include <vector>
```

7.100 Problems/BurgersScalar.h File Reference

```
#include "Problems.h"
#include <vector>
```

Classes

- class corenc::CBurgersScalar

Namespaces

- namespace corenc

7.101 BurgersScalar.h

Go to the documentation of this file.

```
#ifndef CORENC_PROBLEMS_BURGERS_H_
#define CORENC_PROBLEMS_BURGERS_H_

#include "Problems.h"
#include <vector>
namespace corenc
{
    class CBurgersScalar : public CProblem
    {
     public:
         CBurgersScalar();
         ~CBurgersScalar();
         Terms                       getTerm(const unsigned int) const;
         const unsigned int           getNumberOfTerms() const;
         const int                    setTerm(const unsigned int, const Terms&);
         const int                    addTerm(const Terms&);
         const double                 getFlux(const double) const;
         const int                    removeTerm(const Terms&);
         const int                    load_parameters(const std::string& file_name);
     private:
         std::vector<Terms>           m_terms;
     };
 }
 #endif // !CORENC_PROBLEMS_BURGERS_H_
```

7.102 Problems/DiffusionScalar.cpp File Reference

```
#include "DiffusionScalar.h"
#include <vector>
#include <istream>
#include <iostream>
#include <fstream>
```

7.103 Problems/DiffusionScalar.h File Reference

```
#include "Problems.h"
#include <vector>
#include "../CoreNCFEM/Parameter.h"
#include <map>
#include <tuple>
```

Classes

- class corenc::CDiffusionScalar

Namespaces

- namespace corenc

7.104 DiffusionScalar.h

Go to the documentation of this file.

```
#ifndef CORENC_PROBLEMS_DIFFUSIONSCALAR_H_
#define CORENC_PROBLEMS_DIFFUSIONSCALAR_H_

#include "Problems.h"
#include <vector>
#include "../CoreNCFEM/Parameter.h"
#include <map>
#include <tuple>
namespace corenc
{
    class CDiffusionScalar : public CProblem
    {
        using boundary = std::tuple<int, Mesh::parameter<double>, Mesh::parameter<double>>;
    public:
        CDiffusionScalar();
        ~CDiffusionScalar();
        Terms                              getTerm(const unsigned int) const;
        const unsigned int                  getNumberOfTerms() const;
        const int                           findTerm(const Terms&) const;
        const int                           setTerm(const unsigned int, const Terms&);
        const int                           addTerm(const Terms&);
        const int                           removeTerm(const Terms&);
        const int                           load_parameters(const std::string& file_name);
        const double                        get_parameter(const Terms&, const int element_type, const Mesh::Point&) const;
        const double                        get_parameter(const Terms&, const int element_number, const int element_type, const Mesh::Point&) const;
        const Mesh::Point                    get_parameter(const Terms&, const int element_number, const int element_type, const Mesh::Point&, const int) const;
        const double                        get_parameter(const Terms&, const int element_type, const int element_number, const int node, const Mesh::Point&) const;
```

```
        const Mesh::Point                       get_parameter(const Terms&, const int element_type, const int element_number, const int node, const Mesh::Point&, const int v) const;
        const double                            get_boundary_parameter(const int type, const int element_type, const Mesh::Point&) const;
        const double                            get_boundary_parameter(const int type, const int element_type, const int element_number, const Mesh::Point&) const;
        const double                            get_boundary_parameter(const int type, const int element_type, const int element_number, const int node, const Mesh::Point&) const;
        const int                               get_number_of_boundaries() const;
        const int                               get_boundary_type(const int number) const;
        const int                               add_parameter(const Terms&, const int element_type, const double& value);
        const int                               add_parameter(const Terms&, const int element_type, const Mesh::parameter<double>& value);
        const int                               add_parameter(const Terms&, const int element_type, const Mesh::parameter<Mesh::Point>& value);
        const int                               set_parameter(const Terms&, const int element_type, const Mesh::parameter<double>& value);
        const int                               set_parameter(const Terms&, const int element_type, const Mesh::parameter<Mesh::Point>& value);
        const int                               set_boundary_parameter(const int type, const int element_type, const boundary& value);
                                                // 1st and 2nd types of boundary conditions
        const int                               add_boundary_parameter(const int type, const int element_type, const Mesh::parameter<double>& value);
                                                // 3rd type of boundary conditions
        const int                               add_boundary_parameter(const int element_type, const Mesh::parameter<double>& value, const Mesh::parameter<double>& value2);
        const Mesh::point_source<double>        get_point_source(const int number) const;
        void                                    set_point_source(const int number, const Mesh::point_source<double>&);
        const int                               get_total_sources() const;
    private:
        std::vector<Terms>                        m_terms;
        std::map<int, Mesh::parameter<double»      m_params;
        std::map<int, Mesh::parameter<Mesh::Point» m_vels;
        std::map<int, Mesh::parameter<double»      m_srcs;
        std::map<int, Mesh::parameter<double»      m_gams;
        std::map<int, boundary>                    m_bounds;
        //std::map<int, Mesh::point_source<double»   m_pointsrcs;
        std::vector<Mesh::point_source<double»     m_pointsrcs;
        int                                        m_total_params;
        int                                        m_total_srcs;
        int                                        m_total_gams;
        int                                        m_total_bounds;
    };
}
#endif // !CORENC_PROBLEMS_DIFFUSIONSCALAR_H_
```

7.105 Problems/Problems.h File Reference

```
#include "../CoreNCFEM/Point.h"
#include <string>
```

Classes

- class corenc::CProblem

Namespaces

- namespace corenc

Macros

- #define CORENC_PROBLEMS_PROBLEMS_H_

7.105.1 Macro Definition Documentation

7.105.1.1 CORENC_PROBLEMS_PROBLEMS_H_

```
#define CORENC_PROBLEMS_PROBLEMS_H_
```

7.106 Problems.h

Go to the documentation of this file.

```
#pragma once
#ifndef CORENC_PROBLEMS_PROBLEMS_H_
#define CORENC_PROBLEMS_PROBLEMS_H_
#include "../CoreNCFEM/Point.h"
#include <string>

namespace corenc
{
    class CProblem
    {
    public:
        CProblem() {}
        virtual ~CProblem() {};
        virtual Terms                      getTerm(const unsigned int) const = 0;
        virtual const unsigned int          getNumberOfTerms() const = 0;
        virtual const int                   setTerm(const unsigned int, const Terms&) = 0;
        virtual const int                   addTerm(const Terms&) = 0;
        virtual const int                   load_parameters(const std::string& file_name) = 0;
    };
}

#endif // !CORENC_PROBLEMS_PROBLEMS_H_
```

7.107 Problems/ShallowWater.cpp File Reference

```
#include "ShallowWater.h"
#include <vector>
#include <istream>
#include <iostream>
#include <fstream>
```

7.108 Problems/ShallowWater.h File Reference

```
#include "Problems.h"
#include <vector>
#include "../CoreNCFEM/Parameter.h"
#include <map>
#include <tuple>
```

Classes

- class corenc::CShallowWater

Namespaces

- namespace corenc

7.109 ShallowWater.h

Go to the documentation of this file.

```
#ifndef CORENC_PROBLEMS_SHALLOWWATER_H_
#define CORENC_PROBLEMS_SHALLOWWATER_H_

#include "Problems.h"
#include <vector>
#include "../CoreNCFEM/Parameter.h"
#include <map>
#include <tuple>
namespace corenc
{
    class CShallowWater : public CProblem
    {
        using boundary = std::tuple<int, Mesh::parameter<double>, Mesh::parameter<double>>;
    public:
        CShallowWater();
        ~CShallowWater();
        Terms                           getTerm(const unsigned int) const;
        const unsigned int              getNumberOfTerms() const;
        const int                       setTerm(const unsigned int, const Terms&);
        const int                       addTerm(const Terms&);
        const int                       removeTerm(const Terms&);
        const int                       load_parameters(const std::string& file_name);
        const double                    get_parameter(const Terms&, const int element_type, const Mesh::Point&) const;
        const double                    get_parameter(const Terms&, const int element_number, const int element_type, const Mesh::Point&) const;
        const double                    get_boundary_parameter(const int type, const int element_type, const Mesh::Point&) const;
        const double                    get_boundary_parameter(const int type, const int element_number, const int element_type, const Mesh::Point&) const;
        const int                       get_number_of_boundaries() const;
        const double                    get_solution(const int sys_number, const int element_type, const int element_number, const Mesh::Point&) const;
        const int                       get_boundary_type(const int number) const;
        const int                       add_parameter(const Terms&, const int element_type, const Mesh::parameter<double>& value);
        const int                       set_parameter(const Terms&, const int element_type, const Mesh::parameter<double>& value);
        const int                       set_boundary_parameter(const int type, const int element_type, const boundary& value);
        // 1st and 2nd types of boundary conditions
        const int                       add_boundary_parameter(const int type, const int element_type, const Mesh::parameter<double>& value);
        // 3rd type of boundary conditions
        const int                       add_boundary_parameter(const int element_type, const Mesh::parameter<double>& value, const Mesh::parameter<double>& value2);
    private:
        std::vector<Terms>                       m_terms;
        std::map<int, Mesh::parameter<double>»  m_params;
        std::map<int, Mesh::parameter<double>»  m_srcs;
        std::map<int, boundary>                  m_bounds;
        int                                      m_total_params;
        int                                      m_total_srcs;
        int                                      m_total_bounds;
    };
}
#endif // !CORENC_PROBLEMS_SHALLOWWATER_H_
```

7.110 Solvers/dg_solver.h File Reference

```
#include "../CoreNCFEM/Grids/TriangularMesh.h"
#include "../CoreNCFEM/Methods/DGMethod.h"
#include "../Problems/DiffusionScalar.h"
#include "../CoreNCA/MatrixSkyline.h"
#include "../CoreNCFEM/Methods/FEAnalysis.h"
```

Classes

- class corenc::solvers::dg_solver< _Problem, _Mesh, _Result >

Namespaces

- namespace corenc
- namespace corenc::solvers

7.111 dg_solver.h

Go to the documentation of this file.

```
#ifndef CORENC_SOLVERS_DG_SOLVER_H_
#define CORENC_SOLVERS_DG_SOLVER_H_

#include "../CoreNCFEM/Grids/TriangularMesh.h"
#include "../CoreNCFEM/Methods/DGMethod.h"
#include "../Problems/DiffusionScalar.h"
#include "../CoreNCA/MatrixSkyline.h"
#include "../CoreNCFEM/Methods/FEAnalysis.h"

namespace corenc
{
    namespace solvers
    {
        template<class _Problem, class _Mesh, class _Result>
        class dg_solver
        {
            using _Method = method::DGMethod<_Problem, _Mesh, Algebra::MatrixSkyline>;
        public:
            dg_solver() :m_method{ nullptr } {}
            ~dg_solver()
            {
                if (m_method != nullptr)
                    delete m_method;
            }
            // terms, method, mesh, solver, result
            const int               elliptic_solver(_Problem*, _Mesh*, _Result*);
            const double            get_value(const _Mesh&, const _Result&, const Mesh::Point& p) const;
            const double            get_value(const _Method*, const _Mesh&, const _Result&, const Mesh::Point& p) const;
            const double            get_value(const _Mesh&, const _Result&, const Mesh::Point& p, const int i) const;
            const Mesh::Point        get_gradvalue(const _Mesh&, const _Result&, const Mesh::Point& p) const;
            const Mesh::Point        get_gradvalue(const _Mesh&, const _Result&, const Mesh::Point& p, const int i) const;
        private:
            _Method * m_method;
        };

        template<class _Problem, class _Mesh, class _Result>
        const int dg_solver<_Problem, _Mesh, _Result>::elliptic_solver(_Problem* problem, _Mesh* mesh, _Result* result)
        {
            std::vector<double> res;
            //std::shared_ptr<Algebra::MatrixSkyline> matrix{ new Algebra::MatrixSkyline() };
            Algebra::MatrixSkyline* matrix{ new Algebra::MatrixSkyline() };
            std::vector<double> rhs;
            if (m_method != nullptr)
                delete m_method;
            m_method = new _Method{ problem, mesh, matrix, &rhs };
            m_method->Discretization();
            Algebra::ESolver esl{ Algebra::Solvers::GMRES };
            *result = esl.Solve(*matrix, rhs, *result, res, 100000, 1e-13);
            delete matrix;
            return 0;
        }

        template<class _Problem, class _Mesh, class _Result>
        const double dg_solver<_Problem, _Mesh, _Result>::get_value(const _Mesh& mesh, const _Result& res, const Mesh::Point& p) const
        {
            if (m_method != nullptr)
                return m_method->GetSolution(mesh, res, p);
            return 0.;
```

```
        }

        template<class _Problem, class _Mesh, class _Result>
        const Mesh::Point dg_solver<_Problem, _Mesh, _Result>::get_gradvalue(const _Mesh& mesh, const _Result& res, const Mesh::Point& p) const
        {
            if (m_method != nullptr)
                return m_method->GetGradSolution(mesh, res, p);
            return Mesh::Point(0, 0, 0);
        }

        template<class _Problem, class _Mesh, class _Result>
        const double dg_solver<_Problem, _Mesh, _Result>::get_value(const _Method* method2, const _Mesh& mesh, const _Result& res, const Mesh::Point& p) const
        {
            if (method2 != nullptr)
                return method2->GetSolution(mesh, res, p);
            return 0.;
        }

        template<class _Problem, class _Mesh, class _Result>
        const double dg_solver<_Problem, _Mesh, _Result>::get_value(const _Mesh& mesh, const _Result& res, const Mesh::Point& p, const int i) const
        {
            if (m_method != nullptr)
                return m_method->GetSolution(mesh, res, p, i);
            return 0.;
        }
        template<class _Problem, class _Mesh, class _Result>
        const Mesh::Point dg_solver<_Problem, _Mesh, _Result>::get_gradvalue(const _Mesh& mesh, const _Result& res, const Mesh::Point& p, const int i) const
        {
            if (m_method != nullptr)
                return m_method->GetGradSolution(mesh, res, p, i);
            return Mesh::Point(0, 0, 0);
        }

    }
}
#endif // !CORENC_SOLVERS_dg_solver_H_
```

7.112 Solvers/dg_solver_shallow_water.cpp File Reference

```
#include "dg_solver_shallow_water.h"
#include <vector>
#include "../CoreNCFEM/Grids/RegularMesh.h"
#include "../CoreNCFEM/Parameter.h"
#include <algorithm>
#include <functional>
```

7.113 Solvers/dg_solver_shallow_water.h File Reference

```
#include <vector>
#include <functional>
#include <istream>
#include <iostream>
#include <fstream>
#include <algorithm>
#include "../CoreNCFEM/Point.h"
```

Classes

- struct corenc::solvers::vector_solution
- class corenc::solvers::dg_solver_shallow_water
- class corenc::solvers::dg_shallow_water< Mesh >

Namespaces

- namespace corenc
- namespace corenc::solvers

7.114 dg_solver_shallow_water.h

Go to the documentation of this file.

```
#ifndef CORENC_SOLVERS_DG_SOLVER_SHALLOW_WATER_H_
#define CORENC_SOLVERS_DG_SOLVER_SHALLOW_WATER_H_

#include <vector>
#include <functional>
#include <istream>
#include <iostream>
#include <fstream>
#include <algorithm>
#include "../CoreNCFEM/Point.h"
namespace corenc
{
    namespace solvers
    {
        struct vector_solution
        {
            std::vector<double> S[3];
            vector_solution() {}
            vector_solution(const int _size)
            {
                S[0].resize(_size);
                S[1].resize(_size);
                S[2].resize(_size);
            }
        };
        class dg_solver_shallow_water
        {
        public:
            dg_solver_shallow_water();
            ~dg_solver_shallow_water();
            const int               solve() const;
            const int               solve(
                const double t0,
                const double t1,
                const size_t nx,
                const size_t ny,
                const double x0,
                const double x1,
                const double y0,
                const double y1,
                const double g,
                const double H,
                const std::function<const std::vector<double>(const std::vector<double>&)>&,
                const std::function<const std::vector<double>(const std::vector<double>&)>&,
                const std::function<const std::vector<double>(const std::vector<double>&)>&) const;
        };

        template<class Mesh>
        class dg_shallow_water
        {
        public:
            dg_shallow_water();
            ~dg_shallow_water();
            const int               solve(
                const double t0,
                const double t1,
                const Mesh& mesh,
                vector_solution& sol,
                const std::function<const std::vector<double>(const std::vector<double>&)>&,
                const std::function<const std::vector<double>(const std::vector<double>&)>&,
                const std::function<const std::vector<double>(const std::vector<double>&)>&) const;
            const int               solve(
                const double t0,
                const double t1,
                const Mesh& mesh,
                vector_solution& sol,
                std::vector<double>& bath,
                std::vector<double>& ze,
                std::vector<double>& dzx,
                std::vector<double>& dzy,
                std::vector<double>& dbx,
                std::vector<double>& dby,
```

```
        const std::function<const std::vector<double>(const std::vector<double>&, const int)>&,
        const std::function<const std::vector<double>(const std::vector<double>&, const int)>&,
        const std::function<const std::vector<double>(const std::vector<double>&, const int)>&,
        const bool WRITE_FILE) const;
};

template<class Mesh>
dg_shallow_water<Mesh>::dg_shallow_water()
{

}
template<class Mesh>
dg_shallow_water<Mesh>::~dg_shallow_water()
{

}

template<class Mesh>
const int dg_shallow_water<Mesh>::solve(
    const double t0,
    const double t1,
    const Mesh& mesh,
    vector_solution& sol,
    const std::function < const std::vector<double>(const std::vector<double>&)>&R,
    const std::function < const std::vector<double>(const std::vector<double>&)>&G,
    const std::function < const std::vector<double>(const std::vector<double>&)>&F) const
{
     std::vector<double> Ut[3];
     const int max_iter = 30000;
     const double dx = mesh.GetNode(mesh.GetNumberOfNodes() - 1).x - mesh.GetNode(0).x;
     const double dy = mesh.GetNode(mesh.GetNumberOfNodes() - 1).y - mesh.GetNode(0).y;
     //const double dx = (x1 - x0) / nx;
     //const double dy = (y1 - y0) / ny;
     const int size = mesh.GetNumberOfElements();
     const int bsize = mesh.GetNumberOfBoundaries();

     std::vector<vector_solution> U(2);
     std::vector<vector_solution> W(2);
     U[0].S[0].resize(size);
     U[0].S[1].resize(size);
     U[0].S[2].resize(size);
     U[1].S[0].resize(size);
     U[1].S[1].resize(size);
     U[1].S[2].resize(size);

     W[0].S[0].resize(size);
     W[0].S[1].resize(size);
     W[0].S[2].resize(size);
     W[1].S[0].resize(size);
     W[1].S[1].resize(size);
     W[1].S[2].resize(size);
     for (size_t i = 0; i < size; ++i)
     {
         W[0].S[0][i] = sol.S[0][i];
         W[0].S[1][i] = sol.S[1][i];
         W[0].S[2][i] = sol.S[2][i];

         U[0].S[0][i] = sol.S[0][i];
         U[0].S[1][i] = sol.S[1][i] / sol.S[0][i];
         U[0].S[2][i] = sol.S[2][i] / sol.S[0][i];
     }

     double t_step = 0.1;
     const double cfl = 0.5;
     // W = [h hu hv]
     double lambda_x = 0;
     double lambda_y = 0;
     double lambdax = 0;
     double lambday = 0;
     double lambda = 0;
     double t_curr = 0;
     double g = 1;
     size_t iter_max = 10000;
     for (size_t t = 0; t < iter_max && t_curr < t1; ++t, t_curr += t_step)
     {
         lambda_x = 0;
         lambda_y = 0;
         for (size_t i = 0; i < size; ++i)
         {
             const auto& elem = mesh.GetElement(i);
             const auto& res = F(std::vector<double>{W[t].S[0][i], W[t].S[1][i], W[t].S[2][i]});
             W[t + 1].S[0][i] = W[t].S[0][i] + res[0];
             W[t + 1].S[1][i] = W[t].S[1][i] + res[1];
             W[t + 1].S[2][i] = W[t].S[2][i] + res[2];

             lambda_x = std::max(fabs(U[t].S[1][i]), lambda_x);
             lambda_y = std::max(fabs(U[t].S[2][i]), lambda_y);
```

```
                }
                t_step = cfl / 2 * std::min(dx / lambda_x, dy / lambda_y);
                if (t_curr + t_step > t1)
                    t_step = t1 - t_curr;
                for (size_t i = 0; i < bsize; ++i)
                {
                    const auto& bound = mesh.GetBoundary(i);
                    const int nk = bound->GetNeighbour(0);
                    const int ne = bound->GetNeighbour(1);
                    const auto& normal = bound->GetNormal();
                    if (ne > -1)
                    {
                        const auto& normal = bound->GetNormal();
                        std::vector<double> wk(3);
                        wk[0] = U[t].S[0][nk];
                        wk[1] = U[t].S[1][nk] * U[t].S[0][nk];
                        wk[2] = U[t].S[2][nk] * U[t].S[0][nk];

                        std::vector<double> we(3);
                        we[0] = U[t].S[0][ne];
                        we[1] = U[t].S[1][ne] * U[t].S[0][ne];
                        we[2] = U[t].S[2][ne] * U[t].S[0][ne];

                        //lambda_x = std::max(fabs(U[t].S[1][nk]) + sqrt(g * U[t].S[0][nk]), fabs(U[t].S[1][ne]) + sqrt(g * U[t].S[0][ne]));
                        //lambda_y = std::max(fabs(U[t].S[2][nk]) + sqrt(g * U[t].S[0][nk]), fabs(U[t].S[2][ne]) + sqrt(g * U[t].S[0][ne]));

                        lambda_x = std::max(fabs(U[t].S[1][nk]), fabs(U[t].S[1][ne]));
                        lambda_y = std::max(fabs(U[t].S[2][nk]), fabs(U[t].S[2][ne]));


                        lambdax = std::max(lambdax, lambda_x);
                        lambday = std::max(lambday, lambda_y);
                        double ll = std::max(lambda_x, lambda_y);
                        //cout << "max:\t" << ll << endl;
                        std::vector<double> uk(3);
                        uk[0] = U[t].S[0][nk];
                        uk[1] = U[t].S[1][nk];
                        uk[2] = U[t].S[2][nk];

                        std::vector<double> ue(3);
                        ue[0] = U[t].S[0][ne];
                        ue[1] = U[t].S[1][ne];
                        ue[2] = U[t].S[2][ne];

                        const auto rk = R(uk);
                        const auto re = R(ue);
                        const auto gk = G(uk);
                        const auto ge = G(ue);

                        std::vector<double> uu(3);
                        uu[0] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[0] + re[0]) / 2 + normal.y * (gk[0] + ge[0]) / 2 - (lambda_x * normal.x / 2 * (ue[0] - uk[0]) + lambda_y * normal.y / 2 * (ue[0] - uk[0])));
                        uu[1] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[1] + re[1]) / 2 + normal.y * (gk[1] + ge[1]) / 2 - (lambda_x * normal.x / 2 * (ue[1] - uk[1]) + lambda_y * normal.y / 2 * (ue[1] - uk[1])));
                        uu[2] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[2] + re[2]) / 2 + normal.y * (gk[2] + ge[2]) / 2 - (lambda_x * normal.x / 2 * (ue[2] - uk[2]) + lambda_y * normal.y / 2 * (ue[2] - uk[2])));
                        W[t + 1].S[0][nk] -= uu[0];
                        W[t + 1].S[1][nk] -= uu[1];
                        W[t + 1].S[2][nk] -= uu[2];

                        uu[0] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (-normal.x * (rk[0] + re[0]) / 2 - normal.y * (gk[0] + ge[0]) / 2 + (lambda_x * normal.x / 2 * (ue[0] - uk[0]) + lambda_y * normal.y / 2 * (ue[0] - uk[0])));
                        uu[1] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (-normal.x * (rk[1] + re[1]) / 2 - normal.y * (gk[1] + ge[1]) / 2 + (lambda_x * normal.x / 2 * (ue[1] - uk[1]) + lambda_y * normal.y / 2 * (ue[1] - uk[1])));
                        uu[2] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (-normal.x * (rk[2] + re[2]) / 2 - normal.y * (gk[2] + ge[2]) / 2 + (lambda_x * normal.x / 2 * (ue[2] - uk[2]) + lambda_y * normal.y / 2 * (ue[2] - uk[2])));
                        W[t + 1].S[0][ne] -= uu[0];
                        W[t + 1].S[1][ne] -= uu[1];
                        W[t + 1].S[2][ne] -= uu[2];
                    }
                }
                for (size_t i = 0; i < bsize; ++i)
                {
                    const auto& bound = mesh.GetBoundary(i);
                    const int nk = bound->GetNeighbour(0);
                    const int ne = bound->GetNeighbour(1);
                    if (ne == -1)
                    {
                        auto normal = bound->GetNormal();
```

```

                        std::vector<double> u(3);
                        u[0] = U[t].S[0][nk];
                        u[1] = -U[t].S[1][nk];
                        u[2] = -U[t].S[2][nk];

                        lambdax = std::max(lambdax, lambda_x);
                        lambday = std::max(lambday, lambda_y);

                        std::vector<double> uk(3);
                        uk[0] = U[t].S[0][nk];
                        uk[1] = U[t].S[1][nk];
                        uk[2] = U[t].S[2][nk];

                        std::vector<double> ue(3);
                        ue[0] = u[0];
                        ue[1] = u[1];
                        ue[2] = u[2];

                        const auto rk = R(uk);
                        const auto re = R(ue);
                        const auto gk = G(uk);
                        const auto ge = G(ue);
                        lambda_x = std::max(fabs(uk[1]), fabs(ue[1]));
                        lambda_y = std::max(fabs(uk[2]), fabs(ue[2]));
                        std::vector<double> uu(3);
                        if (normal.x > 0 || normal.y > 0)
                        {
                            uu[0] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[0] + re[0]) / 2 + normal.y * (gk[0] + ge[0]) / 2 - (lambda_x * normal.x / 2 * (ue[0] - uk[0]) + lambda_y * normal.y / 2 * (ue[0] - uk[0])));
                            uu[1] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[1] + re[1]) / 2 + normal.y * (gk[1] + ge[1]) / 2 - (lambda_x * normal.x / 2 * (ue[1] - uk[1]) + lambda_y * normal.y / 2 * (ue[1] - uk[1])));
                            uu[2] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[2] + re[2]) / 2 + normal.y * (gk[2] + ge[2]) / 2 - (lambda_x * normal.x / 2 * (ue[2] - uk[2]) + lambda_y * normal.y / 2 * (ue[2] - uk[2])));
                        }
                        else
                        {
                            uu[0] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[0] + re[0]) / 2 + normal.y * (gk[0] + ge[0]) / 2 + (lambda_x * normal.x / 2 * (ue[0] - uk[0]) + lambda_y * normal.y / 2 * (ue[0] - uk[0])));
                            uu[1] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[1] + re[1]) / 2 + normal.y * (gk[1] + ge[1]) / 2 + (lambda_x * normal.x / 2 * (ue[1] - uk[1]) + lambda_y * normal.y / 2 * (ue[1] - uk[1])));
                            uu[2] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[2] + re[2]) / 2 + normal.y * (gk[2] + ge[2]) / 2 + (lambda_x * normal.x / 2 * (ue[2] - uk[2]) + lambda_y * normal.y / 2 * (ue[2] - uk[2])));
                        }

                        W[t + 1].S[0][nk] -= uu[0];
                        W[t + 1].S[1][nk] -= uu[1];
                        W[t + 1].S[2][nk] -= uu[2];
                    }
                }
                for (size_t i = 0; i < size; ++i)
                {

                    U[t + 1].S[0][i] = W[t + 1].S[0][i];
                    U[t + 1].S[1][i] = W[t + 1].S[1][i] / W[t + 1].S[0][i];
                    U[t + 1].S[2][i] = W[t + 1].S[2][i] / W[t + 1].S[0][i];
                }
                W.push_back(vector_solution(size));
                U.push_back(vector_solution(size));

            }
            const auto ut = W.size() - 2;
            for (size_t i = 0; i < size; ++i)
            {
                sol.S[0][i] = W[ut].S[0][i];
                sol.S[1][i] = W[ut].S[1][i];
                sol.S[2][i] = W[ut].S[2][i];
            }
            return 0;
        }

        template<class Mesh>
        const int dg_shallow_water<Mesh>::solve(
            const double t0,
            const double t1,
            const Mesh& mesh,
            vector_solution& sol,
            std::vector<double>& bath,
            std::vector<double>& ze,
            std::vector<double>& dzx,
            std::vector<double>& dzy,
```

```cpp
            std::vector<double>& dbx,
            std::vector<double>& dby,
            const std::function < const std::vector<double>(const std::vector<double>&, const int)>&R,
            const std::function < const std::vector<double>(const std::vector<double>&, const int)>&G,
            const std::function < const std::vector<double>(const std::vector<double>&, const int)>&F,
            const bool WRITE_FILE) const
        {
            std::vector<double> Ut[3];
            const int max_iter = 30000;
            double dx = 100, dy = 100;
            //const double dx = mesh.GetNode(mesh.GetNumberOfNodes() - 1).x - mesh.GetNode(0).x;
            //const double dy = mesh.GetNode(mesh.GetNumberOfNodes() - 1).y - mesh.GetNode(0).y;
            //const double dx = (x1 - x0) / nx;
            //const double dy = (y1 - y0) / ny;
            const int size = mesh.GetNumberOfElements();
            const int bsize = mesh.GetNumberOfBoundaries();

            std::vector<vector_solution> U(2);
            std::vector<vector_solution> W(2);
            U[0].S[0].resize(size);
            U[0].S[1].resize(size);
            U[0].S[2].resize(size);
            U[1].S[0].resize(size);
            U[1].S[1].resize(size);
            U[1].S[2].resize(size);

            W[0].S[0].resize(size);
            W[0].S[1].resize(size);
            W[0].S[2].resize(size);
            W[1].S[0].resize(size);
            W[1].S[1].resize(size);
            W[1].S[2].resize(size);
            for (size_t i = 0; i < size; ++i)
            {
                W[0].S[0][i] = sol.S[0][i];
                W[0].S[1][i] = sol.S[1][i];
                W[0].S[2][i] = sol.S[2][i];

                U[0].S[0][i] = sol.S[0][i];
                U[0].S[1][i] = sol.S[1][i] / sol.S[0][i];
                U[0].S[2][i] = sol.S[2][i] / sol.S[0][i];
            }
            auto center = [=](const size_t i)
            {
                const auto& elem = mesh.GetElement(i);
                std::vector<corenc::Mesh::Point> pts(4);
                pts[0] = mesh.GetNode(elem->GetNode(0));
                pts[1] = mesh.GetNode(elem->GetNode(1));
                pts[2] = mesh.GetNode(elem->GetNode(2));
                pts[3] = mesh.GetNode(elem->GetNode(3));
                return corenc::Mesh::Point(pts[0].x + (pts[3].x - pts[0].x) / 2, pts[0].y + (pts[3].y - pts[0].y) / 2);
            };
            double t_step = 0.1;
            const double cfl = 0.1;
            // W = [h hu hv]
            double lambda_x = 0;
            double lambda_y = 0;
            double lambdax = 0;
            double lambday = 0;
            double lambda = 0;
            double t_curr = 0;
            double g = 1;
            size_t iter_max = 100;
            for (size_t t = 0; t < iter_max && t_curr < t1; ++t, t_curr += t_step)
            {
                lambda_x = 0;
                lambda_y = 0;
                for (size_t i = 0; i < size; ++i)
                {
                    const auto& elem = mesh.GetElement(i);
                    const auto& res = F(std::vector<double>{W[t].S[0][i], W[t].S[1][i], W[t].S[2][i]}, i);
                    W[t + 1].S[0][i] = W[t].S[0][i] + res[0];
                    W[t + 1].S[1][i] = W[t].S[1][i] + res[1];
                    W[t + 1].S[2][i] = W[t].S[2][i] + res[2];

                    lambda_x = std::max(fabs(U[t].S[1][i]) + sqrt(g*U[t].S[0][i]), lambda_x);
                    lambda_y = std::max(fabs(U[t].S[2][i]) + sqrt(g*U[t].S[0][i]), lambda_y);
                    dx = std::min(mesh.GetNode(elem->GetNode(3)).x - mesh.GetNode(elem->GetNode(0)).x, dx);
                    dy = std::min(mesh.GetNode(elem->GetNode(3)).y - mesh.GetNode(elem->GetNode(0)).y, dy);
                    //lambda_x = std::min(U[t].S[1][i])
                    //lambda_x = std::max(fabs(U[t].S[1][i]), lambda_x);
                    //lambda_y = std::max(fabs(U[t].S[2][i]), lambda_y);
                }
```

```
            t_step = cfl / 2 * std::min(dx / lambda_x, dy / lambda_y);
            //std::cout « t_step « std::endl;
            if (t_curr + t_step > t1)
                t_step = t1 - t_curr;
            for (size_t i = 0; i < bsize; ++i)
            {
                const auto& bound = mesh.GetBoundary(i);
                const int nk = bound->GetNeighbour(0);
                const int ne = bound->GetNeighbour(1);
                const auto& normal = bound->GetNormal();
                if (ne > -1)
                {
                    const auto& normal = bound->GetNormal();
                    std::vector<double> wk(3);
                    wk[0] = U[t].S[0][nk];
                    wk[1] = U[t].S[1][nk] * U[t].S[0][nk];
                    wk[2] = U[t].S[2][nk] * U[t].S[0][nk];

                    std::vector<double> we(3);
                    we[0] = U[t].S[0][ne];
                    we[1] = U[t].S[1][ne] * U[t].S[0][ne];
                    we[2] = U[t].S[2][ne] * U[t].S[0][ne];

                    lambda_x = std::max(fabs(U[t].S[1][nk]) + sqrt(g * U[t].S[0][nk]), fabs(U[t].S[1][ne]) + sqrt(g * U[t].S[0][ne]));
                    lambda_y = std::max(fabs(U[t].S[2][nk]) + sqrt(g * U[t].S[0][nk]), fabs(U[t].S[2][ne]) + sqrt(g * U[t].S[0][ne]));

                    //lambda_x = std::max(fabs(U[t].S[1][nk]), fabs(U[t].S[1][ne]));
                    //lambda_y = std::max(fabs(U[t].S[2][nk]), fabs(U[t].S[2][ne]));


                    lambdax = std::max(lambdax, lambda_x);
                    lambday = std::max(lambday, lambda_y);
                    double ll = std::max(lambda_x, lambda_y);
                    //cout « "max:\t" « ll « endl;
                    std::vector<double> uk(3);
                    uk[0] = U[t].S[0][nk];
                    uk[1] = U[t].S[1][nk];
                    uk[2] = U[t].S[2][nk];

                    std::vector<double> ue(3);
                    ue[0] = U[t].S[0][ne];
                    ue[1] = U[t].S[1][ne];
                    ue[2] = U[t].S[2][ne];

                    const auto rk = R(uk, nk);
                    const auto re = R(ue, ne);
                    const auto gk = G(uk, nk);
                    const auto ge = G(ue, ne);

                    std::vector<double> uu(3);
                    uu[0] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[0] + re[0]) / 2 + normal.y * (gk[0] + ge[0]) / 2 - (lambda_x * normal.x / 2 * (ue[0] - uk[0]) + lambda_y * normal.y / 2 * (ue[0] - uk[0])));
                    uu[1] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[1] + re[1]) / 2 + normal.y * (gk[1] + ge[1]) / 2 - (lambda_x * normal.x / 2 * (ue[1] - uk[1]) + lambda_y * normal.y / 2 * (ue[1] - uk[1])));
                    uu[2] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[2] + re[2]) / 2 + normal.y * (gk[2] + ge[2]) / 2 - (lambda_x * normal.x / 2 * (ue[2] - uk[2]) + lambda_y * normal.y / 2 * (ue[2] - uk[2])));
                    W[t + 1].S[0][nk] -= uu[0];
                    W[t + 1].S[1][nk] -= uu[1];
                    W[t + 1].S[2][nk] -= uu[2];

                    uu[0] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (-normal.x * (rk[0] + re[0]) / 2 - normal.y * (gk[0] + ge[0]) / 2 + (lambda_x * normal.x / 2 * (ue[0] - uk[0]) + lambda_y * normal.y / 2 * (ue[0] - uk[0])));
                    uu[1] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (-normal.x * (rk[1] + re[1]) / 2 - normal.y * (gk[1] + ge[1]) / 2 + (lambda_x * normal.x / 2 * (ue[1] - uk[1]) + lambda_y * normal.y / 2 * (ue[1] - uk[1])));
                    uu[2] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (-normal.x * (rk[2] + re[2]) / 2 - normal.y * (gk[2] + ge[2]) / 2 + (lambda_x * normal.x / 2 * (ue[2] - uk[2]) + lambda_y * normal.y / 2 * (ue[2] - uk[2])));
                    W[t + 1].S[0][ne] -= uu[0];
                    W[t + 1].S[1][ne] -= uu[1];
                    W[t + 1].S[2][ne] -= uu[2];


                }
            }
            for (size_t i = 0; i < bsize; ++i)
            {
                const auto& bound = mesh.GetBoundary(i);
                const int nk = bound->GetNeighbour(0);
                const int ne = bound->GetNeighbour(1);
                if (ne == -1)
```

```
                    {
                        auto normal = bound->GetNormal();

                        std::vector<double> u(3);
                        u[0] = U[t].S[0][nk];
                        u[1] = -U[t].S[1][nk];
                        u[2] = -U[t].S[2][nk];

                        //u[0] = U[t].S[0][nk];
                        //u[1] = U[t].S[1][nk];
                        //u[2] = U[t].S[2][nk];

                        lambdax = std::max(lambdax, lambda_x);
                        lambday = std::max(lambday, lambda_y);

                        std::vector<double> uk(3);
                        uk[0] = U[t].S[0][nk];
                        uk[1] = U[t].S[1][nk];
                        uk[2] = U[t].S[2][nk];

                        std::vector<double> ue(3);
                        ue[0] = u[0];
                        ue[1] = u[1];
                        ue[2] = u[2];

                        const auto rk = R(uk, nk);
                        const auto re = R(ue, nk);
                        const auto gk = G(uk, nk);
                        const auto ge = G(ue, nk);
                        lambda_x = std::max(fabs(uk[1]) + sqrt(g*uk[0]), fabs(ue[1]) + sqrt(g*ue[0]));
                        lambda_y = std::max(fabs(uk[2]) + sqrt(g*uk[0]), fabs(ue[2]) + sqrt(g*ue[0]));
                        std::vector<double> uu(3);
                        if (normal.x > 0 || normal.y > 0)
                        {
                            uu[0] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[0] + re[0]) / 2 + normal.y * (gk[0] + ge[0]) / 2 - (lambda_x * normal.x / 2 * (ue[0] - uk[0]) + lambda_y * normal.y / 2 * (ue[0] - uk[0])));
                            uu[1] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[1] + re[1]) / 2 + normal.y * (gk[1] + ge[1]) / 2 - (lambda_x * normal.x / 2 * (ue[1] - uk[1]) + lambda_y * normal.y / 2 * (ue[1] - uk[1])));
                            uu[2] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[2] + re[2]) / 2 + normal.y * (gk[2] + ge[2]) / 2 - (lambda_x * normal.x / 2 * (ue[2] - uk[2]) + lambda_y * normal.y / 2 * (ue[2] - uk[2])));
                        }
                        else
                        {
                            uu[0] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[0] + re[0]) / 2 + normal.y * (gk[0] + ge[0]) / 2 + (lambda_x * normal.x / 2 * (ue[0] - uk[0]) + lambda_y * normal.y / 2 * (ue[0] - uk[0])));
                            uu[1] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[1] + re[1]) / 2 + normal.y * (gk[1] + ge[1]) / 2 + (lambda_x * normal.x / 2 * (ue[1] - uk[1]) + lambda_y * normal.y / 2 * (ue[1] - uk[1])));
                            uu[2] = t_step / mesh.GetElement(nk)->GetMeasure() * bound->GetMeasure() * (normal.x * (rk[2] + re[2]) / 2 + normal.y * (gk[2] + ge[2]) / 2 + (lambda_x * normal.x / 2 * (ue[2] - uk[2]) + lambda_y * normal.y / 2 * (ue[2] - uk[2])));
                        }

                        W[t + 1].S[0][nk] -= uu[0];
                        W[t + 1].S[1][nk] -= uu[1];
                        W[t + 1].S[2][nk] -= uu[2];
                    }
                }
                for (size_t i = 0; i < size; ++i)
                {

                    U[t + 1].S[0][i] = W[t + 1].S[0][i];
                    U[t + 1].S[1][i] = W[t + 1].S[1][i] / W[t + 1].S[0][i];
                    U[t + 1].S[2][i] = W[t + 1].S[2][i] / W[t + 1].S[0][i];
                }
                W.push_back(vector_solution(size));
                U.push_back(vector_solution(size));

                for (size_t k = 0; k < bsize; ++k)
                {
                    const auto& bound = mesh.GetBoundary(k);
                    const auto nk = bound->GetNeighbour(0);
                    const auto ne = bound->GetNeighbour(1);
                    ze[nk] = W[t + 1].S[0][nk] - bath[nk];
                    if (ne > -1)
                    {
                        ze[ne] = W[t + 1].S[0][ne] - bath[ne];
                        const auto ce = center(ne);
                        const auto ck = center(nk);
                        const double cx = ce.x - ck.x;
                        const double cy = ce.y - ck.y;
                        if (fabs(cy) < 1e-13)
                        {
```

```
                            dzx[nk] = (ze[ne] - ze[nk]) / cx;
                            dzx[ne] = dzx[nk];
                            dbx[nk] = (bath[ne] - bath[nk]) / cx;
                            dbx[ne] = dbx[nk];
                        }
                        else
                        {
                            dzy[nk] = (ze[ne] - ze[nk]) / cy;
                            dzy[ne] = dzy[nk];
                            dby[nk] = (bath[ne] - bath[nk]) / cy;
                            dby[ne] = dby[nk];
                        }
                    }
                }
            }
            /*const auto ut = W.size() - 2;
            for (size_t i = 0; i < size; ++i)
            {
                sol.S[0][i] = W[ut].S[0][i];
                sol.S[1][i] = W[ut].S[1][i];
                sol.S[2][i] = W[ut].S[2][i];
            }
            std::ofstream ofs;
            ofs.open("meshU.txt");
            const size_t t_r = U.size() - 1;
            ofs « t_r « std::endl;
            for (size_t i = 0; i < t_r; ++i)
                for (size_t j = 0; j < size; ++j)
                    ofs « U[i].S[0][j] - bath[j] « std::endl;
            ofs.close();*/
            return 0;
        }
    }
}


#endif // !CORENC_SOLVERS_DG_SOLVER_SHALLOW_WATER_H_
```

7.115 Solvers/eigen_solver.h File Reference

```
#include <vector>
#include <complex>
```

Classes

- class corenc::solvers::eigen_solver< Matrix, Solver >

Namespaces

- namespace corenc
- namespace corenc::solvers

7.116 eigen_solver.h

Go to the documentation of this file.

```
#ifndef EIGEN_SOLVER_H
#define EIGEN_SOLVER_H
#include <vector>
#include <complex>
namespace corenc
{
    namespace solvers
    {
        template<class Matrix, class Solver>
```

```
        class eigen_solver
        {
        public:
            eigen_solver(){}
            ~eigen_solver(){}
            void        rayleigh(Matrix* A, Matrix* B, Solver* esl, std::complex<double>* mu0, double* x0, const int n) const
            {
                std::vector<std::complex<double» x(n);
                std::vector<std::complex<double» y(n);
                std::vector<std::complex<double» lam(n);
                double norm_mu = 0;
                double norm_x = 0;
                for (int i = 0; i < n; ++i)
                {
                    norm_mu += std::norm(mu0[i]) * std::norm(mu0[i]);
                    norm_x += std::norm(x0[i]) * std::norm(x0[i]);
                }
                norm_mu = sqrt(norm_mu);
                norm_x = sqrt(norm_x);
                for (int i = 0; i < n; ++i)
                {
                    x[i] = x0[i] / norm_x;
                    y[i] = mu0[i] / norm_mu;
                }
                std::complex<double> temp(0, 0);
                temp =
            }
        };
    }
}
#endif // EIGEN_SOLVER_H
```

7.117 Solvers/fem_solver.h File Reference

```
#include "../CoreNCFEM/Grids/TriangularMesh.h"
#include "../CoreNCFEM/Methods/FEMethod.h"
#include "../Problems/DiffusionScalar.h"
#include "../CoreNCA/MatrixSkyline.h"
#include "../CoreNCFEM/Methods/FEAnalysis.h"
```

Classes

- class corenc::solvers::fem_solver< _Problem, _Mesh, _Result >

Namespaces

- namespace corenc
- namespace corenc::solvers

7.118 fem_solver.h

Go to the documentation of this file.

```
#ifndef CORENC_SOLVERS_FEM_SOLVER_H_
#define CORENC_SOLVERS_FEM_SOLVER_H_

#include "../CoreNCFEM/Grids/TriangularMesh.h"
#include "../CoreNCFEM/Methods/FEMethod.h"
#include "../Problems/DiffusionScalar.h"
#include "../CoreNCA/MatrixSkyline.h"
#include "../CoreNCFEM/Methods/FEAnalysis.h"

// FINITE ELEMENT METHOD SOLVER ONLY IN SPACE
```

```

namespace corenc
{
    namespace solvers
    {
        template<class _Problem, class _Mesh, class _Result>
        class fem_solver
        {
            using _Method = method::FEMethod<_Problem, _Mesh, Algebra::MatrixSkyline>;
            using _Method2 = method::FEMethod<_Problem, _Mesh, Algebra::Matrix>;
        public:
            fem_solver() :m_method2{ nullptr }, m_method{nullptr}{}
            ~fem_solver()
            {
                if(m_method2 != nullptr)
                    delete m_method2;
                if(m_method != nullptr)
                    delete m_method;
            }
            // terms, method, mesh, solver, result
            const int               elliptic_solver(_Problem*, _Mesh*, _Result*);
            const int               elliptic_solver_gauss(_Problem*, _Mesh*, _Result*);
            const double            get_value(const _Mesh&, const _Result&, const Mesh::Point& p) const;
            const double            get_value(const _Method2*, const _Mesh&, const _Result&, const Mesh::Point& p) const;
            const double            get_value(const _Method*, const _Mesh&, const _Result&, const Mesh::Point& p) const;
            const double            get_value(const _Mesh&, const _Result&, const Mesh::Point& p, const int i) const;
            const Mesh::Point        get_gradvalue(const _Mesh&, const _Result&, const Mesh::Point& p) const;
            const Mesh::Point        get_gradvalue(const _Mesh&, const _Result&, const Mesh::Point& p, const int i) const;
        private:
            _Method*             m_method;
            _Method*             m_method2;
            //_Method2*             m_method2;
        };

        template<class _Problem, class _Mesh, class _Result>
        const int fem_solver<_Problem, _Mesh, _Result>::elliptic_solver(_Problem* problem, _Mesh* mesh, _Result* result)
        {
            std::vector<double> res;
            std::vector<double> res2;
            //std::shared_ptr<Algebra::MatrixSkyline> matrix{ new Algebra::MatrixSkyline() };
            Algebra::MatrixSkyline* matrix{ new Algebra::MatrixSkyline() };
            std::vector<double> rhs;
            if (m_method != nullptr)
                delete m_method;

            m_method = new _Method{ problem, mesh, matrix, &rhs };

            m_method->Discretization();

            Algebra::ESolver esl{ Algebra::Solvers::BiCGStab };
            //std::cout « "Size:\t" « matrix->GetSize() « std::endl;

            //std::cout « matrix->GetSize() « std::endl;
            *result = esl.Solve(*matrix, rhs, *result, res, 100000, 1e-13);
            //std::cout « matrix->GetSize() « std::endl;

            //esl.Pardiso(*matrix, rhs, *result);
            //res.resize(matrix2->GetSize());
            //for (int i = 0; i < matrix2->GetSize(); ++i)
            //{
                //for (int j = 0; j < matrix2->GetSize(); ++j)
                //{
                    //res[i] += result->operator[](j) * (matrix2->GetElement(i, j));
                //}
            //}
            delete matrix;

            return 0;
        }

        template<class _Problem, class _Mesh, class _Result>
        const int fem_solver<_Problem, _Mesh, _Result>::elliptic_solver_gauss(_Problem* problem, _Mesh* mesh, _Result* result)
        {
            std::vector<double> res;
            std::vector<double> res2;
            //std::shared_ptr<Algebra::MatrixSkyline> matrix{ new Algebra::MatrixSkyline() };
            //Algebra::MatrixSkyline* matrix{ new Algebra::MatrixSkyline() };
            Algebra::Matrix* matrix2{ new Algebra::Matrix() };
            std::vector<double> rhs;
            //if (m_method != nullptr)
```

```
            //    delete m_method;
            if (m_method2 != nullptr)
                delete m_method2;

            //m_method = new _Method{ problem, mesh, matrix, &rhs };
            m_method2 = new _Method2{ problem, mesh, matrix2, &rhs };
            //m_method->Discretization();
            m_method2->Discretization();
            //Algebra::ESolver esl{ Algebra::Solvers::BiCGStab };
             //std::cout « "Size:\t" « matrix->GetSize() « std::endl;
             Algebra::ESolver esl{ Algebra::Solvers::Gauss };
             //std::cout « matrix->GetSize() « std::endl;
             //*result = esl.Solve(*matrix, rhs, *result, res, 100000, 1e-13);
             //std::cout « matrix->GetSize() « std::endl;
             esl.Gauss(*matrix2, rhs, *result);
             //esl.Pardiso(*matrix, rhs, *result);
             //res.resize(matrix2->GetSize());
             //for (int i = 0; i < matrix2->GetSize(); ++i)
             //{
                 //for (int j = 0; j < matrix2->GetSize(); ++j)
                 //{
                     //res[i] += result->operator[](j) * (matrix2->GetElement(i, j));
                 //}
             //}
             //delete matrix;
             delete matrix2;
             return 0;
         }
         template<class _Problem, class _Mesh, class _Result>
         const double fem_solver<_Problem, _Mesh, _Result>::get_value(const _Mesh& mesh, const _Result& res, const Mesh::Point& p) const
         {
             if (m_method2 != nullptr)
                 return m_method2->GetSolution(mesh, res, p);
             return 0.;
         }
         template<class _Problem, class _Mesh, class _Result>
         const double fem_solver<_Problem, _Mesh, _Result>::get_value(const _Method2* method2, const _Mesh& mesh, const _Result& res, const Mesh::Point& p) const
         {
             if (method2 != nullptr)
                 return method2->GetSolution(mesh, res, p);
             return 0.;
         }
         template<class _Problem, class _Mesh, class _Result>
         const double fem_solver<_Problem, _Mesh, _Result>::get_value(const _Method* method2, const _Mesh& mesh, const _Result& res, const Mesh::Point& p) const
         {
             if (method2 != nullptr)
                 return method2->GetSolution(mesh, res, p);
             return 0.;
         }

         template<class _Problem, class _Mesh, class _Result>
         const double fem_solver<_Problem, _Mesh, _Result>::get_value(const _Mesh& mesh, const _Result& res, const Mesh::Point& p, const int i) const
         {
             if (m_method2 != nullptr)
                 return m_method2->GetSolution(mesh, res, p, i);
             return 0.;
         }

         template<class _Problem, class _Mesh, class _Result>
         const Mesh::Point fem_solver<_Problem, _Mesh, _Result>::get_gradvalue(const _Mesh& mesh, const _Result& res, const Mesh::Point& p) const
         {
             if (m_method2 != nullptr)
                 return m_method2->GetGradSolution(mesh, res, p);
             return Mesh::Point(0, 0, 0);
         }

         template<class _Problem, class _Mesh, class _Result>
         const Mesh::Point fem_solver<_Problem, _Mesh, _Result>::get_gradvalue(const _Mesh& mesh, const _Result& res, const Mesh::Point& p, const int i) const
         {
             if (m_method2 != nullptr)
                 return m_method2->GetGradSolution(mesh, res, p, i);
             return Mesh::Point(0, 0, 0);
         }
     }
}
#endif // !CORENC_SOLVERS_FEM_SOLVER_H_
```

7.119 Solvers/fem_solver_lib.h File Reference

```
#include "../CoreNCFEM/Grids/TriangularMesh.h"
#include "../CoreNCFEM/Methods/FEMethod.h"
#include "../Problems/DiffusionScalar.h"
#include "../CoreNCA/MatrixSkyline.h"
#include "../CoreNCFEM/Methods/FEAnalysis.h"
#include <chrono>
#include <iostream>
#include <fstream>
#include <eigen3/Eigen/SparseCore>
#include <cstdlib>
#include <string>
#include <eigen3/Eigen/Cholesky>
#include <eigen3/Eigen/Jacobi>
#include <eigen3/Eigen/Householder>
#include <eigen3/Eigen/IterativeLinearSolvers>
#include <eigen3/unsupported/Eigen/IterativeSolvers>
#include <eigen3/Eigen/LU>
#include <eigen3/unsupported/Eigen/SparseExtra>
#include <eigen3/Eigen/SparseLU>
#include <eigen3/Eigen/UmfPackSupport>
```

Classes

- class corenc::solvers::fem_solver_lib< _Problem, _Mesh, _Result >

Namespaces

- namespace corenc
- namespace corenc::solvers

7.120 fem_solver_lib.h

Go to the documentation of this file.

```
#ifndef CORENC_SOLVERS_fem_solver_lib_H_
#define CORENC_SOLVERS_fem_solver_lib_H_

#include "../CoreNCFEM/Grids/TriangularMesh.h"
#include "../CoreNCFEM/Methods/FEMethod.h"
#include "../Problems/DiffusionScalar.h"
#include "../CoreNCA/MatrixSkyline.h"
#include "../CoreNCFEM/Methods/FEAnalysis.h"
#include <chrono>

/*#include <eigen3/Eigen/SparseCore>
//#include <eigen3/Eigen/Sparse>
//#include <eigen3/Eigen/SparseLU>
//#include <eigen3/Eigen/SparseCholesky>
#include <eigen3/Eigen/Cholesky>
//#include <eigen3/Eigen/Dense>
#include <eigen3/Eigen/UmfPackSupport>
//#include <eigen3/Eigen/SparseCore>
#include <eigen3/unsupported/Eigen/SparseExtra>
#include <eigen3/Eigen/SparseLU>*/

#include <iostream>
#include <fstream>
#include <eigen3/Eigen/SparseCore>
```

```
#include <cstdlib>
#include <string>
#include <eigen3/Eigen/Cholesky>
#include <eigen3/Eigen/Jacobi>
#include <eigen3/Eigen/Householder>
#include <eigen3/Eigen/IterativeLinearSolvers>
#include <eigen3/unsupported/Eigen/IterativeSolvers>
#include <eigen3/Eigen/LU>
#include <eigen3/unsupported/Eigen/SparseExtra>
#include <eigen3/Eigen/SparseLU>
#include <eigen3/Eigen/UmfPackSupport>


// FINITE ELEMENT METHOD SOLVER ONLY IN SPACE

namespace corenc
{
    namespace solvers
    {
        template<class _Problem, class _Mesh, class _Result>
        class fem_solver_lib
        {
            using _Method = method::FEMethod<_Problem, _Mesh, Algebra::MatrixSkyline>;
            using _Method2 = method::FEMethod<_Problem, _Mesh, Algebra::Matrix>;
        public:
            fem_solver_lib() :m_method2{ nullptr }, m_method{nullptr}{}
            ~fem_solver_lib()
            {
                if(m_method2 != nullptr)
                    delete m_method2;
                if(m_method != nullptr)
                    delete m_method;
            }
            // terms, method, mesh, solver, result
            const int               elliptic_solver(_Problem*, _Mesh*, _Result*);
            const int               elliptic_solver_gauss(_Problem*, _Mesh*, _Result*);
            const double            get_value(const _Mesh&, const _Result&, const Mesh::Point& p) const;
            const double            get_value(const _Method2*, const _Mesh&, const _Result&, const Mesh::Point& p) const;
            const double            get_value(const _Method*, const _Mesh&, const _Result&, const Mesh::Point& p) const;
            const double            get_value(const _Mesh&, const _Result&, const Mesh::Point& p, const int i) const;
            const Mesh::Point        get_gradvalue(const _Mesh&, const _Result&, const Mesh::Point& p) const;
            const Mesh::Point        get_gradvalue(const _Mesh&, const _Result&, const Mesh::Point& p, const int i) const;
        private:
            _Method*             m_method;
            _Method*             m_method2;
            //_Method2*             m_method2;
        };

        template<class _Problem, class _Mesh, class _Result>
        const int fem_solver_lib<_Problem, _Mesh, _Result>::elliptic_solver(_Problem* problem, _Mesh* mesh, _Result* result)
        {


            std::vector<double> res;
            std::vector<double> res2;
            //std::shared_ptr<Algebra::MatrixSkyline> matrix{ new Algebra::MatrixSkyline() };
            Algebra::MatrixSkyline* matrix{ new Algebra::MatrixSkyline() };



            std::vector<double> rhs;
            if (m_method != nullptr)
                delete m_method;

            m_method = new _Method{ problem, mesh, matrix, &rhs };

            m_method->Discretization();

            int n = matrix->GetSize();
            Eigen::SparseMatrix<double> eA(n, n);
            for (int i = 0; i < n; ++i)
            {
                for (int j = 0; j < n; ++j)
                {
                    auto elem = matrix->GetElement(i, j);
                     if (fabs(elem) > 1e-12)
                         eA.insert(i, j) = elem;
                 }
             }
             eA.makeCompressed();

```

```
        //Algebra::ESolver esl{ Algebra::Solvers::BiCGStab };


        //std::cout « "Size:\t" « matrix->GetSize() « std::endl;

        //std::cout « matrix->GetSize() « std::endl;
        //*result = esl.Solve(*matrix, rhs, *result, res, 100000, 1e-13);
        Eigen::MatrixMarketIterator<double> it("matr");
        Eigen::VectorXd xx(n);
        for (int i = 0; i < n; ++i)
            xx[i] = rhs[i];
        std::chrono::steady_clock::time_point beg{ std::chrono::steady_clock::now() };

        //Eigen::SparseLU<Eigen::SparseMatrix<double» chol;//(eA);
        //Eigen::BiCGSTAB<Eigen::SparseMatrix<double» chol;//(eA);

        Eigen::UmfPackLU<Eigen::SparseMatrix<double» chol;//(eA);

        //chol.analyzePattern(eA);
        chol.compute(eA);
        //chol.factorize(eA);
        if (chol.info() != Eigen::Success)
            std::cout « "oops" « std::endl;
        Eigen::Matrix<double, Eigen::Dynamic, 1> bb;
        //auto bb = chol.solve(xx);
        bb = chol.solve(xx);
        if (chol.info() != Eigen::Success)
            std::cout « "oops xx" « std::endl;

        //Eigen::saveMarket(eA, "matrix.mtx");
        //Eigen::saveMarketVector(xx, "vector.mtx");
        //Eigen::saveMarketVector(bb, "MatrixName_x.mtx");


        std::chrono::steady_clock::time_point end{ std::chrono::steady_clock::now() };
        auto dur = std::chrono::duration_cast<std::chrono::milliseconds>(end - beg).count();
        std::cout « dur « std::endl;

        result->resize(n);
        for (int i = 0; i < n; ++i)
            (*result)[i] = bb[i];

        //std::cout « matrix->GetSize() « std::endl;

        //esl.Pardiso(*matrix, rhs, *result);
        //res.resize(matrix2->GetSize());
        //for (int i = 0; i < matrix2->GetSize(); ++i)
        //{
            //for (int j = 0; j < matrix2->GetSize(); ++j)
            //{
                //res[i] += result->operator[](j) * (matrix2->GetElement(i, j));
            //}
        //}
        delete matrix;

        return 0;
    }

    template<class _Problem, class _Mesh, class _Result>
    const int fem_solver_lib<_Problem, _Mesh, _Result>::elliptic_solver_gauss(_Problem* problem, _Mesh* mesh, _Result* result)
    {
        std::vector<double> res;
        std::vector<double> res2;
        //std::shared_ptr<Algebra::MatrixSkyline> matrix{ new Algebra::MatrixSkyline() };
        //Algebra::MatrixSkyline* matrix{ new Algebra::MatrixSkyline() };
        Algebra::Matrix* matrix2{ new Algebra::Matrix() };
        std::vector<double> rhs;
        //if (m_method != nullptr)
        //    delete m_method;
        if (m_method2 != nullptr)
            delete m_method2;

        //m_method = new _Method{ problem, mesh, matrix, &rhs };
        m_method2 = new _Method2{ problem, mesh, matrix2, &rhs };
        //m_method->Discretization();
        m_method2->Discretization();
        //Algebra::ESolver esl{ Algebra::Solvers::BiCGStab };
        //std::cout « "Size:\t" « matrix->GetSize() « std::endl;
        Algebra::ESolver esl{ Algebra::Solvers::Gauss };
        //std::cout « matrix->GetSize() « std::endl;
        //*result = esl.Solve(*matrix, rhs, *result, res, 100000, 1e-13);
        //std::cout « matrix->GetSize() « std::endl;
        esl.Gauss(*matrix2, rhs, *result);
        //esl.Pardiso(*matrix, rhs, *result);
        //res.resize(matrix2->GetSize());
        //for (int i = 0; i < matrix2->GetSize(); ++i)
```

```
            //{
                //for (int j = 0; j < matrix2->GetSize(); ++j)
                //{
                    //res[i] += result->operator[](j) * (matrix2->GetElement(i, j));
                //}
            //}
            //delete matrix;
            delete matrix2;
            return 0;
        }
        template<class _Problem, class _Mesh, class _Result>
        const double fem_solver_lib<_Problem, _Mesh, _Result>::get_value(const _Mesh& mesh, const _Result& res, const Mesh::Point& p) const
        {
            if (m_method2 != nullptr)
                return m_method2->GetSolution(mesh, res, p);
            return 0.;
        }
        template<class _Problem, class _Mesh, class _Result>
        const double fem_solver_lib<_Problem, _Mesh, _Result>::get_value(const _Method2* method2, const _Mesh& mesh, const _Result& res, const Mesh::Point& p) const
        {
            if (method2 != nullptr)
                return method2->GetSolution(mesh, res, p);
            return 0.;
        }
        template<class _Problem, class _Mesh, class _Result>
        const double fem_solver_lib<_Problem, _Mesh, _Result>::get_value(const _Method* method2, const _Mesh& mesh, const _Result& res, const Mesh::Point& p) const
        {
            if (method2 != nullptr)
                return method2->GetSolution(mesh, res, p);
            return 0.;
        }

        template<class _Problem, class _Mesh, class _Result>
        const double fem_solver_lib<_Problem, _Mesh, _Result>::get_value(const _Mesh& mesh, const _Result& res, const Mesh::Point& p, const int i) const
        {
            if (m_method2 != nullptr)
                return m_method2->GetSolution(mesh, res, p, i);
            return 0.;
        }

        template<class _Problem, class _Mesh, class _Result>
        const Mesh::Point fem_solver_lib<_Problem, _Mesh, _Result>::get_gradvalue(const _Mesh& mesh, const _Result& res, const Mesh::Point& p) const
        {
            if (m_method2 != nullptr)
                return m_method2->GetGradSolution(mesh, res, p);
            return Mesh::Point(0, 0, 0);
        }

        template<class _Problem, class _Mesh, class _Result>
        const Mesh::Point fem_solver_lib<_Problem, _Mesh, _Result>::get_gradvalue(const _Mesh& mesh, const _Result& res, const Mesh::Point& p, const int i) const
        {
            if (m_method2 != nullptr)
                return m_method2->GetGradSolution(mesh, res, p, i);
            return Mesh::Point(0, 0, 0);
        }
    }
}
#endif // !CORENC_SOLVERS_fem_solver_lib_H_
```

7.121 Tests/FiniteElements/test_case_rectanglebasis.cpp File Reference

```
#include "test_case_rectanglebasis.h"
#include "../../CoreNCFEM/FiniteElements/Rectangle.h"
```

7.122 Tests/FiniteElements/test_case_rectanglebasis.h File Reference

Classes

- class corenc::tests::test_case_rectanglebasis

Namespaces

- namespace corenc
- namespace corenc::tests

Macros

- #define CORENC_TEST_CASE_RECTANGLEBASIS_H_

7.122.1 Macro Definition Documentation

7.122.1.1 CORENC_TEST_CASE_RECTANGLEBASIS_H_

```
#define CORENC_TEST_CASE_RECTANGLEBASIS_H_
```

7.123 test_case_rectanglebasis.h

Go to the documentation of this file.

```
#pragma once
#ifndef CORENC_TEST_CASE_RECTANGLEBASIS_H_
#define CORENC_TEST_CASE_RECTANGLEBASIS_H_
namespace corenc
{
    namespace tests
    {
        class test_case_rectanglebasis
        {
         public:
             test_case_rectanglebasis();
             ~test_case_rectanglebasis();
             const int           mass_matrix() const;
             const int           stress_matrix() const;
         };
     }
 }
 #endif // !CORENC_TEST_CASE_RECTANGLEBASIS_H_
```

7.124 Tests/FiniteElements/test_case_trianglebasis.cpp File Reference

```
#include "test_case_trianglebasis.h"
#include "../../CoreNCFEM/FiniteElements/Triangle.h"
```

7.125 Tests/FiniteElements/test_case_trianglebasis.h File Reference

Classes

- class corenc::tests::test_case_trianglebasis

Namespaces

- namespace corenc
- namespace corenc::tests

Macros

- #define CORENC_TEST_CASE_TRIANGLEBASIS_H_

7.125.1 Macro Definition Documentation

7.125.1.1 CORENC_TEST_CASE_TRIANGLEBASIS_H_

```
#define CORENC_TEST_CASE_TRIANGLEBASIS_H_
```

7.126 test_case_trianglebasis.h

Go to the documentation of this file.

```
#pragma once
#ifndef CORENC_TEST_CASE_TRIANGLEBASIS_H_
#define CORENC_TEST_CASE_TRIANGLEBASIS_H_
namespace corenc
{
    namespace tests
    {
        class test_case_trianglebasis
        {
         public:
            test_case_trianglebasis();
            ~test_case_trianglebasis();
            const int           mass_matrix() const;
            const int           stress_matrix() const;
         };
     }
}
#endif // !CORENC_TEST_CASE_TRIANGLEBASIS_H_
```

7.127 Tests/test_case_elliptic_fem.cpp File Reference

```
#include "test_case_elliptic_fem.h"
#include "../CoreNCFEM/Grids/TriangularMesh.h"
#include "../CoreNCFEM/Grids/RegularMesh.h"
#include "../CoreNCFEM/Methods/FEMethod.h"
#include "../Problems/DiffusionScalar.h"
#include "../CoreNCA/MatrixSkyline.h"
#include "../CoreNCFEM/Methods/FEAnalysis.h"
#include "../Solvers/fem_solver.h"
#include "../CoreNCFEM/GaussianField.h"
#include "../CoreNCFEM/FiniteElements/Triangle.h"
#include <random>
#include <math.h>
```

Macros

- #define _USE_MATH_DEFINES

Functions

- const double kekus (const double c, const double a=0, const double b=90.)

7.127.1 Macro Definition Documentation

7.127.1.1 _USE_MATH_DEFINES

```
#define _USE_MATH_DEFINES
```

7.127.2 Function Documentation

7.127.2.1 kekus()

```
const double kekus (
            const double c,
            const double a = 0,
            const double b = 90. )
```

7.128 Tests/test_case_elliptic_fem.h File Reference

Classes

- class corenc::test_case_elliptic_fem

Namespaces

- namespace corenc

7.129 test_case_elliptic_fem.h

Go to the documentation of this file.

```
#ifndef CORENC_TEST_CASE_ELLIPTIC_FEM_H_
#define CORENC_TEST_CASE_ELLIPTIC_FEM_H_

// SOME TEST PROBLEMS FOR ELLIPTIC CASE WITH FEM && DG\
// 0th, 1st, 2nd order definitely maybe more high-order
// LAGRANGE && HIERARHICAL BASIS FUNCTIONS
// LATER MAYBE EVEN TESTS WITH MULTISCALE

namespace corenc
{
    class test_case_elliptic_fem
    {
    public:
        test_case_elliptic_fem();
        ~test_case_elliptic_fem();
        //const int                 test_case_elliptic_fem_3d_tetra() const;
        const int                   elliptic_fem_2d_tria() const;
        const int                   elliptic_fem_solver() const;
        const int                   elliptic_fem_square_lin_basis() const;
        const int                   elliptic_fem_hp_fixed(const int h_ref_max, const int p_ref_max)
    const;
        const int                   elliptic_fem_hp_fixed_triangle(const int h_ref_max, const int
    p_ref_max) const;
        const int                   elliptic_fem_hp_lagrange_triangle(const int h_ref_max, const int
    p_ref_max) const;
        const int                   elliptic_fem_hxhy_fixed_triangle(const int hx_max, const int hy_max)
    const;
        const int                   conv_diff_fem_fixed_triangle(const int h_ref_max, const int
    p_ref_max) const;
        const int                   global_matrix(const int h_ref_max, const int p_ref_max) const;
        //const int                 test_case_elliptic_fem_square_2nd_basis() const;
        //const int                 test_case_elliptic_fem_square_nth_basis() const;
        const int                   elliptic_2layer_fem_2d_tria_h() const;
        const int                   elliptic_fem_2d_rect_source() const;
        const int                   elliptic_gaussian_triangle() const;
        const int                   mass_matrix_3rd_order() const;
        const int                   strees_matrix_3rd_order() const;
        const int                   mass_matrix_4th_order() const;
        const int                   stress_matrix_4th_order() const;
        const int                   homotopy_conv_diff_fem(const double step) const;
        //const int                 test_case_elliptic_fem_2d_rect() const;
        //const int                 test_case_elliptic_fem_3d_hex() const;
        //const int                 test_case_elliptic_dg_3d_tetra() const;
        //const int                 test_case_elliptic_dg_2d_tria() const;
        //const int                 test_case_elliptic_dg_2d_rect() const;
        //const int                 test_case_elliptic_dg_3d_hex() const;
    };
}

#endif // !CORENC_TEST_CASE_ELLIPTIC_FEM_H_
```

7.130 Tests/test_case_regular_mesh.cpp File Reference

```
#include "test_case_regular_mesh.h"
#include "../CoreNCFEM/Grids/RegularMesh.h"
```

7.131 Tests/test_case_regular_mesh.h File Reference

Classes

- class corenc::tests::test_case_regular_mesh

Namespaces

- namespace corenc
- namespace corenc::tests

Macros

- #define CORENC_TEST_CASE_REGULAR_MESH_H_

7.131.1 Macro Definition Documentation

7.131.1.1 CORENC_TEST_CASE_REGULAR_MESH_H_

```
#define CORENC_TEST_CASE_REGULAR_MESH_H_
```

7.132 test_case_regular_mesh.h

Go to the documentation of this file.

```
#pragma once
#ifndef CORENC_TEST_CASE_REGULAR_MESH_H_
#define CORENC_TEST_CASE_REGULAR_MESH_H_

namespace corenc
{
    namespace tests
    {
        class test_case_regular_mesh
         {
         public:
             test_case_regular_mesh();
             ~test_case_regular_mesh();
             const int                    construct_mesh() const;
         };
     }
 }

 #endif // !CORENC_TEST_CASE_REGULAR_MESH_H_
```

7.133 Tests/test_case_solver.cpp File Reference

```
#include "test_case_solver.h"
#include "../CoreNCFEM/Grids/TriangularMesh.h"
#include "../CoreNCFEM/Grids/RegularMesh.h"
#include "../CoreNCFEM/Methods/FEMethod.h"
#include "../Problems/DiffusionScalar.h"
#include "../CoreNCA/MatrixSkyline.h"
#include "../CoreNCFEM/Methods/FEAnalysis.h"
#include "../Solvers/fem_solver.h"
#include "../CoreNCFEM/GaussianField.h"
#include <random>
#include <math.h>
```

Macros

- #define _USE_MATH_DEFINES

Functions

- const int solver (const Algebra::Matrix &matrix, double ∗x, double ∗res)

7.133.1 Macro Definition Documentation

7.133.1.1 _USE_MATH_DEFINES

```
#define _USE_MATH_DEFINES
```

7.133.2 Function Documentation

7.133.2.1 solver()

```
const int solver (
            const Algebra::Matrix & matrix,
            double * x,
            double * res )
```

7.134 Tests/test_case_solver.h File Reference

Classes

- class corenc::test_case_solver

Namespaces

- namespace corenc

7.135 test_case_solver.h

Go to the documentation of this file.

```
#ifndef CORENC_TEST_CASE_SOLVER_H_
#define CORENC_TEST_CASE_SOLVER_H_

// SOME TEST PROBLEMS FOR ELLIPTIC CASE WITH FEM && DG\
// 0th, 1st, 2nd order definitely maybe more high-order
// LAGRANGE && HIERARHICAL BASIS FUNCTIONS
// LATER MAYBE EVEN TESTS WITH MULTISCALE

namespace corenc
{
    class test_case_solver
    {
    public:
        test_case_solver();
        ~test_case_solver();
        const int                  gauss_solver() const;
    };
}

#endif // !CORENC_TEST_CASE_SOLVER_H_
```

7.136 Tests/test_cases.cpp File Reference

```
#include "test_cases.h"
#include "test_case_elliptic_fem.h"
#include "test_case_solver.h"
#include "test_case_regular_mesh.h"
#include "FiniteElements/test_case_rectanglebasis.h"
#include "FiniteElements/test_case_trianglebasis.h"
#include <iostream>
#include <thread>
#include <future>
#include <chrono>
#include <ostream>
#include "../colors.h"
#include "test_conv_diff.h"
```

7.137 Tests/test_cases.h File Reference

```
#include <functional>
#include <ostream>
```

Classes

- class corenc::test_cases

Namespaces

- namespace corenc

Macros

- #define CORENC_TEST_CASES_H_

7.137.1 Macro Definition Documentation

7.137.1.1 CORENC_TEST_CASES_H_

```
#define CORENC_TEST_CASES_H_
```

7.138 test_cases.h

Go to the documentation of this file.

```
#pragma once
#ifndef CORENC_TEST_CASES_H_
#define CORENC_TEST_CASES_H_
#include <functional>
#include <ostream>
namespace corenc
{
    class test_cases
    {
     public:
         test_cases();
         ~test_cases();
         const int perform() const;
         const int perform(const std::function<const int()>&) const;
         const int perform(const std::function<const int(std::ostream&)>&, std::ostream&) const;
     };
}


#endif // !CORENC_TEST_CASES_H_
```

7.139 Tests/test_conv_diff.cpp File Reference

```
#include "test_conv_diff.h"
#include "../CoreNCFEM/Grids/TriangularMesh.h"
#include "../CoreNCFEM/Grids/RegularMesh.h"
#include "../CoreNCFEM/Methods/FEMethod.h"
#include "../Problems/DiffusionScalar.h"
#include "../CoreNCA/MatrixSkyline.h"
#include "../CoreNCFEM/Methods/FEAnalysis.h"
#include "../Solvers/fem_solver.h"
#include "../Solvers/fem_solver_lib.h"
#include "../CoreNCFEM/GaussianField.h"
#include "../CoreNCFEM/FiniteElements/Triangle.h"
#include <random>
#include <math.h>
```

Macros

- #define _USE_MATH_DEFINES

7.139.1 Macro Definition Documentation

7.139.1.1 _USE_MATH_DEFINES

```
#define _USE_MATH_DEFINES
```

7.140 Tests/test_conv_diff.h File Reference

Classes

- class corenc::test_conv_diff

Namespaces

- namespace corenc

7.141 test_conv_diff.h

Go to the documentation of this file.

```
#ifndef TEST_CONV_DIFF_H
#define TEST_CONV_DIFF_H

namespace corenc
{
    class test_conv_diff
    {
    public:
        test_conv_diff(){};
         ~test_conv_diff(){};
         void conv_diff_fem(const int h_ref_max, const int p_ref_max = 1) const;
         void conv_diff_eigen(const int h_ref_max, const int p_ref_max = 1) const;
     };
 }

 #endif // TEST_CONV_DIFF_H
```

Index