SemSolver

Generated by Doxygen 1.7.3

Mon Sep 19 2011 18:17:27

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class 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 SemSolver Namespace Reference

Project main namespace.

Namespaces

- namespace Assembler

  Assembler namespace.

- namespace IO

  Namespace for Input/Output operations on SemSolver Classes.

- namespace PostProcessor
- namespace PreProcessor

  PreProcessor namespace.

- namespace Solver

  Solver namespace.

Classes

- class BilinearTransformation

  Class representing a bilinear tranformation of 2D euclidean space.

- class BoundaryConditions

  Class for handling boundary conditions on a SemGeometry.

- class BoundaryConditions< 2, X >

  Class for handling boundary conditions on a 2D SemGeometry.

- class DiffusionConvectionReactionEquation

  Class for handling Diffusion-Convection-Reaction steady equation.

- class DiffusionConvectionReactionEquation< 2, X >

  Class for handling Diffusion-Convection-Reaction steady equation on 2D spaces.

- class Equation

  Virtual class for handling general equation.

- class Function

  Prototype class for mathematical functions $: X \rightarrow Y$.

- class HilbertSpace

  prototype class for handling the concept of Hilbert Space

- class Homeomorphism

  Prototype class for mathematical homemorphism $: X \rightarrow Y$.

- class Matrix

  Class for handling mathematical matrices.

- class MultiIndex

  Class multi-index notation.

- class Point< 2, X >

  Class for handling 2D euclidean points.

- class PointsBimap< 2, X >

  Class for handling bi-directional maps between 2D Points and Integers.

- class PointsMap< 2, X, Y >

  Class for handling maps with 2D Point as KeyType.

- class PointsSet< 2, X >

  Class for handling sets of 2D Points.

- class Polygon< 2, X >

  Class for handling 2D polygons.

- class Polygonation< 2, X >

  Class for handling 2D polygonations.

- class PolygonWithHoles< 2, X >

  Class for handling 2D polygons with holes.

- class Polynomial

  Class for handing the mathematical concept of polynomials over X.

- class PolynomialFunction

  Class for handling polynomial separable functions.

- class Problem

  Class for handling a mathematical problem given by an equation, boundary conditions, geometry and parameters.

- class PSLG

  Class for handing Planar Straight Line Graphs.

- class ScriptFunction< Point< d, X >, Y >

  Class for handling function from Euclidean space X^d to scalar space Y defined in scripts.

- class ScriptFunction< Point< d, X >, Vector< Y > >

  Class for handling function from Euclidean space X^d to Vectorial space Y^n defined inscripts.

- class ScriptFunction< Point< 2, X >, Y >

  Class for handling function from 2D Euclidean space X^2 to scalar space Y defined in scripts.

- class ScriptFunction< Point< 2, X >, Vector< Y > >

  Class for handling function from 2D Euclidean space X^2 to Vectorial space Y^n defined in scripts.

- class Segment< 2, X >
- class SegmentsMap
- class SemFunction

  Class for handling spectral elements functions.

- class SemFunction< 2, X >

  Class for handling 2D spectral elements functions.

- class SemGeometry

  Class for describing the geometry of a SemProblem.

- class SemGeometry< 2, X >
- class SemParameters
- class SemSpace
- class SemSpace< 2, X >
- class Vector

Typedefs

- typedef std::list< int > Sequence
- typedef std::list< Sequence > Sequences_list

Functions

- template<class X >
  Matrix< X > operator* (Matrix< X > const &mat1, Matrix< X > const &mat2)

  Matrix multiplication.

- template<class X >
  Matrix< X > operator+ (Matrix< X > const &mat1, Matrix< X > const &mat2)

  Matrix summation.

- template<class X >
  Vector< X > operator+ (Vector< X > const &vec1, Vector< X > const &vec2)

  Vector summation.

- template<class X >
  X scalar (Vector< X > const &vec1, Vector< X > const &vec2)

  Vector multiplication for a scalar.

5.1.1 Detailed Description

Project main namespace. Class for handling maps with int as KeyType and Point as MappedType.

The SemSolver namespace.

Parameters

d	Dimension of the space
X	Must be a type for which operations +, -, * and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of *

5.1.2 Typedef Documentation

5.1.2.1 typedef std::list<int> SemSolver::Sequence

Definition at line 8 of file sequence.hpp.

5.1.2.2 typedef std::list<Sequence> SemSolver::Sequences_list

Definition at line 10 of file sequenceslist.hpp.

5.1.3 Function Documentation

5.1.3.1 template<class X > SemSolver::Matrix< X > SemSolver::operator* (Matrix< X > const & *mat1*, Matrix< X > const & *mat2*)

Matrix multiplication.

Parameters

mat1	First matrix
mat2	Second matrix

Returns

Product matrix

Definition at line 108 of file matrix.hpp.

5.1.3.2 template<class X > SemSolver::Matrix< X > SemSolver::operator+ (Matrix< X > const & *mat1*, Matrix< X > const & *mat2*)

Matrix summation.

Parameters

mat1	First matrix
mat2	Second matrix

Returns

Sum matrix

Definition at line 133 of file matrix.hpp.

5.1.3.3 template<class X > Vector<X> SemSolver::operator+ (Vector< X > const & *vec1*, Vector< X > const & *vec2*)

Vector summation.

Definition at line 51 of file vector.hpp.

5.1.3.4 template<class X > X SemSolver::scalar (Vector< X > const & *vec1*, Vector< X > const & *vec2*)

Vector multiplication for a scalar.

Definition at line 60 of file vector.hpp.

5.2 SemSolver::Assembler Namespace Reference

Assembler namespace.

Functions

- template<class X >
  void compute_algebraic_system (const SemSpace< 2, X > &space, const Problem< 2, X > &problem, Matrix< X > &A, Vector< X > &f)

  The computed system is stored in the Matrix and vectored refereced by A and f.

- template<class X >
  void compute_border_matrix (const SemSpace< 2, X > &space, const BoundaryConditions< 2, X > *boundary_conditions, const Function< Point< 2, X >, X > *diffusion, double const &penality, Matrix< X > &matrix)

  The computed matrix is stored in the Matrix referenced by matrix.

- template<class X >
  void compute_border_vector (const SemSpace< 2, X > &space, const BoundaryConditions< 2, X > *boundary_conditions, const Function< Point< 2, X >, X > *diffusion, const double &penality, Vector< X > &vector)

  The computed matrix is stored in the Matrix referenced by matrix.

- template<class X >
  void compute_convection_matrix (const SemSpace< 2, X > &space, const Function< Point< 2, X >, Vector< X > > *convection, Matrix< X > &matrix)

  The computed matrix is stored in the Matrix referenced by matrix.

- template<class X >
  void compute_diffusion_matrix (const SemSpace< 2, X > &space, const Function< Point< 2, X >, X > *diffusion, Matrix< X > &matrix)

  The computed matrix is stored in the Matrix referenced by matrix.

- template<class X >
  void compute_forcing_vector (const SemSpace< 2, X > &space, const Function< Point< 2, X >, X > *forcing, Vector< X > &vector)

  The computed matrix is stored in the Matrix referenced by matrix.

- template< class X >
void compute_reaction_matrix (const SemSpace< 2, X > &space, Function< Point< 2, X >, X > const *reaction, Matrix< double > &matrix)

  The computed matrix is stored in the Matrix referenced by matrix.

5.2.1 Detailed Description

Assembler namespace. This namespace provides algorithms for the constuction of the algebraic matrices and vectors associated to the discretized problem from geometric and functional information stored in a SemProblem.

5.2.2 Function Documentation

5.2.2.1 template< class X > void SemSolver::Assembler::compute_algebraic_system (const SemSpace< 2, X > & *space*, const Problem< 2, X > & *problem*, Matrix< X > & *A*, Vector< X > & *f*)

The computed system is stored in the Matrix and vectored refereced by A and f.

Compute the algebraic system A $*$ u = f associated to a 2D elliptic problem in a Spectral Element Space

Definition at line 29 of file computealgebraicsystem.hpp.

5.2.2.2 template< class X > void SemSolver::Assembler::compute_border_matrix (const SemSpace< 2, X > & *space*, const BoundaryConditions< 2, X > $*$ *boundary_conditions*, const Function< Point< 2, X >, X > $*$ *diffusion*, double const & *penality*, Matrix< X > & *matrix*)

The computed matrix is stored in the Matrix referenced by matrix.

Compute the boundary matrix for a 2D elliptic problem in a Spectral Element Space using penality method

Definition at line 22 of file computebordermatrix.hpp.

5.2.2.3 template< class X > void SemSolver::Assembler::compute_border_vector (const SemSpace< 2, X > & *space*, const BoundaryConditions< 2, X > $*$ *boundary_conditions*, const Function< Point< 2, X >, X > $*$ *diffusion*, const double & *penality*, Vector< X > & *vector*)

The computed matrix is stored in the Matrix referenced by matrix.

Compute the boundary vector for a 2D elliptic problem in a Spectral Element Space using penality method

Definition at line 20 of file computebordervector.hpp.

5.2.2.4 template<class X > void SemSolver::Assembler::compute_convection_matrix (const SemSpace< 2, X > & *space*, const Function< Point< 2, X >, Vector< X > > ∗ *convection*, Matrix< X > & *matrix*)

The computed matrix is stored in the Matrix referenced by matrix.

Compute the convection matrix for a 2D elliptic problem in a Spectral Element Space

Definition at line 21 of file computeconvectionmatrix.hpp.

5.2.2.5 template<class X > void SemSolver::Assembler::compute_diffusion_matrix (const SemSpace< 2, X > & *space*, const Function< Point< 2, X >, X > ∗ *diffusion*, Matrix< X > & *matrix*)

The computed matrix is stored in the Matrix referenced by matrix.

Compute the diffusion matrix for a 2D elliptic problem in a Spectral Element Space

Definition at line 21 of file computediffusionmatrix.hpp.

5.2.2.6 template<class X > void SemSolver::Assembler::compute_forcing_vector (const SemSpace< 2, X > & *space*, const Function< Point< 2, X >, X > ∗ *forcing*, Vector< X > & *vector*)

The computed matrix is stored in the Matrix referenced by matrix.

Compute the forcing vector for a 2D elliptic problem in a Spectral Element Space

Definition at line 19 of file computeforcingvector.hpp.

5.2.2.7 template<class X > void SemSolver::Assembler::compute_reaction_matrix (const SemSpace< 2, X > & *space*, Function< Point< 2, X >, X > const ∗ *reaction*, Matrix< double > & *matrix*)

The computed matrix is stored in the Matrix referenced by matrix.

Compute the reaction matrix for a 2D elliptic problem in a Spectral Element Space

Definition at line 19 of file computereactionmatrix.hpp.

5.3 SemSolver::IO Namespace Reference

Namespace for Input/Output operations on SemSolver Classes.

Classes

- class Archive

  Class for handling tar uncompressed archives.

Functions

- template< class X >
  bool read_boundary_conditions (QFile *file, BoundaryConditions< 2, X > &boundary_-conditions)

  Read 2D BoundaryConditions from file.

- template< class X >
  bool read_equation (QFile *file, Equation< 2, X > *&equation)

  Read 2D Equation from file.

- template< class X >
  bool read_geometry (QFile *file, SemGeometry< 2, X > &geometry)

  Read SemGeometry form file.

- bool write_geometry (QFile *pslg_file, QFile *domains_file, QFile *file)

  Write Geometry archive from pslg and subdomains files.

- QStringList next_non_empty_line_values (QTextStream &text_stream)

  Get list of values on next non empty line in a text stream skipping comments.

- template< class X >
  bool read_parameters (QFile *file, SemParameters< X > &parameters)

  Read SemParameters from file.

- template< class X >
  bool read_PSLG (QFile *file, PSLG< X > &pslg)
- template< class X >
  bool read_subdomains (QFile *file, Polygonation< 2, X > &sub_domains)

  Read subdomains Polygonation from file.

- template< class X >
  bool write_subdomains (Polygonation< 2, X > const &sub_domains, QFile *file)

  Write subdomains Polygonation to file.

- bool get_geometries_list_from_workspace (QFile *file, QStringList &geometries)

  Get list of geometries in workspace.

- bool get_equations_list_from_workspace (QFile *file, QStringList &equations)

  Get list of equations in workspace.

- bool get_boundary_conditions_list_from_workspace (QFile *file, QStringList &boundary_conditions)

  Get list of boundary conditions in workspace.

- bool get_parameters_list_from_workspace (QFile ∗file, QStringList &parameters)

  Get list of parameters in workspace.

- template< class X >
  bool get_geometry_from_workspace (QFile ∗file, QString const &name, SemGeometry< 2, X > &geometry)

  Read a geometry from workspace.

- template< class X >
  bool get_equation_from_workspace (QFile ∗file, QString const &name, Equation< 2, X > ∗&equation)

  Read an equation from workspace.

- template< class X >
  bool get_boundary_conditions_from_workspace (QFile ∗file, QString const &name, BoundaryConditions< 2, X > &bc)

  Read boundary conditions from workspace.

- template< class X >
  bool get_parameters_from_workspace (QFile ∗file, QString const &name, SemParameters< X > &bc)

  Read parameters from workspace.

- bool extract_file_from_workspace (QFile ∗workspace, QString const &name, QFile ∗file)

  Extract an entry from workspace.

- bool add_file_to_workspace (QFile ∗workspace, QString const &name, QFile ∗file)

  Add an entry to workspace.

- bool remove_file_from_workspace (QFile ∗workspace, QString const &name)

  Remove an entry from workspace.

5.3.1 Detailed Description

Namespace for Input/Output operations on SemSolver Classes.

5.3.2 Function Documentation

5.3.2.1 bool SemSolver::IO::add_file_to_workspace (QFile ∗ *workspace*, QString const & *name*, QFile ∗ *file*)

Add an entry to workspace.

5.3.2.2 bool SemSolver::IO::extract_file_from_workspace (QFile ∗ *workspace,* QString const & *name,* QFile ∗ *file*)

Extract an entry from workspace.

5.3.2.3 template<class X > bool SemSolver::IO::get_boundary_conditions_from_workspace (QFile ∗ *file,* QString const & *name,* BoundaryConditions< 2, X > & *bc*)

Read boundary conditions from workspace.

Definition at line 110 of file workspace.hpp.

5.3.2.4 bool SemSolver::IO::get_boundary_conditions_list_from_workspace (QFile ∗ *file,* QStringList & *boundary_conditions*)

Get list of boundary conditions in workspace.

5.3.2.5 template<class X > bool SemSolver::IO::get_equation_from_workspace (QFile ∗ *file,* QString const & *name,* Equation< 2, X > ∗& *equation*)

Read an equation from workspace.

Definition at line 90 of file workspace.hpp.

5.3.2.6 bool SemSolver::IO::get_equations_list_from_workspace (QFile ∗ *file,* QStringList & *equations*)

Get list of equations in workspace.

5.3.2.7 bool SemSolver::IO::get_geometries_list_from_workspace (QFile ∗ *file,* QStringList & *geometries*)

Get list of geometries in workspace.

5.3.2.8 template<class X > bool SemSolver::IO::get_geometry_from_workspace (QFile ∗ *file,* QString const & *name,* SemGeometry< 2, X > & *geometry*)

Read a geometry from workspace.

Definition at line 70 of file workspace.hpp.

5.3.2.9 template<class X > bool SemSolver::IO::get_parameters_from_workspace (QFile ∗ *file,* QString const & *name,* SemParameters< X > & *bc*)

Read parameters from workspace.

Definition at line 130 of file workspace.hpp.

5.3.2.10 bool SemSolver::IO::get_parameters_list_from_workspace (QFile * *file*, QStringList & *parameters*)

Get list of parameters in workspace.

5.3.2.11 QStringList SemSolver::IO::next_non_empty_line_values (QTextStream & *text_stream*)

Get list of values on next non empty line in a text stream skipping comments.

5.3.2.12 template<class X > bool SemSolver::IO::read_boundary_conditions (QFile * *file*, BoundaryConditions< 2, X > & *boundary_conditions*)

Read 2D BoundaryConditions from file.

Definition at line 23 of file IO/boundaryconditions.hpp.

5.3.2.13 template<class X > bool SemSolver::IO::read_equation (QFile * *file*, Equation< 2, X > *& *equation*)

Read 2D Equation from file.

Definition at line 24 of file IO/equation.hpp.

5.3.2.14 template<class X > bool SemSolver::IO::read_geometry (QFile * *file*, SemGeometry< 2, X > & *geometry*)

Read SemGeometry form file.

Definition at line 19 of file geometry.hpp.

5.3.2.15 template<class X > bool SemSolver::IO::read_parameters (QFile * *file*, SemParameters< X > & *parameters*)

Read SemParameters from file.

Definition at line 23 of file parameters.hpp.

5.3.2.16 template<class X > bool SemSolver::IO::read_PSLG (QFile * *file*, PSLG< X > & *pslg*)

Definition at line 15 of file IO/pslg.hpp.

5.3.2.17 template<class X > bool SemSolver::IO::read_subdomains (QFile * *file*, Polygonation< 2, X > & *sub_domains*)

Read subdomains Polygonation from file.

Definition at line 19 of file subdomains.hpp.

5.3.2.18 bool SemSolver::IO::remove_file_from_workspace (QFile ∗ *workspace,* QString const & *name*)

Remove an entry from workspace.

5.3.2.19 bool SemSolver::IO::write_geometry (QFile ∗ *pslg_file,* QFile ∗ *domains_file,* QFile ∗ *file*)

Write Geometry archive from pslg and subdomains files.

5.3.2.20 template< class X > bool SemSolver::IO::write_subdomains (Polygonation< 2, X > const & *sub_domains,* QFile ∗ *file*)

Write subdomains Polygonation to file.

Definition at line 60 of file subdomains.hpp.

5.4 SemSolver::PostProcessor Namespace Reference

Functions

- template< class X >
  void build_solution (const SemSpace< 2, X > &space, const Vector< X > &coefficients, Function< Point< 2, X >, X > ∗&solution)
- template< class X >
  void compute_plot_data (const SemSpace< 2, X > &space, const Vector< X > &u, Qwt3D::TripleField &data, Qwt3D::CellField &poly)
- template< class X >
  void compute_solution_hull (const SemSpace< 2, X > &space, const Vector< X > &coefficients, double &xmin, double &ymin, double &zmin, double &xmax, double &ymax, double &zmax)

5.4.1 Function Documentation

5.4.1.1 template< class X > void SemSolver::PostProcessor::build_solution (const SemSpace< 2, X > & *space,* const Vector< X > & *coefficients,* Function< Point< 2, X >, X > ∗& *solution*)

Definition at line 13 of file buildsolution.hpp.

5.4.1.2 template< class X > void SemSolver::PostProcessor::compute_plot_data (const SemSpace< 2, X > & *space,* const Vector< X > & *u,* Qwt3D::TripleField & *data,* Qwt3D::CellField & *poly*)

Definition at line 14 of file computeplotdata.hpp.

5.4.1.3 template< class X > void SemSolver::PostProcessor::compute_solution_hull (const SemSpace< 2, X > & *space*, const Vector< X > & *coefficients*, double & *xmin*, double & *ymin*, double & *zmin*, double & *xmax*, double & *ymax*, double & *zmax*)

Definition at line 12 of file computesolutionhull.hpp.

5.5 SemSolver::PreProcessor Namespace Reference

PreProcessor namespace.

Functions

- template<class X >
  bool compute_polygonation_from_pslg (const PSLG< X > &pslg, Polygonation< 2, X > &polygonation)
- template<class X >
  bool compute_vertices_sequences_from_pslg (PSLG< X > const &pslg, Sequences_-list &vertices_sequences)

  Get sequences of vertices representing the set of polygons given by a PSLG.
- template<class X >
  bool compute_polygon_with_holes_from_pslg (PSLG< X > const &pslg, PolygonWithHoles< 2, X > &polygon)

  Compute the polygon with holes given by a PSLG.

5.5.1 Detailed Description

PreProcessor namespace. This namespace provides algorithms for the constuction of the geometric structures associated to the problem from geometric information stored in PSLG format.

5.5.2 Function Documentation

5.5.2.1 template< class X > bool SemSolver::PreProcessor::compute_polygon_with_holes_-from_pslg (PSLG< X > const & *pslg*, PolygonWithHoles< 2, X > & *polygon*)

Compute the polygon with holes given by a PSLG.

Definition at line 108 of file computepolygonwithholesfrompslg.hpp.

5.5.2.2 template< class X > bool SemSolver::PreProcessor::compute_polygonation_from_pslg (const PSLG< X > & *pslg*, Polygonation< 2, X > & *polygonation*)

Compute a polygonation of a 2D geoemetry defined as a Planar Straight Line Graph

Definition at line 42 of file computepolygonationfrompslg.hpp.

5.5.2.3 template< class X > bool SemSolver::PreProcessor::compute_vertices_sequences_-from_pslg (PSLG< X > const & *pslg*, Sequences_list & *vertices_sequences*)

Get sequences of vertices representing the set of polygons given by a PSLG.

Definition at line 17 of file computepolygonwithholesfrompslg.hpp.

5.6 SemSolver::Solver Namespace Reference

Solver namespace.

Functions

- template< class X >
  bool cholesky_solve (Matrix< X > const &A, Vector< X > const &b, Vector< X > &x)
- template< class X >
  bool lu_solve (Matrix< X > const &A, Vector< X > const &b, Vector< X > &x)
- template< class X >
  bool qr_solve (Matrix< X > const &A, Vector< X > const &b, Vector< X > &x)

5.6.1 Detailed Description

Solver namespace. This namespace provides algorithms for solving an algebraic system A∗x=b

5.6.2 Function Documentation

5.6.2.1 template< class X > bool SemSolver::Solver::cholesky_solve (Matrix< X > const & *A*, Vector< X > const & *b*, Vector< X > & *x*)

Solve the algebraic system A∗x=b with Cholesky method

Parameters

A	must be a SPD matrix

b	constant term
x	Vector reference to the computed solution

Definition at line 24 of file choleskysolve.hpp.

5.6.2.2 template< class X > bool SemSolver::Solver::lu_solve (Matrix< X > const & *A*, Vector< X > const & *b*, Vector< X > & *x*)

Solve the algebraic system A∗x=b with LU factorization method

Parameters

A	must be a non singular matrix
b	constant term
x	Vector reference to the computed solution

Definition at line 24 of file lusolve.hpp.

5.6.2.3 template< class X > bool SemSolver::Solver::qr_solve (Matrix< X > const & *A*, Vector< X > const & *b*, Vector< X > & *x*)

Solve the algebraic system A∗x=b with QR factorization method

Parameters

A	must be a full rank matrix
b	constant term
x	Vector reference to the computed solution

Definition at line 24 of file qrsolve.hpp.

Chapter 6

Class Documentation

6.1 SemSolver::IO::Archive Class Reference

Class for handling tar uncompressed archives.

`#include <archive.hpp>`

Public Member Functions

- Archive (QFile ∗file)
- bool openRead ()

  Open archive in read mode.
- bool openWrite ()

  Open archive in write mode.
- bool closeRead ()

  Close archive in read mode.
- bool closeWrite ()

  Close archive in write mode.
- QStringList entries ()

  Get list of entries in archive.
- template<class T >
  bool addValue (T const &value, QString const &name)
- bool addFile (QFile ∗file)
- bool addFile (QFile ∗file, QString const &name)
- bool extractFile (QString const &name)
- bool extractFile (QString const &name, QFile ∗file)

6.1.1 Detailed Description

Class for handling tar uncompressed archives.

Definition at line 18 of file archive.hpp.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 SemSolver::IO::Archive::Archive (QFile * *file*)

Default constructor file pointer to archive file

6.1.3 Member Function Documentation

6.1.3.1 bool SemSolver::IO::Archive::addFile (QFile * *file*)

Add a new entry to archive

Parameters

file	File to be used as entry

6.1.3.2 bool SemSolver::IO::Archive::addFile (QFile * *file*, QString const & *name*)

Add a new entry to archive

Parameters

name	Name of the entry
file	File to be used as entry

6.1.3.3 template<class T > bool SemSolver::IO::Archive::addValue (T const & *value*, QString const & *name*) [inline]

Add a new entry to archive

Parameters

name	Name of the entry
value	Value of the entry

Definition at line 57 of file archive.hpp.

6.1.3.4 bool SemSolver::IO::Archive::closeRead ( )

Close archive in read mode.

6.1.3.5 bool SemSolver::IO::Archive::closeWrite ( )

Close archive in write mode.

6.1.3.6 QStringList SemSolver::IO::Archive::entries ( )

Get list of entries in archive.

6.1.3.7 bool SemSolver::IO::Archive::extractFile (QString const & *name*, QFile * *file*)

Get an entry

Parameters

file	Pointer to the file where to extract entry data
name	Name of the entry

6.1.3.8 bool SemSolver::IO::Archive::extractFile (QString const & *name*)

Get an entry

Parameters

name	Name of the entry

6.1.3.9 bool SemSolver::IO::Archive::openRead ( )

Open archive in read mode.

6.1.3.10 bool SemSolver::IO::Archive::openWrite ( )

Open archive in write mode.

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

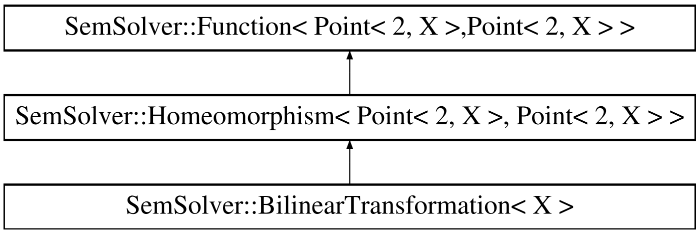
- archive.hpp

6.2 SemSolver::BilinearTransformation< X > Class Template Reference

Class representing a bilinear tranformation of 2D euclidean space.

`#include <bilineartransformation.hpp>`

Inheritance diagram for SemSolver::BilinearTransformation< X >:



Public Member Functions

- BilinearTransformation ()

  Construct the trivial transformation.

- BilinearTransformation (Point< 2, X > const &A, Point< 2, X > const &B, Point< 2, X > const &C, Point< 2, X > const &D)

  Construct the bilinear transformation that maps the quadrangle $\Omega = ABCD$ into the cononical square $\hat{\Omega} = (-1., 1.)^2$.

- BilinearTransformation (Polygon< 2, X > const &omega, X const &tolerance)

  Construct the bilinear transformation that maps the quadrangle $\Omega = ABCD$ into the cononical square $\hat{\Omega} = (-1., 1.)^2$.

- ~BilinearTransformation ()

  Destructor.

- void setOmega (Polygon< 2, X > const &omega)

  Set the quadrangle Omega to be mapped onto the canonical square.

- void setTolerance (X const &tolerance)

  Set the tolerance used in evaluating the inverse tranformation.

- Polygon< 2, X > const & omega () const

  Get the reference to Ω.

- X const & tolerance () const

  Get the tolerance used in evaluating the inverse tranformation.

- Point< 2, X > evaluate (Point< 2, X > const &point) const

  Evaluate the direct tranformation at a point.

- Point< 2, X > evaluateInverse (Point< 2, X > const &point) const

  Evaluate the inverse tranformation at a point.

- X evaluateJacobianDeterminant (Point< 2, X > const &point) const

  Evaluate the Jacobian determinant at a point.

- Matrix< double > evaluateTransposeInverseJacobian (Point< 2, X > const &point) const

  Evaluate the Transpose Inverse of the Jacobian matrix.

6.2.1 Detailed Description

template<class X> class SemSolver::BilinearTransformation< X >

Class representing a bilinear tranformation of 2D euclidean space.

Parameters

X	Must be a type for which operations +, -, $*$ and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of $*$

Definition at line 27 of file bilineartransformation.hpp.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 template<class X > SemSolver::BilinearTransformation< X >::BilinearTransformation ( )

Construct the trivial transformation.

Definition at line 76 of file bilineartransformation.hpp.

6.2.2.2 template<class X > SemSolver::BilinearTransformation< X >::BilinearTransformation (Point< 2, X > const & *A*, Point< 2, X > const & *B*, Point< 2, X > const & *C*, Point< 2, X > const & *D*)

Construct the bilinear transformation that maps the quadrangle $\Omega = ABCD$ into the cononical square $\hat{\Omega} = (-1., 1.)^2$.

It maps A into $(-1., -1.)$, B into $(1., -1.)$, C into $(1., 1.)$, and D into $(-1., 1.)$.

Definition at line 93 of file bilineartransformation.hpp.

6.2.2.3 template<class X > SemSolver::BilinearTransformation< X >::BilinearTransformation (Polygon< 2, X > const & *omega*, X const & *tolerance*)

Construct the bilinear transformation that maps the quadrangle $\Omega = ABCD$ into the cononical square $\hat{\Omega} = (-1., 1.)^2$.

It maps A into $(-1., -1.)$, B into $(1., -1.)$, C into $(1., 1.)$, and D into $(-1., 1.)$.

Parameters

omega	The polygon to be mapped onto $\hat{\Omega}$. It must be a simple convex counterclockwise-oriented quadrangle
tolerance	The value bolow which numbers are treates as zero. It must be non negative

Definition at line 126 of file bilineartransformation.hpp.

6.2.2.4 template<class X > SemSolver::BilinearTransformation< X >::~BilinearTransformation ( )

Destructor.

Definition at line 135 of file bilineartransformation.hpp.

6.2.3 Member Function Documentation

6.2.3.1 template<class X > SemSolver::Point< 2, X > SemSolver::BilinearTransformation< X >::evaluate (Point< 2, X > const & *point*) const

Evaluate the direct tranformation at a point.

Parameters

point	The point to be mapped

Definition at line 217 of file bilineartransformation.hpp.

6.2.3.2 template<class X > SemSolver::Point< 2, X > SemSolver::BilinearTransformation< X >::evaluateInverse (Point< 2, X > const & *point*) const

Evaluate the inverse tranformation at a point.

It uses different algorithms if Ω is a parallelogram a trapezoid or a more generic quadrangle.

Parameters

point	The point to be mapped

Definition at line 230 of file bilineartransformation.hpp.

6.2.3.3 template<class X > X SemSolver::BilinearTransformation< X >::evaluateJacobianDeterminant (Point< 2, X > const & *point*) const [inline]

Evaluate the Jacobian determinant at a point.

Parameters

point	The point where to evaluate the Jacobian

Definition at line 273 of file bilineartransformation.hpp.

6.2.3.4 template<class X > SemSolver::Matrix< double > SemSolver::BilinearTransformation< X >::evaluateTransposeInverseJacobian (Point< 2, X > const & *point*) const

Evaluate the Transpose Inverse of the Jacobian matrix.

Parameters

point	The point where to evaluate the Jacobian

Definition at line 283 of file bilineartransformation.hpp.

6.2.3.5 template<class X > SemSolver::Polygon< 2, X > const & SemSolver::BilinearTransformation< X >::omega ( ) const [inline]

Get the reference to Ω.

Returns

The quadrangle transformed onto $\hat{\Omega}$

Definition at line 200 of file bilineartransformation.hpp.

6.2.3.6 template<class X > void SemSolver::BilinearTransformation< X >::setOmega (Polygon< 2, X > const & *omega*)

Set the quadrangle *Omega* to be mapped onto the canonical square.

Parameters

omega	The polygon to be mapped onto $\hat{\Omega}$. It must be a simple convex counterclockwise-oriented quadrangle

Definition at line 143 of file bilineartransformation.hpp.

6.2.3.7 template<class X > void SemSolver::BilinearTransformation< X >::setTolerance (X const & *tolerance*)

Set the tolerance used in evaluating the inverse tranformation.

Parameters

tolerance	The value bolow which numbers are treates as zero. It must be non negative

Definition at line 187 of file bilineartransformation.hpp.

6.2.3.8 template<class X > X const & SemSolver::BilinearTransformation< X >::tolerance ( ) const [inline]

Get the tolerance used in evaluating the inverse tranformation.

Returns

Reference to the tolerance

Definition at line 208 of file bilineartransformation.hpp.

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

- bilineartransformation.hpp

6.3 SemSolver::BoundaryConditions< d, X > Class Template Reference

Class for handling boundary conditions on a SemGeometry.

`#include <boundaryconditions.hpp>`

6.3.1 Detailed Description

template<int d, class X> class SemSolver::BoundaryConditions< d, X >

Class for handling boundary conditions on a SemGeometry.

Parameters

d	Dimension of the space on which the geometry lives
X	Must be a type for which operations +, -, $*$ and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of $*$.

Definition at line 31 of file boundaryconditions.hpp.

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

- boundaryconditions.hpp

6.4 SemSolver::BoundaryConditions< 2, X > Class Template Reference

Class for handling boundary conditions on a 2D SemGeometry.

`#include <boundaryconditions.hpp>`

Public Types

- enum Type { UNDEFINED, DIRICHLET, NEUMANN, ROBIN }

  Enumeration of the available boundary condition types.

- typedef Function< Point< 2, X >, X > const ∗ FunctionPtr
- typedef std::map< int, FunctionPtr > FunctionsMap
- typedef std::map< int, Type > TypesMap
- typedef FunctionsMap::const_iterator FunctionConstIterator

Public Member Functions

- BoundaryConditions ()

  Construct empty boundary conditions for a specific geometry.

- BoundaryConditions (TypesMap const &types, FunctionsMap const ∗g, FunctionsMap const ∗h, FunctionsMap const ∗gamma, FunctionsMap const ∗r)

  Construct specific boundary conditions for a specific geometry.

- template<class Y >
  BoundaryConditions (BoundaryConditions< 2, Y > const &conditions)

  Copy constructor.

- int conditions () const

  Get the number of boundary conditions.

- Type const & borderType (int const &index) const

  Get the condition type on a border.

- FunctionPtr dirichletData (int const &index) const

  Get the Dirichlet data g on a border.

- FunctionPtr neumannData (int const &index) const

  Get the Neumann data h on a border.

- FunctionPtr robinCoefficient (int const &index) const

  Get the Robin coefficient γ on a border.

- FunctionPtr robinData (int const &index) const

  Get the Robin data r on a border.

- void setBorder (int const &index, Type const &type, FunctionPtr const f1=0, FunctionPtr const f2=0)

  Set condition on a border.

- QStringList labels () const

  Get the list of boundary labels.

- QStringList mmls () const

  Get the list of boundary conditions in Mathematical Markup Language format.

- void clear ()

  Free boundary conditions' content.

6.4.1 Detailed Description

template< class X> class SemSolver::BoundaryConditions< 2, X >

Class for handling boundary conditions on a 2D SemGeometry.

Parameters

X	Must be a type for which operations +, -, $*$ and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of $*$.

Definition at line 42 of file boundaryconditions.hpp.

6.4.2 Member Typedef Documentation

6.4.2.1 template< class X > typedef FunctionsMap::const_iterator SemSolver::BoundaryConditions< 2, X >::FunctionConstIterator

Definition at line 63 of file boundaryconditions.hpp.

6.4.2.2 template<class X > typedef Function< Point<2,X>, X > const* SemSolver::BoundaryConditions< 2, X >::FunctionPtr

Definition at line 60 of file boundaryconditions.hpp.

6.4.2.3 template<class X > typedef std::map<int, FunctionPtr> SemSolver::BoundaryConditions< 2, X >::FunctionsMap

Definition at line 61 of file boundaryconditions.hpp.

6.4.2.4 template<class X > typedef std::map<int, Type > SemSolver::BoundaryConditions< 2, X >::TypesMap

Definition at line 62 of file boundaryconditions.hpp.

6.4.3 Member Enumeration Documentation

6.4.3.1 template<class X > enum SemSolver::BoundaryConditions< 2, X >::Type

Enumeration of the available boundary condition types.

Enumerator:

UNDEFINED

DIRICHLET Dirichlet condition $u = g$ on border Γ_i.

NEUMANN Neumann condition $\nabla u \cdot \mathbf{n} = h$ on border Γ_i.

ROBIN Robin condition $u + \gamma \nabla u \cdot \mathbf{n} = r$ on border Γ_i.

Definition at line 46 of file boundaryconditions.hpp.

6.4.4 Constructor & Destructor Documentation

6.4.4.1 template<class X > SemSolver::BoundaryConditions< 2, X >::BoundaryConditions ( ) [inline]

Construct empty boundary conditions for a specific geometry.

Definition at line 115 of file boundaryconditions.hpp.

6.4.4.2 template<class X > SemSolver::BoundaryConditions< 2, X >::BoundaryConditions (TypesMap const & *types*, FunctionsMap const * *g*, FunctionsMap const * *h*, FunctionsMap const * *gamma*, FunctionsMap const * *r*)

Construct specific boundary conditions for a specific geometry.

Parameters

types	Vector of the condition types for each geometry border
g	Border datas for Dirichlet conditions
h	Border datas for Neumann conditions
gamma	Border coefficient for Robin conditions
r	Border datas for Robin conditions

Definition at line 126 of file boundaryconditions.hpp.

6.4.4.3 template< class X > template< class Y > SemSolver::BoundaryConditions< 2, X >::BoundaryConditions (BoundaryConditions< 2, Y > const & *conditions*)

Copy constructor.

Parameters

conditions	Boundary conditions to be copied

Definition at line 175 of file boundaryconditions.hpp.

6.4.5 Member Function Documentation

6.4.5.1 template< class X > SemSolver::BoundaryConditions< 2, X >::Type const & SemSolver::BoundaryConditions< 2, X >::borderType (int const & *index*) const [inline]

Get the condition type on a border.

Parameters

index	Of the border

Returns

The border Type

Definition at line 198 of file boundaryconditions.hpp.

6.4.5.2 template< class X > void SemSolver::BoundaryConditions< 2, X >::clear ( )

Free boundary conditions' content.

Definition at line 390 of file boundaryconditions.hpp.

6.4.5.3 template< class X > int SemSolver::BoundaryConditions< 2, X >::conditions ( ) const [inline]

Get the number of boundary conditions.

Returns

The conditions number

Definition at line 188 of file boundaryconditions.hpp.

6.4.5.4 template<class X > SemSolver::Function< SemSolver::Point< 2, X >, X > const * SemSolver::BoundaryConditions< 2, X >::dirichletData (int const & *index*) const [inline]

Get the Dirichlet data g on a border.

Returns

Pointer to a scalar function

Parameters

index	of the border

Definition at line 213 of file boundaryconditions.hpp.

6.4.5.5 template<class X > QStringList SemSolver::BoundaryConditions< 2, X >::labels ( ) const

Get the list of boundary labels.

Returns

QStringList containing one QString for each edge given by segment id number preceded by an 'S'

Definition at line 325 of file boundaryconditions.hpp.

6.4.5.6 template<class X > QStringList SemSolver::BoundaryConditions< 2, X >::mmls ( ) const

Get the list of boundary conditions in Mathematical Markup Language format.

Returns

QStringList containing one QString for each boundary condition in MathML notation

Definition at line 337 of file boundaryconditions.hpp.

6.4.5.7 template< class X > SemSolver::Function< SemSolver::Point< 2, X >, X > const * SemSolver::BoundaryConditions< 2, X >::neumannData (int const & *index*) const [inline]

Get the Neumann data h on a border.

Parameters

index	of the border

Returns

Pointer to a scalar function

Definition at line 229 of file boundaryconditions.hpp.

6.4.5.8 template< class X > SemSolver::Function< SemSolver::Point< 2, X >, X > const * SemSolver::BoundaryConditions< 2, X >::robinCoefficient (int const & *index*) const [inline]

Get the Robin coefficient γ on a border.

Parameters

index	of the border

Returns

Pointer to a scalar function

Definition at line 245 of file boundaryconditions.hpp.

6.4.5.9 template< class X > SemSolver::Function< SemSolver::Point< 2, X >, X > const * SemSolver::BoundaryConditions< 2, X >::robinData (int const & *index*) const [inline]

Get the Robin data r on a border.

Parameters

index	of the border

Returns

Pointer to a scalar function

Definition at line 261 of file boundaryconditions.hpp.

6.4.5.10 template< class X > void SemSolver::BoundaryConditions< 2, X >::setBorder (int const & *index*, Type const & *type*, FunctionPtr const *f1* = 0, FunctionPtr const *f2* = 0)

Set condition on a border.

Parameters

index	Index of the border
type	Type of the border
f1	Pointer to scalar function, it is used as border data for Dirichlet and Neumann conditions and as coefficient for Robin one, it is ignored otherwise
f2	Pointer to scalar function, it is used as border data for Robin conditions, it is ignored otherwise

Definition at line 280 of file boundaryconditions.hpp.

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

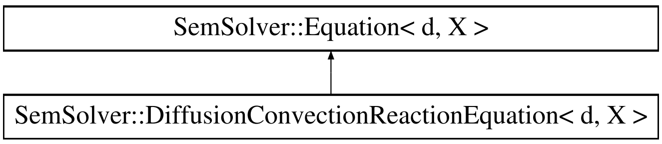
- boundaryconditions.hpp

6.5 SemSolver::DiffusionConvectionReactionEquation< d, X > Class Template Reference

Class for handling Diffusion-Convection-Reaction steady equation.

```
#include <diffusionconvectionreactionequation.hpp>
```

Inheritance diagram for SemSolver::DiffusionConvectionReactionEquation< d, X >:



Public Types

- typedef Equation< d, X >::Type Type

    Enumeration of the avilable equation types.

Public Member Functions

- DiffusionConvectionReactionEquation ()

    Default constructor Construct an emty equation.

- ~DiffusionConvectionReactionEquation ()
  Destructor.
- Type type () const
  Get the equation type.
- QString mml () const
  Get equation in Mathematical Markup Language notation.
- void setDiffusion (Function< Point< d, X >, X > ∗diffusion)
  Set the diffusion coefficient.
- void setConvection (Function< Point< d, X >, Vector< X > > ∗convection)
  Set the convection coefficient.
- void setReaction (Function< Point< d, X >, X > ∗reaction)
  Set the reaction coefficient.
- void setForcing (Function< Point< d, X >, X > ∗forcing)
  Set the forcing term.
- Function< Point< d, X >, X > ∗ diffusion () const
  Get the diffusion coefficient.
- Function< Point< d, X >, Vector< X > > ∗ convection () const
  Get the convection coefficient.
- Function< Point< d, X >, X > ∗ reaction () const
  Get the reaction coefficient.
- Function< Point< d, X >, X > ∗ forcing () const
  Get the forcing term.

6.5.1 Detailed Description

template<int d, class X> class SemSolver::DiffusionConvectionReactionEquation< d, X >

Class for handling Diffusion-Convection-Reaction steady equation.

Parameters

d	Dimension of the space on which the equation is defined
X	Must be a type for which operations +, -, ∗ and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of ∗.

Definition at line 28 of file diffusionconvectionreactionequation.hpp.

6.5.2 Member Typedef Documentation

6.5.2.1 template<int d, class X > typedef Equation<d, X>::Type SemSolver::DiffusionConvectionReactionEquation< d, X >::Type

Enumeration of the avilable equation types.

Reimplemented from SemSolver::Equation< d, X >.

Definition at line 39 of file diffusionconvectionreactionequation.hpp.

6.5.3 Constructor & Destructor Documentation

6.5.3.1 template<int d, class X > SemSolver::DiffusionConvectionReactionEquation< d, X >::DiffusionConvectionReactionEquation ( ) [inline]

Default constructor Construct an emty equation.

Definition at line 43 of file diffusionconvectionreactionequation.hpp.

6.5.3.2 template<int d, class X > SemSolver::DiffusionConvectionReactionEquation< d, X >::~DiffusionConvectionReactionEquation ( ) [inline]

Destructor.

Definition at line 52 of file diffusionconvectionreactionequation.hpp.

6.5.4 Member Function Documentation

6.5.4.1 template<int d, class X > Function< Point<d, X>, Vector<X> >* SemSolver::DiffusionConvectionReactionEquation< d, X >::convection ( ) const [inline]

Get the convection coefficient.

Returns

Pointer to function from X^d to X

Definition at line 110 of file diffusionconvectionreactionequation.hpp.

6.5.4.2 template<int d, class X > Function< Point<d, X>, X >* SemSolver::DiffusionConvectionReactionEquation< d, X >::diffusion ( ) const [inline]

Get the diffusion coefficient.

Returns

Pointer to function from X^d to X

Definition at line 103 of file diffusionconvectionreactionequation.hpp.

6.5.4.3 template<int d, class X > Function< Point<d, X>, X >* SemSolver::DiffusionConvectionReactionEquation< d, X >::forcing ( ) const [inline]

Get the forcing term.

Returns

Pointer to function from X^d to X

Definition at line 124 of file diffusionconvectionreactionequation.hpp.

6.5.4.4 template<int d, class X > QString SemSolver::DiffusionConvectionReactionEquation< d, X >::mml ( ) const [virtual]

Get equation in Mathematical Markup Language notation.

Returns

QString of equation in MathML format

Reimplemented from SemSolver::Equation< d, X >.

6.5.4.5 template<int d, class X > Function< Point<d, X>, X >* SemSolver::DiffusionConvectionReactionEquation< d, X >::reaction ( ) const [inline]

Get the reaction coefficient.

Returns

Pointer to function from X^d to X

Definition at line 117 of file diffusionconvectionreactionequation.hpp.

6.5.4.6 template<int d, class X > void SemSolver::DiffusionConvectionReactionEquation< d, X >::setConvection (Function< Point< d, X >, Vector< X > > * *convection*) [inline]

Set the convection coefficient.

Parameters

convection	Pointer to function from X^d to X

Definition at line 79 of file diffusionconvectionreactionequation.hpp.

6.5.4.7 template<int d, class X > void SemSolver::DiffusionConvectionReactionEquation< d, X >::setDiffusion (Function< Point< d, X >, X > * *diffusion*) [inline]

Set the diffusion coefficient.

Parameters

diffusion	Pointer to function from X^d to X

Definition at line 71 of file diffusionconvectionreactionequation.hpp.

6.5.4.8 template<int d, class X > void SemSolver::DiffusionConvectionReactionEquation< d, X >::setForcing (Function< Point< d, X >, X > * *forcing*) [inline]

Set the forcing term.

Parameters

forcing	Pointer to function from X^d to X

Definition at line 95 of file diffusionconvectionreactionequation.hpp.

6.5.4.9 template<int d, class X > void SemSolver::DiffusionConvectionReactionEquation< d, X >::setReaction (Function< Point< d, X >, X > * *reaction*) [inline]

Set the reaction coefficient.

Parameters

reaction	Pointer to function from X^d to X

Definition at line 87 of file diffusionconvectionreactionequation.hpp.

6.5.4.10 template<int d, class X > Type SemSolver::DiffusionConvectionReactionEquation< d, X >::type ( ) const [inline, virtual]

Get the equation type.

Returns

DIFFUSION_CONVECTION_REACTION Equation::Type

Reimplemented from SemSolver::Equation< d, X >.

Definition at line 62 of file diffusionconvectionreactionequation.hpp.

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

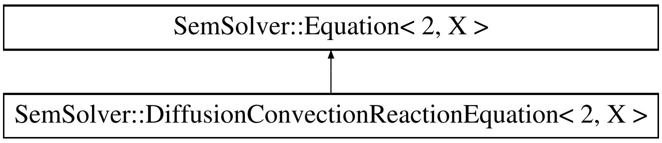
- diffusionconvectionreactionequation.hpp

6.6 SemSolver::DiffusionConvectionReactionEquation< 2, X > Class Template Reference

Class for handling Diffusion-Convection-Reaction steady equation on 2D spaces.

`#include <diffusionconvectionreactionequation.hpp>`

Inheritance diagram for SemSolver::DiffusionConvectionReactionEquation< 2, X >:



Public Types

- typedef Equation< 2, X >::Type Type

  Enumeration of the avilable equation types.

Public Member Functions

- DiffusionConvectionReactionEquation ()

  Default constructor Construct an emty equation.

- ~DiffusionConvectionReactionEquation ()

  Destructor.

- Type type () const

  Get the equation type.

- QString mml () const

  Get the equation in Mathematical Markup Languace fformat.

- void setDiffusion (Function< Point< 2, X >, X > ∗diffusion)

  Set the diffusion coefficient.

- void setConvection (Function< Point< 2, X >, Vector< X > > ∗convection)

Set the convection coefficient.

- void setReaction (Function< Point< 2, X >, X > *reaction)

  Set the reaction coefficient.

- void setForcing (Function< Point< 2, X >, X > *forcing)

  Set the forcing term.

- Function< Point< 2, X >, X > * diffusion () const

  Get the diffusion coefficient.

- Function< Point< 2, X >, Vector< X > > * convection () const

  Get the convection coefficient.

- Function< Point< 2, X >, X > * reaction () const

  Get the reaction coefficient.

- Function< Point< 2, X >, X > * forcing () const

  Get the forcing term.

6.6.1 Detailed Description

template<class X> class SemSolver::DiffusionConvectionReactionEquation< 2, X >

Class for handling Diffusion-Convection-Reaction steady equation on 2D spaces.

Parameters

X	Must be a type for which operations +, -, * and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of *.

Definition at line 138 of file diffusionconvectionreactionequation.hpp.

6.6.2 Member Typedef Documentation

6.6.2.1 template<class X > typedef Equation<2, X>::Type SemSolver::DiffusionConvectionReactionEquation< 2, X >::Type

Enumeration of the avilable equation types.

Reimplemented from SemSolver::Equation< 2, X >.

Definition at line 149 of file diffusionconvectionreactionequation.hpp.

6.6.3 Constructor & Destructor Documentation

6.6.3.1 template< class X > SemSolver::DiffusionConvectionReactionEquation< 2, X >::DiffusionConvectionReactionEquation ( ) [inline]

Default constructor Construct an emty equation.

Definition at line 153 of file diffusionconvectionreactionequation.hpp.

6.6.3.2 template< class X > SemSolver::DiffusionConvectionReactionEquation< 2, X >::~DiffusionConvectionReactionEquation ( ) [inline]

Destructor.

Definition at line 162 of file diffusionconvectionreactionequation.hpp.

6.6.4 Member Function Documentation

6.6.4.1 template< class X > Function< Point<2, X>, Vector<X> >* SemSolver::DiffusionConvectionReactionEquation< 2, X >::convection ( ) const [inline]

Get the convection coefficient.

Returns

Pointer to function from X^d to X

Definition at line 282 of file diffusionconvectionreactionequation.hpp.

6.6.4.2 template< class X > Function< Point<2, X>, X >* SemSolver::DiffusionConvectionReactionEquation< 2, X >::diffusion ( ) const [inline]

Get the diffusion coefficient.

Returns

Pointer to function from X^d to X

Definition at line 275 of file diffusionconvectionreactionequation.hpp.

6.6.4.3 template< class X > Function< Point<2, X>, X >* SemSolver::DiffusionConvectionReactionEquation< 2, X >::forcing ( ) const [inline]

Get the forcing term.

Returns

Pointer to function from X^d to X

Definition at line 296 of file diffusionconvectionreactionequation.hpp.

6.6.4.4 template<class X > QString SemSolver::DiffusionConvectionReactionEquation< 2, X >::mml ( ) const [inline, virtual]

Get the equation in Mathematical Markup Languace fformat.

Returns

QString of the equation in MathML notation

Reimplemented from SemSolver::Equation< 2, X >.

Definition at line 179 of file diffusionconvectionreactionequation.hpp.

6.6.4.5 template<class X > Function< Point<2, X>, X >* SemSolver::DiffusionConvectionReactionEquation< 2, X >::reaction ( ) const [inline]

Get the reaction coefficient.

Returns

Pointer to function from X^d to X

Definition at line 289 of file diffusionconvectionreactionequation.hpp.

6.6.4.6 template<class X > void SemSolver::DiffusionConvectionReactionEquation< 2, X >::setConvection (Function< Point< 2, X >, Vector< X > > * *convection*) [inline]

Set the convection coefficient.

Parameters

convection	Pointer to function from X^d to X

Definition at line 251 of file diffusionconvectionreactionequation.hpp.

6.6.4.7 template<class X > void SemSolver::DiffusionConvectionReactionEquation< 2, X >::setDiffusion (Function< Point< 2, X >, X > * *diffusion*) [inline]

Set the diffusion coefficient.

Parameters

diffusion	Pointer to function from X^d to X

Definition at line 243 of file diffusionconvectionreactionequation.hpp.

6.6.4.8 template< class X > void SemSolver::DiffusionConvectionReactionEquation< 2, X >::setForcing (Function< Point< 2, X >, X > * *forcing*) [inline]

Set the forcing term.

Parameters

forcing	Pointer to function from X^d to X

Definition at line 267 of file diffusionconvectionreactionequation.hpp.

6.6.4.9 template< class X > void SemSolver::DiffusionConvectionReactionEquation< 2, X >::setReaction (Function< Point< 2, X >, X > * *reaction*) [inline]

Set the reaction coefficient.

Parameters

reaction	Pointer to function from X^d to X

Definition at line 259 of file diffusionconvectionreactionequation.hpp.

6.6.4.10 template< class X > Type SemSolver::DiffusionConvectionReactionEquation< 2, X >::type ( ) const [inline, virtual]

Get the equation type.

Returns

DIFFUSION_CONVECTION_REACTION Equation::Type

Reimplemented from SemSolver::Equation< 2, X >.

Definition at line 172 of file diffusionconvectionreactionequation.hpp.

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

- diffusionconvectionreactionequation.hpp

6.7 SemSolver::HilbertSpace< Function, X >::Element Class Reference

Class for handling space elements as Fourier coefficients.

`#include <hilbertspace.hpp>`

Public Member Functions

- Element (const HilbertSpace< Function, X > *space)

  Constructor.

- virtual ~Element ()

  Destructor.

6.7.1 Detailed Description

template<class Function, class X> class SemSolver::HilbertSpace< Function, X >::Element

Class for handling space elements as Fourier coefficients.

Definition at line 21 of file hilbertspace.hpp.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 template<class Function, class X> SemSolver::HilbertSpace< Function, X >::Element::Element (const HilbertSpace< Function, X > * *space*) `[inline]`

Constructor.

Definition at line 28 of file hilbertspace.hpp.

6.7.2.2 template<class Function, class X> virtual SemSolver::HilbertSpace< Function, X >::Element::~Element ( ) `[inline, virtual]`

Destructor.

Definition at line 31 of file hilbertspace.hpp.

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

- hilbertspace.hpp

6.8 SemSolver::Polygonation< 2, X >::Element Class Reference

A Polygonation element.

`#include <polygonation.hpp>`

Public Member Functions

- Element ()

  Default constructor.

- Element (Polygon< 2, X > const &geometry, std::vector< int > const &neighbours)

  Construct an Element from a given Polygon and Neighbours list.

- Polygon< 2, X > const & geometry () const

  Get the element geometry return The Element's Polygon.

- unsigned vertexPosition (Point< 2, X > const &vertex) const

  Get the position of a vertex.

- void clear ()

  Clear Element's content.

- Point< 2, X > vertex (int const &index) const

  Get a vertex of an Element.

- int size () const

  Get an Elemnt size.

- Polygon< 2, X >::Vertex_const_iterator verticesBegin () const

  Get iterator to first vertex.

- Polygon< 2, X >::Vertex_const_iterator verticesEnd () const

  Get iterator to last vertex.

- std::vector< int >::const_iterator neighboursBegin () const

  Get iterator to first neghbour id.

- std::vector< int >::const_iterator neighboursEnd () const

  Get iterator to last neghbour id.

- void setGeometry (Point< 2, X > const ∗first, Point< 2, X > const ∗last)

  Set Element's geometry from a Point's sequence.

- void setNeighbour (const unsigned &index, int const &id)

  Set a neighbour id.

- int neighbour (const unsigned &index) const

  Get a neighbour id.

- bool contains (Point< 2, X > const &point) const

  Test if a point lies on the Element.

6.8.1 Detailed Description

template<class X> class SemSolver::Polygonation< 2, X >::Element

A Polygonation element. It can be a triangle, a quadrangle, or any Polygon. It also stores information about its neighbours

Definition at line 44 of file polygonation.hpp.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 template<class X > SemSolver::Polygonation< 2, X >::Element::Element ( ) [inline]

Default constructor.

Definition at line 269 of file polygonation.hpp.

6.8.2.2 template<class X > SemSolver::Polygonation< 2, X >::Element::Element (Polygon< 2, X > const & *geometry*, std::vector< int > const & *neighbours*) [inline]

Construct an Element from a given Polygon and Neighbours list.

Parameters

geometry	The geometry of the element
neighbours	Vector of neighbour element ids in counterclokwise order

Definition at line 273 of file polygonation.hpp.

6.8.3 Member Function Documentation

6.8.3.1 template<class X > void SemSolver::Polygonation< 2, X >::Element::clear ( ) [inline]

Clear Element's content.

Definition at line 302 of file polygonation.hpp.

6.8.3.2 template<class X > bool SemSolver::Polygonation< 2, X >::Element::contains (Point< 2, X > const & *point*) const [inline]

Test if a point lies on the Element.

Parameters

point	The Point to test with

Returns

The test result

Definition at line 382 of file polygonation.hpp.

6.8.3.3 template< class X > SemSolver::Polygon< 2, X > const & SemSolver::Polygonation< 2, X >::Element::geometry ( ) const [inline]

Get the element geometry return The Element's Polygon.

Definition at line 281 of file polygonation.hpp.

6.8.3.4 template< class X > int SemSolver::Polygonation< 2, X >::Element::neighbour (const unsigned & *index*) const [inline]

Get a neighbour id.

Parameters

index	Posiition of the neighbour

Returns

Id of the neighbour

Definition at line 370 of file polygonation.hpp.

6.8.3.5 template< class X > std::vector< int >::const_iterator SemSolver::Polygonation< 2, X >::Element::neighboursBegin ( ) const [inline]

Get iterator to first neghbour id.

Returns

Constant iterator

Definition at line 337 of file polygonation.hpp.

6.8.3.6 template< class X > std::vector< int >::const_iterator SemSolver::Polygonation< 2, X >::Element::neighboursEnd ( ) const [inline]

Get iterator to last neghbour id.

Returns

Constant iterator

Definition at line 344 of file polygonation.hpp.

6.8.3.7 template<class X > void SemSolver::Polygonation< 2, X >::Element::setGeometry (Point< 2, X > const * *first*, Point< 2, X > const * *last*) [inline]

Set Element's geometry from a Point's sequence.

Parameters

first	Pointer to the first Point of the sequence
last	Pointer to the last Point of the sequence

Definition at line 350 of file polygonation.hpp.

6.8.3.8 template<class X > void SemSolver::Polygonation< 2, X >::Element::setNeighbour (const unsigned & *index*, int const & *id*) [inline]

Set a neighbour id.

Parameters

index	Posiition of the neighbour
id	Id of the neighbour

Definition at line 358 of file polygonation.hpp.

6.8.3.9 template<class X > int SemSolver::Polygonation< 2, X >::Element::size ( ) const [inline]

Get an Elemnt size.

Returns

The number of vertices

Definition at line 316 of file polygonation.hpp.

6.8.3.10 template<class X > SemSolver::Point< 2, X > SemSolver::Polygonation< 2, X >::Element::vertex (int const & *index*) const [inline]

Get a vertex of an Element.

Parameters

index	The vertex position

Returns

The vertex

Definition at line 309 of file polygonation.hpp.

6.8.3.11 template< class X > unsigned SemSolver::Polygonation< 2, X >::Element::vertexPosition (Point< 2, X > const & *vertex*) const

Get the position of a vertex.

Parameters

vertex	The Point to search

Returns

The position (starting at 0) of the vertex if there is, -1 otherwise

Definition at line 288 of file polygonation.hpp.

6.8.3.12 template< class X > SemSolver::Polygon< 2, X >::Vertex_const_iterator SemSolver::Polygonation< 2, X >::Element::verticesBegin ( ) const [inline]

Get iterator to first vertex.

Returns

Constant iterator

Definition at line 323 of file polygonation.hpp.

6.8.3.13 template< class X > SemSolver::Polygon< 2, X >::Vertex_const_iterator SemSolver::Polygonation< 2, X >::Element::verticesEnd ( ) const [inline]

Get iterator to last vertex.

Returns

Constant iterator

Definition at line 330 of file polygonation.hpp.

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

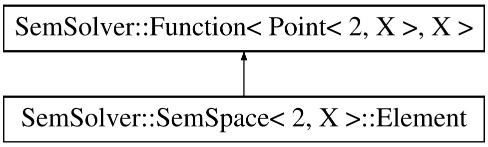
- polygonation.hpp

6.9 SemSolver::SemSpace< 2, X >::Element Class Reference

Class for handling members of the space as Fourier coefficients.

`#include <semspace.hpp>`

Inheritance diagram for SemSolver::SemSpace< 2, X >::Element:



Public Member Functions

- Element (SemSpace const ∗space, Vector< X > const &coefficients)

  Construct space element from Fourier coefficients.

- ∼Element ()
- X evaluate (Point< 2, X > const &x) const

  Compute element value at a point.

6.9.1 Detailed Description

template<class X> class SemSolver::SemSpace< 2, X >::Element

Class for handling members of the space as Fourier coefficients.

Definition at line 139 of file semspace.hpp.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 template<class X > SemSolver::SemSpace< 2, X >::Element::Element (SemSpace< 2, X > const ∗ *space*, Vector< X > const & *coefficients*) [inline]

Construct space element from Fourier coefficients.

Definition at line 148 of file semspace.hpp.

6.9.2.2 template<class X > SemSolver::SemSpace< 2, X >::Element::∼Element ( ) [inline]

Definition at line 156 of file semspace.hpp.

6.9.3 Member Function Documentation

6.9.3.1 template<class X > X SemSolver::SemSpace< 2, X >::Element::evaluate (Point< 2, X > const & *x*) const [inline, virtual]

Compute element value at a point.

Reimplemented from SemSolver::Function< Point< 2, X >, X >.

Definition at line 159 of file semspace.hpp.

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

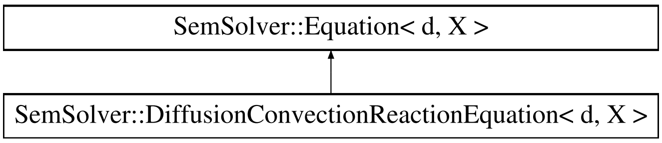
- semspace.hpp

6.10 SemSolver::Equation< d, X > Class Template Reference

Virtual class for handling general equation.

`#include <equation.hpp>`

Inheritance diagram for SemSolver::Equation< d, X >:



Public Types

- enum Type { NONE, DIFFUSION_CONVECTION_REACTION }

  Enumeration of the avilable equation types.

Public Member Functions

- Equation ()

  Default constructor.

- virtual ~Equation ()

  Destructor.

- virtual Type type () const

  Get equation type.

- virtual QString mml () const

  Get equation in Mathematical Markup Language notation.

6.10.1 Detailed Description

template<int d, class X> class SemSolver::Equation< d, X >

Virtual class for handling general equation.

Parameters

d	Dimension of the space on which the equation is defined
X	Must be a type for which operations +, -, $*$ and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of $*$.

Definition at line 24 of file equation.hpp.

6.10.2 Member Enumeration Documentation

6.10.2.1 template<int d, class X> enum SemSolver::Equation::Type

Enumeration of the avilable equation types.

Enumerator:

NONE

DIFFUSION_CONVECTION_REACTION Diffusion-Convection-Reaction steady equation.

Reimplemented in SemSolver::DiffusionConvectionReactionEquation< d, X >, and SemSolver::DiffusionConvectionReactionEquation< 2, X >.

Definition at line 28 of file equation.hpp.

6.10.3 Constructor & Destructor Documentation

6.10.3.1 template<int d, class X> SemSolver::Equation< d, X >::Equation ( ) [inline]

Default constructor.

Definition at line 36 of file equation.hpp.

6.10.3.2 template<int d, class X> virtual SemSolver::Equation< d, X >::~Equation ( ) [inline, virtual]

Destructor.

Definition at line 39 of file equation.hpp.

6.10.4 Member Function Documentation

6.10.4.1 template<int d, class X> virtual QString SemSolver::Equation< d, X >::mml ( ) const [inline, virtual]

Get equation in Mathematical Markup Language notation.

Returns

QString of equation in MathML format

Reimplemented in SemSolver::DiffusionConvectionReactionEquation< d, X >, and SemSolver::DiffusionConvectionReactionEquation< 2, X >.

Definition at line 46 of file equation.hpp.

6.10.4.2 template<int d, class X> virtual Type SemSolver::Equation< d, X >::type ( ) const [inline, virtual]

Get equation type.

Reimplemented in SemSolver::DiffusionConvectionReactionEquation< d, X >, and SemSolver::DiffusionConvectionReactionEquation< 2, X >.

Definition at line 42 of file equation.hpp.

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

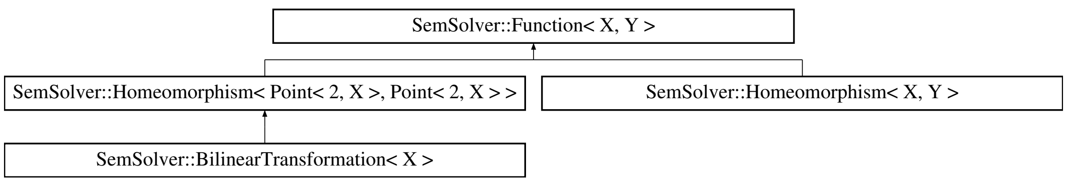
- equation.hpp

6.11 SemSolver::Function< X, Y > Class Template Reference

Prototype class for mathematical functions : $X \rightarrow Y$.

```
#include <function.hpp>
```

Inheritance diagram for SemSolver::Function< X, Y >:



Public Member Functions

- Function ()

  Default constructor.

- virtual ~Function ()

    Destructor.

- virtual Y evaluate (X const &) const

    Evaluate function at a point.

- virtual QString mml () const

    Get function definition in Mathematical Markup Language notation.

6.11.1 Detailed Description

template<class X, class Y> class SemSolver::Function< X, Y >

Prototype class for mathematical functions : $X \to Y$.

Parameters

X	The type of the domain
Y	The type of the codomain

Definition at line 19 of file function.hpp.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 template<class X, class Y> SemSolver::Function< X, Y >::Function ( ) [inline]

Default constructor.

Definition at line 25 of file function.hpp.

6.11.2.2 template<class X, class Y> virtual SemSolver::Function< X, Y >::~Function ( ) [inline, virtual]

Destructor.

Definition at line 28 of file function.hpp.

6.11.3 Member Function Documentation

6.11.3.1 template<class X, class Y> virtual Y SemSolver::Function< X, Y >::evaluate (X const &) const [inline, virtual]

Evaluate function at a point.

Returns

value assumed by function at x

Reimplemented in SemSolver::PolynomialFunction< d, X >, SemSolver::ScriptFunction< Point< d, X >, Y >, SemSolver::ScriptFunction< Point< d, X >, Vector< Y > >, SemSolver::ScriptFunction< Point< 2, X >, Y >, SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > >, SemSolver::SemFunction< 2, X >, and SemSolver::SemSpace< 2, X >::Element.

Definition at line 32 of file function.hpp.

6.11.3.2 template<class X, class Y> virtual QString SemSolver::Function< X, Y >::mml ( ) const [inline, virtual]

Get function definition in Mathematical Markup Language notation.

Returns

QString of function definition in MathML format

Reimplemented in SemSolver::ScriptFunction< Point< d, X >, Y >, SemSolver::ScriptFunction< Point< d, X >, Vector< Y > >, SemSolver::ScriptFunction< Point< 2, X >, Y >, and SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > >.

Definition at line 36 of file function.hpp.

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

- function.hpp

6.12 SemSolver::HilbertSpace< Function, X > Class Template Reference

prototype class for handling the concept of Hilbert Space

`#include <hilbertspace.hpp>`

Classes

- class Element

  Class for handling space elements as Fourier coefficients.

Public Member Functions

- virtual Function const * baseFunction (int const &index) const

  Get a base function.

- virtual X scalarProduct (Element const &, Element const &) const

    Evaluate the scalar product between two elements.

- virtual X norm (Element element)

    Evaluate the norm of an element.

- virtual int dimension () const

    Get the dimension of the space.

- virtual Element projection (Function const ∗) const

    Get the projection of a generic function on the Hilbert Space.

6.12.1 Detailed Description

template<class Function, class X> class SemSolver::HilbertSpace< Function, X >

prototype class for handling the concept of Hilbert Space

Parameters

Function	the type of the elements of the space
X	The type returned by scalar product

Definition at line 15 of file hilbertspace.hpp.

6.12.2 Member Function Documentation

6.12.2.1 template<class Function, class X> virtual Function const∗ SemSolver::HilbertSpace< Function, X >::baseFunction (int const & *index*) const [inline, virtual]

Get a base function.

Parameters

index	of the base function to be accessed

Returns

Pointer to function

Definition at line 37 of file hilbertspace.hpp.

6.12.2.2 template<class Function, class X> virtual int SemSolver::HilbertSpace< Function, X >::dimension ( ) const [inline, virtual]

Get the dimension of the space.

Returns

Base size

Definition at line 65 of file hilbertspace.hpp.

6.12.2.3 template<class Function, class X> virtual X SemSolver::HilbertSpace< Function, X >::norm (Element *element*) [inline, virtual]

Evaluate the norm of an element.

Parameters

element	

Returns

Square root of scalar product of element with itself

Definition at line 58 of file hilbertspace.hpp.

6.12.2.4 template<class Function, class X> virtual Element SemSolver::HilbertSpace< Function, X >::projection (Function const *) const [inline, virtual]

Get the projection of a generic function on the Hilbert Space.

Returns

Space element

Definition at line 72 of file hilbertspace.hpp.

6.12.2.5 template<class Function, class X> virtual X SemSolver::HilbertSpace< Function, X >::scalarProduct (Element const &, Element const &) const [inline, virtual]

Evaluate the scalar product between two elements.

Returns

Default constructed X

Definition at line 49 of file hilbertspace.hpp.

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

- hilbertspace.hpp

6.13 SemSolver::PSLG< X >::Hole Struct Reference

PSLG Hole struct.

`#include <pslg.hpp>`

Public Attributes

- int number
- X x
- X y

6.13.1 Detailed Description

template<class X> struct SemSolver::PSLG< X >::Hole

PSLG Hole struct. Holes are specified by identifying a point inside each hole.

Definition at line 57 of file pslg.hpp.

6.13.2 Member Data Documentation

6.13.2.1 template<class X> int SemSolver::PSLG< X >::Hole::number

Definition at line 59 of file pslg.hpp.

6.13.2.2 template<class X> X SemSolver::PSLG< X >::Hole::x

Definition at line 60 of file pslg.hpp.

6.13.2.3 template<class X> X SemSolver::PSLG< X >::Hole::y

Definition at line 61 of file pslg.hpp.

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

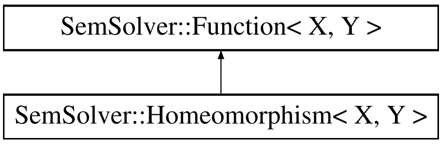
- pslg.hpp

6.14 SemSolver::Homeomorphism< X, Y > Class Template Reference

Prototype class for mathematical homemorphism : $X \to Y$.

```
#include <homeomorphism.hpp>
```

Inheritance diagram for SemSolver::Homeomorphism< X, Y >:



Public Member Functions

- Homeomorphism ()

  Default constructor.

- virtual ~Homeomorphism ()

  Destructor.

- virtual X evaluateInverse (Y const &)

  Evaluate function inverse of a value y.

6.14.1 Detailed Description

template<class X, class Y> class SemSolver::Homeomorphism< X, Y >

Prototype class for mathematical homemorphism : $X \to Y$.

Parameters

X	The type of the domain
Y	The type of the codomain

Definition at line 13 of file homeomorphism.hpp.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 template<class X, class Y> SemSolver::Homeomorphism< X, Y >::Homeomorphism ( ) [inline]

Default constructor.

Definition at line 18 of file homeomorphism.hpp.

6.14.2.2 template<class X, class Y> virtual SemSolver::Homeomorphism< X, Y >::~Homeomorphism ( ) [inline, virtual]

Destructor.

Definition at line 21 of file homeomorphism.hpp.

6.14.3 Member Function Documentation

6.14.3.1 template<class X, class Y> virtual X SemSolver::Homeomorphism< X, Y >::evaluateInverse (Y const &) [inline, virtual]

Evaluate function inverse of a value y.

Returns

The preimage of y

Definition at line 25 of file homeomorphism.hpp.

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

- homeomorphism.hpp

6.15 SemSolver::Segment< 2, X >::less Struct Reference

`#include <segment.hpp>`

Public Member Functions

- bool operator() (Segment const &s0, Segment const &s1) const

6.15.1 Detailed Description

template<class X> struct SemSolver::Segment< 2, X >::less

Definition at line 63 of file segment.hpp.

6.15.2 Member Function Documentation

6.15.2.1 template<class X > bool SemSolver::Segment< 2, X >::less::operator() (Segment< 2, X > const & *s0*, Segment< 2, X > const & *s1*) const [inline]

Definition at line 65 of file segment.hpp.

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

- segment.hpp

6.16 SemSolver::MultiIndex< N >::less Struct Reference

Partial order.

`#include <multiindex.hpp>`

Public Member Functions

- bool operator() (MultiIndex< N > const &mi0, MultiIndex< N > const &mi1) const

6.16.1 Detailed Description

template<int N> struct SemSolver::MultiIndex< N >::less

Partial order.

Definition at line 59 of file multiindex.hpp.

6.16.2 Member Function Documentation

6.16.2.1 template<int N> bool SemSolver::MultiIndex< N >::less::operator() (MultiIndex< N > const & *mi0*, MultiIndex< N > const & *mi1*) const `[inline]`

Definition at line 61 of file multiindex.hpp.

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

- multiindex.hpp

6.17 SemSolver::Matrix< X > Class Template Reference

Class for handling mathematical matrices.

`#include <matrix.hpp>`

Public Member Functions

- Matrix ()

  Construct an empty (0x0) matrix.

- Matrix (int rows, int columns)

Construct a matrix of a specific size.

- Matrix (int rows, int columns, X const &value)

    Construct a matrix with specific size and equal entries.

- int rows () const

    Get the number of rows.

- int columns () const

    Get the number of columns.

- Vector< X > realEigenvalues () const

    Get the eigenvalues.

Friends

- template<class Y >
  Matrix< Y > operator+ (Matrix< Y > const &, Matrix< Y > const &)

6.17.1 Detailed Description

template<class X> class SemSolver::Matrix< X >

Class for handling mathematical matrices.

Definition at line 20 of file matrix.hpp.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 template<class X > SemSolver::Matrix< X >::Matrix ( )

Construct an empty (0x0) matrix.

Definition at line 52 of file matrix.hpp.

6.17.2.2 template<class X > SemSolver::Matrix< X >::Matrix (int *rows*, int *columns*)

Construct a matrix of a specific size.

Parameters

rows	Number of rows
columns	Number of columns

Definition at line 61 of file matrix.hpp.

6.17.2.3 template<class X > SemSolver::Matrix< X >::Matrix (int *rows*, int *columns*, X const & *value*)

Construct a matrix with specific size and equal entries.

Parameters

rows	Number of rows
columns	Number of columns
value	to assign to matrix entries

Definition at line 71 of file matrix.hpp.

6.17.3 Member Function Documentation

6.17.3.1 template<class X > int SemSolver::Matrix< X >::columns ( ) const [inline]

Get the number of columns.

Returns

Columns number

Definition at line 87 of file matrix.hpp.

6.17.3.2 template<class X > SemSolver::Vector< X > SemSolver::Matrix< X >::realEigenvalues ( ) const

Get the eigenvalues.

Returns

Eigenvalues vector

Definition at line 95 of file matrix.hpp.

6.17.3.3 template<class X > int SemSolver::Matrix< X >::rows ( ) const [inline]

Get the number of rows.

Returns

Rows number

Definition at line 79 of file matrix.hpp.

6.17.4 Friends And Related Function Documentation

6.17.4.1 template<class X> template<class Y > Matrix<Y> operator+ (Matrix< Y > const & , Matrix< Y > const &) [friend]

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

- matrix.hpp

6.18 SemSolver::MultiIndex< N > Class Template Reference

Class multi-index notation.

`#include <multiindex.hpp>`

Classes

- struct less

  Partial order.

Public Member Functions

- MultiIndex ()

  Construct zero multi-index.

- MultiIndex (int const sub_indices[N])

  Construct multi-index.

- MultiIndex (MultiIndex< N > const &index)

  Copy constructor.

- int const & subIndex (int const &index) const

  Get a component.

- void setSubIndex (int const &index, int const &sub_index)

  Set a component.

6.18.1 Detailed Description

template<int N> class SemSolver::MultiIndex< N >

Class multi-index notation.

Parameters

N	Length of the multi-index

Definition at line 10 of file multiindex.hpp.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 template<int N> SemSolver::MultiIndex< N >::MultiIndex ( ) [inline]

Construct zero multi-index.

Definition at line 16 of file multiindex.hpp.

6.18.2.2 template<int N> SemSolver::MultiIndex< N >::MultiIndex (int const *sub_indices[N]*) [inline]

Construct multi-index.

Parameters

sub_indices	Index components

Definition at line 24 of file multiindex.hpp.

6.18.2.3 template<int N> SemSolver::MultiIndex< N >::MultiIndex (MultiIndex< N > const & *index*) [inline]

Copy constructor.

Parameters

index	The MultiIndex to be copied

Definition at line 32 of file multiindex.hpp.

6.18.3 Member Function Documentation

6.18.3.1 template<int N> void SemSolver::MultiIndex< N >::setSubIndex (int const & *index*, int const & *sub_index*) [inline]

Set a component.

Parameters

index	The component index
sub_index	The component value

Definition at line 51 of file multiindex.hpp.

6.18.3.2 template<int N> int const& SemSolver::MultiIndex< N >::subIndex (int const & *index*) const [inline]

Get a component.

Parameters

index	The component index

Returns

The component value

Definition at line 41 of file multiindex.hpp.

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

- multiindex.hpp

6.19 SemSolver::SemSpace< 2, X >::Node Class Reference

`#include <semspace.hpp>`

Public Member Functions

- Node (Point< 2, X > const &point)

  Construct node from a point.

- Point< 2, X > const & point () const

  Access node point.

- int supportSubDomains () const

  Get number of subdomain of which node is member.

- MultiIndex< 3 > const & subDomainIndex (int const &index) const

  Get the index-th subdomain index of wich node is member.

- int supportBorders () const

  Get number of borders of which node is member.

- MultiIndex< 2 > const & borderIndex (int const &index) const

  Get the index-th border index of wich node is member.

Friends

- class SemSpace

6.19.1 Detailed Description

template<class X> class SemSolver::SemSpace< 2, X >::Node

Class for space nodes It stores information about the subdomains and borders of which is member

Definition at line 51 of file semspace.hpp.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 template<class X > SemSolver::SemSpace< 2, X >::Node::Node (Point< 2, X > const & *point*) [inline]

Construct node from a point.

Definition at line 98 of file semspace.hpp.

6.19.3 Member Function Documentation

6.19.3.1 template<class X > MultiIndex<2> const& SemSolver::SemSpace< 2, X >::Node::borderIndex (int const & *index*) const [inline]

Get the index-th border index of wich node is member.

Definition at line 129 of file semspace.hpp.

6.19.3.2 template<class X > Point<2,X> const& SemSolver::SemSpace< 2, X >::Node::point ( ) const [inline]

Access node point.

Definition at line 102 of file semspace.hpp.

6.19.3.3 template<class X > MultiIndex<3> const& SemSolver::SemSpace< 2, X >::Node::subDomainIndex (int const & *index*) const [inline]

Get the index-th subdomain index of wich node is member.

Definition at line 114 of file semspace.hpp.

6.19.3.4 template<class X > int SemSolver::SemSpace< 2, X >::Node::supportBorders ( ) const [inline]

Get number of borders of which node is member.

Definition at line 123 of file semspace.hpp.

6.19.3.5 template<class X > int SemSolver::SemSpace< 2, X >::Node::supportSubDomains ( ) const [inline]

Get number of subdomain of which node is member.

Definition at line 108 of file semspace.hpp.

6.19.4 Friends And Related Function Documentation

6.19.4.1 template<class X > friend class SemSpace [friend]

Definition at line 134 of file semspace.hpp.

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

- semspace.hpp

6.20 SemSolver::Point< 2, X > Class Template Reference

Class for handling 2D euclidean points.

`#include <point.hpp>`

Public Member Functions

- Point ()
  Default constructor.
- Point (CGAL_Point const &cgal_point)
  Construct Point from CGAL_Point.
- Point (X const &x, X const &y)
  Construct a Point with given coordinates.

6.20.1 Detailed Description

template<class X> class SemSolver::Point< 2, X >

Class for handling 2D euclidean points.

Parameters

X	Must be a type for which operations +, -, $*$ and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of $*$

Definition at line 38 of file point.hpp.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 template<class X > SemSolver::Point< 2, X >::Point ( ) [inline]

Default constructor.

Definition at line 64 of file point.hpp.

6.20.2.2 template<class X > SemSolver::Point< 2, X >::Point (CGAL_Point< 2, X > const & *cgal_point*) [inline]

Construct Point from CGAL_Point.

Definition at line 69 of file point.hpp.

6.20.2.3 template<class X > SemSolver::Point< 2, X >::Point (X const & *x*, X const & *y*) [inline]

Construct a Point with given coordinates.

Parameters

x	Abscissa
y	Ordinate

Definition at line 74 of file point.hpp.

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

- point.hpp

6.21 SemSolver::PointsBimap< 2, X > Class Template Reference

Class for handling bi-directional maps between 2D Points and Integers.

```
#include <pointsbimap.hpp>
```

Public Types

- typedef Base::KeyType KeyType
- typedef Base::MappedType MappedType
- typedef Base::ValueType ValueType
- typedef Base::ConstIterator ConstIterator
- typedef Base::SizeType SizeType

Public Member Functions

- PointsBimap (const X &tolerance)

  Constructor.

- Iterator insert (const ValueType &x)

  Insert a point-id value.

- MappedType insertPoint (const KeyType &x)

  Insert a point if it doesn't exists otherwise do nothing.

- const KeyType & point (const MappedType &y) const

  Get a point.

- const MappedType & id (const KeyType &x) const

  Get an id.

- void modifyPoint (const MappedType &y, const KeyType &x)
- void modifyId (const KeyType &x, const MappedType &y)
- void erasePoint (const KeyType &x)

  Erase a point-id entry if exists.

- void eraseId (const MappedType &y)

  Erase a point-id entry if exists.

- ConstIterator findPoint (const KeyType &x) const

  Find a point-id entry by its point value.

- Iterator findPoint (const KeyType &x)

  Find a point-id entry by its point value.

- ConstIterator findId (const MappedType &y) const

  Find a point-id entry by its id value.

- Iterator findId (const MappedType &y)

  Find a point-id entry by its id value.

- bool hasPoint (const KeyType &x) const

Check if an entry exists.

- bool hasId (const MappedType &y) const

  Check if an entry exists.

- bool hasPointOn (const Segment< 2, X > &s) const

  Check if there is a point lying on a given segment.

6.21.1 Detailed Description

template<class X> class SemSolver::PointsBimap< 2, X >

Class for handling bi-directional maps between 2D Points and Integers.

Parameters

X	Must be a type for which operations +, -, ∗ and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of ∗

Definition at line 36 of file pointsbimap.hpp.

6.21.2 Member Typedef Documentation

6.21.2.1 template<class X > typedef Base::ConstIterator SemSolver::PointsBimap< 2, X >::ConstIterator

Definition at line 46 of file pointsbimap.hpp.

6.21.2.2 template<class X > typedef Base::KeyType SemSolver::PointsBimap< 2, X >::KeyType

Definition at line 43 of file pointsbimap.hpp.

6.21.2.3 template<class X > typedef Base::MappedType SemSolver::PointsBimap< 2, X >::MappedType

Definition at line 44 of file pointsbimap.hpp.

6.21.2.4 template<class X > typedef Base::SizeType SemSolver::PointsBimap< 2, X >::SizeType

Definition at line 47 of file pointsbimap.hpp.

6.21.2.5 template<class X > typedef Base::ValueType SemSolver::PointsBimap< 2, X >::ValueType

Definition at line 45 of file pointsbimap.hpp.

6.21.3 Constructor & Destructor Documentation

6.21.3.1 template<class X > SemSolver::PointsBimap< 2, X >::PointsBimap (const X & *tol*) [inline]

Constructor.

Parameters

tol	Points whose distance is below this value are treated as the same point

Definition at line 108 of file pointsbimap.hpp.

6.21.4 Member Function Documentation

6.21.4.1 template<class X > void SemSolver::PointsBimap< 2, X >::eraseId (const MappedType & *y*) [inline]

Erase a point-id entry if exists.

Parameters

y	The id to be removed

Definition at line 215 of file pointsbimap.hpp.

6.21.4.2 template<class X > void SemSolver::PointsBimap< 2, X >::erasePoint (const KeyType & *x*) [inline]

Erase a point-id entry if exists.

Parameters

x	The point to be removed

Definition at line 205 of file pointsbimap.hpp.

6.21.4.3 template<class X > SemSolver::PointsBimap< 2, X >::ConstIterator SemSolver::PointsBimap< 2, X >::findId (const MappedType & *id*) const

Find a point-id entry by its id value.

Parameters

id	The id value

Returns

ConstIterator to the entry if found, to end otherwise

Definition at line 247 of file pointsbimap.hpp.

6.21.4.4 template< class X > SemSolver::PointsBimap< 2, X >::Iterator SemSolver::PointsBimap< 2, X >::findId (const MappedType & *id*)

Find a point-id entry by its id value.

Parameters

id	The id value

Returns

ConstIterator to the entry if found, to end otherwise

Definition at line 265 of file pointsbimap.hpp.

6.21.4.5 template< class X > SemSolver::PointsBimap< 2, X >::ConstIterator SemSolver::PointsBimap< 2, X >::findPoint (const KeyType & *x*) const [inline]

Find a point-id entry by its point value.

Parameters

x	The point value

Returns

ConstIterator to the entry if found, to end otherwise

Definition at line 227 of file pointsbimap.hpp.

6.21.4.6 template< class X > SemSolver::PointsBimap< 2, X >::Iterator SemSolver::PointsBimap< 2, X >::findPoint (const KeyType & *x*) [inline]

Find a point-id entry by its point value.

Parameters

x	The point value

Returns

Iterator to the entry if found, to end otherwise

Definition at line 237 of file pointsbimap.hpp.

6.21.4.7 template<class X > bool SemSolver::PointsBimap< 2, X >::hasId (const MappedType & *y*) const

Check if an entry exists.

Parameters

y	The id to be found

Returns

Whether the id exists or not

Definition at line 291 of file pointsbimap.hpp.

6.21.4.8 template<class X > bool SemSolver::PointsBimap< 2, X >::hasPoint (const KeyType & *x*) const [inline]

Check if an entry exists.

Parameters

x	The point to be found

Returns

Whether the point exists or not

Definition at line 282 of file pointsbimap.hpp.

6.21.4.9 template<class X > bool SemSolver::PointsBimap< 2, X >::hasPointOn (const Segment< 2, X > & *segment*) const [inline]

Check if there is a point lying on a given segment.

Parameters

segment	The segment to be checked

Returns

Whether there exists such a point or not

Definition at line 300 of file pointsbimap.hpp.

6.21.4.10 template<class X > const SemSolver::PointsBimap< 2, X >::MappedType & SemSolver::PointsBimap< 2, X >::id (const KeyType & *x*) const [inline]

Get an id.

Parameters

x	The point to be find. It must exist

Returns

The id corresponding to the specified point

Definition at line 160 of file pointsbimap.hpp.

6.21.4.11 template<class X > SemSolver::PointsBimap< 2, X >::Iterator SemSolver::PointsBimap< 2, X >::insert (const ValueType & *x*)

Insert a point-id value.

Definition at line 116 of file pointsbimap.hpp.

6.21.4.12 template<class X > SemSolver::PointsBimap< 2, X >::MappedType SemSolver::PointsBimap< 2, X >::insertPoint (const KeyType & *x*) [inline]

Insert a point if it doesn't exists otherwise do nothing.

Parameters

x	the point to be inserted

Returns

the id of the point inserted

Definition at line 128 of file pointsbimap.hpp.

6.21.4.13 template<class X > void SemSolver::PointsBimap< 2, X >::modifyId (const KeyType & *x*, const MappedType & *y*) [inline]

Modify an id

Parameters

x	The point corresponding to the id to be modified. It must exist
y	The new id value

Definition at line 190 of file pointsbimap.hpp.

6.21.4.14 template<class X > void SemSolver::PointsBimap< 2, X >::modifyPoint (const MappedType & *y*, const KeyType & *x*) [inline]

Modify a point

Parameters

y	The id corresponding to the point to be modified. It must exist
x	The new point value

Definition at line 174 of file pointsbimap.hpp.

6.21.4.15 template<class X > const SemSolver::PointsBimap< 2, X >::KeyType & SemSolver::PointsBimap< 2, X >::point (const MappedType & *y*) const [inline]

Get a point.

Parameters

y	The id of the point to be find. It must exist

Returns

The point corresponding to the specified id

Definition at line 138 of file pointsbimap.hpp.

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

- pointsbimap.hpp

6.22 SemSolver::PointsMap< 2, X, Y > Class Template Reference

Class for handling maps with 2D Point as KeyType.

`#include <pointsmap.hpp>`

Public Types

- typedef Point< 2, X > KeyType
- typedef Y MappedType
- typedef list::value_type ValueType
- typedef list::const_iterator ConstIterator
- typedef list::size_type SizeType

Public Member Functions

- Iterator begin ()
- Iterator end ()
- Iterator find (const KeyType &x)
- PointsMap (const X &tolerance)

  Constructor.
- ConstIterator begin () const

  Get ConstIterator to first element.
- void clear ()

  Clear map constent.
- bool has (const KeyType &x) const

  Check if an entry exists.
- bool isEmpty () const

  Check if there are entries.
- ConstIterator end () const

  Get ConstIterator to past-the-end location.
- void erase (Iterator position)

  Erase an entry.
- SizeType erase (const KeyType &x)

  Erase entries if exist.
- void erase (Iterator first, Iterator last)

  Erase entries in a range.
- ConstIterator find (const KeyType &x) const

  Find a point-id entry by its point value.
- Iterator insert (const ValueType &x)

  Insert a point if it doesn't exists otherwise do nothing.
- Iterator insert (const KeyType &x, const MappedType &y)

  Insert a point if it doesn't exists otherwise do nothing.
- Iterator insert (Iterator position, const ValueType &x)

  Insert a point if it doesn't exists otherwise do nothing.
- template<class InputIterator >
  void insert (InputIterator first, InputIterator last)

Insert multiple pairs if they don't exist.

- SizeType size () const

    Get the map size.

- MappedType & operator[ ] (const KeyType &key)

    Access element.

Protected Types

- typedef list::iterator Iterator

6.22.1 Detailed Description

template<class X, class Y> class SemSolver::PointsMap< 2, X, Y >

Class for handling maps with 2D Point as KeyType.

Parameters

X	Must be a type for which operations +, -, $*$ and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of $*$
Y	the MappedType

Definition at line 37 of file pointsmap.hpp.

6.22.2 Member Typedef Documentation

6.22.2.1 template<class X , class Y > typedef list::const_iterator SemSolver::PointsMap< 2, X, Y >::ConstIterator

Definition at line 48 of file pointsmap.hpp.

6.22.2.2 template<class X , class Y > typedef list::iterator SemSolver::PointsMap< 2, X, Y >::Iterator `[protected]`

Definition at line 42 of file pointsmap.hpp.

6.22.2.3 template<class X , class Y > typedef Point<2, X> SemSolver::PointsMap< 2, X, Y >::KeyType

Definition at line 45 of file pointsmap.hpp.

6.22.2.4 template< class X , class Y > typedef Y SemSolver::PointsMap< 2, X, Y >::MappedType

Definition at line 46 of file pointsmap.hpp.

6.22.2.5 template< class X , class Y > typedef list::size_type SemSolver::PointsMap< 2, X, Y >::SizeType

Definition at line 49 of file pointsmap.hpp.

6.22.2.6 template< class X , class Y > typedef list::value_type SemSolver::PointsMap< 2, X, Y >::ValueType

Definition at line 47 of file pointsmap.hpp.

6.22.3 Constructor & Destructor Documentation

6.22.3.1 template< class X , class Y > SemSolver::PointsMap< 2, X, Y >::PointsMap (const X & *tol*) [inline]

Constructor.

Parameters

tol	Points whose distance is below this value are treated as the same point

Definition at line 104 of file pointsmap.hpp.

6.22.4 Member Function Documentation

6.22.4.1 template< class X , class Y > SemSolver::PointsMap< 2, X, Y >::Iterator SemSolver::PointsMap< 2, X, Y >::begin ( ) [inline]

Definition at line 111 of file pointsmap.hpp.

6.22.4.2 template< class X , class Y > SemSolver::PointsMap< 2, X, Y >::ConstIterator SemSolver::PointsMap< 2, X, Y >::begin ( ) const [inline]

Get ConstIterator to first element.

Returns

ConstIterator to the first element of the map

Definition at line 120 of file pointsmap.hpp.

6.22.4.3 template<class X , class Y > void SemSolver::PointsMap< 2, X, Y >::clear ( ) [inline]

Clear map constent.

Definition at line 127 of file pointsmap.hpp.

6.22.4.4 template<class X , class Y > SemSolver::PointsMap< 2, X, Y >::Iterator SemSolver::PointsMap< 2, X, Y >::end ( ) [inline]

Definition at line 150 of file pointsmap.hpp.

6.22.4.5 template<class X , class Y > SemSolver::PointsMap< 2, X, Y >::ConstIterator SemSolver::PointsMap< 2, X, Y >::end ( ) const [inline]

Get ConstIterator to past-the-end location.

Returns

ConstIterator to the location succeeding the last element in a map

Definition at line 159 of file pointsmap.hpp.

6.22.4.6 template<class X , class Y > void SemSolver::PointsMap< 2, X, Y >::erase (Iterator *position*) [inline]

Erase an entry.

Parameters

position	Iterator to the element to be removed

Definition at line 167 of file pointsmap.hpp.

6.22.4.7 template<class X , class Y > SemSolver::PointsMap< 2, X, Y >::SizeType SemSolver::PointsMap< 2, X, Y >::erase (const KeyType & *x*) [inline]

Erase entries if exist.

Parameters

x	The point value corresponding to the entry to be removed

Returns

The number of elements erased: 1 or 0

Definition at line 177 of file pointsmap.hpp.

6.22.4.8 template< class X , class Y > void SemSolver::PointsMap< 2, X, Y >::erase (Iterator *first*, Iterator *last*) [inline]

Erase entries in a range.

Parameters

first	Iterator to the first entry to be removed
last	Iterator to the position just beyond the last element to be removed

Definition at line 186 of file pointsmap.hpp.

6.22.4.9 template< class X , class Y > SemSolver::PointsMap< 2, X, Y >::Iterator SemSolver::PointsMap< 2, X, Y >::find (const KeyType & *x*) [inline]

Definition at line 213 of file pointsmap.hpp.

6.22.4.10 template< class X , class Y > SemSolver::PointsMap< 2, X, Y >::ConstIterator SemSolver::PointsMap< 2, X, Y >::find (const KeyType & *x*) const [inline]

Find a point-id entry by its point value.

Parameters

x	The point value

Returns

ConstIterator to the entry if found, to end otherwise

Definition at line 196 of file pointsmap.hpp.

6.22.4.11 template< class X , class Y > bool SemSolver::PointsMap< 2, X, Y >::has (const KeyType & *x*) const [inline]

Check if an entry exists.

Parameters

x	The point to be found

Returns

Whether the entry exists or not

Definition at line 136 of file pointsmap.hpp.

6.22.4.12 template<class X , class Y > SemSolver::PointsMap< 2, X, Y >::Iterator SemSolver::PointsMap< 2, X, Y >::insert (Iterator *it*, const ValueType & *x*)

Insert a point if it doesn't exists otherwise do nothing.

Parameters

x	the point-mapped_value pair to be inserted
it	iterator to the position guess where to insert new entry

Returns

the iterator to the point inserted

Definition at line 256 of file pointsmap.hpp.

6.22.4.13 template<class X , class Y > template<class InputIterator > void SemSolver::PointsMap< 2, X, Y >::insert (InputIterator *first*, InputIterator *last*)

Insert multiple pairs if they don't exist.

Parameters

first	iterator to the first pair
last	iterator to the last pair

Definition at line 282 of file pointsmap.hpp.

6.22.4.14 template<class X , class Y > SemSolver::PointsMap< 2, X, Y >::Iterator SemSolver::PointsMap< 2, X, Y >::insert (const ValueType & *x*) [inline]

Insert a point if it doesn't exists otherwise do nothing.

Parameters

x	the point-mapped_value pair to be inserted

Returns

the iterator to the point inserted

Definition at line 233 of file pointsmap.hpp.

6.22.4.15 template<class X , class Y > SemSolver::PointsMap< 2, X, Y >::Iterator SemSolver::PointsMap< 2, X, Y >::insert (const KeyType & *x*, const MappedType & *y*) [inline]

Insert a point if it doesn't exists otherwise do nothing.

Parameters

x	the point value to be inserted
y	the mapped value to be inserted

Returns

the iterator to the point inserted

Definition at line 244 of file pointsmap.hpp.

6.22.4.16 template< class X , class Y > bool SemSolver::PointsMap< 2, X, Y >::isEmpty ( ) const [inline]

Check if there are entries.

Returns

Whether the map is empty or not

Definition at line 144 of file pointsmap.hpp.

6.22.4.17 template< class X , class Y > SemSolver::PointsMap< 2, X, Y >::MappedType & SemSolver::PointsMap< 2, X, Y >::operator[] (const KeyType & x) [inline]

Access element.

If x matches the key of an element in the container, the function returns a reference to its mapped value. If x does not match the key of any element in the container, the function inserts a new element with that key and returns a reference to its mapped value.

Parameters

x	Key value of the element whose mapped value is accessed.

Returns

A reference to the element with a key value equal to x.

Definition at line 309 of file pointsmap.hpp.

6.22.4.18 template< class X , class Y > SemSolver::PointsMap< 2, X, Y >::SizeType SemSolver::PointsMap< 2, X, Y >::size ( ) const [inline]

Get the map size.

Returns

The map size

Definition at line 294 of file pointsmap.hpp.

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

- pointsmap.hpp

6.23 SemSolver::PointsSet< 2, X > Class Template Reference

Class for handling sets of 2D Points.

`#include <pointsset.hpp>`

Public Types

- typedef Point< 2, X > KeyType
- typedef Point< 2, X > MappedType
- typedef list::value_type ValueType
- typedef list::const_iterator ConstIterator
- typedef list::size_type SizeType

Public Member Functions

- PointsSet (const X &tolerance)

  Constructor.
- ConstIterator begin () const

  Get ConstIterator to first element.
- void clear ()

  Clear set constent.
- bool has (const KeyType &x) const

  Check if an entry exists.
- bool isEmpty () const

  Check if there are entries.
- ConstIterator end () const

  Get ConstIterator to past-the-end location.
- void erase (Iterator position)

  Erase an entry.
- SizeType erase (const KeyType &x)

  Erase entries if exist.

- void erase (Iterator first, Iterator last)

  Erase entries in a range.

- ConstIterator find (const KeyType &x) const

  Find a point entry.

- Iterator insert (const ValueType &x)

  Insert a point if it doesn't exists otherwise do nothing.

- Iterator insert (Iterator position, const ValueType &x)

  Insert a point if it doesn't exists otherwise do nothing.

- template<class InputIterator >
  void insert (InputIterator first, InputIterator last)

  Insert multiple pairs if they don't exist.

- SizeType size () const

  Get the set size.

- MappedType & operator[ ] (const KeyType &key)

  Access element.

Protected Types

- typedef list::iterator Iterator

Protected Member Functions

- Iterator begin ()
- Iterator end ()
- Iterator find (const KeyType &x)

6.23.1 Detailed Description

template<class X> class SemSolver::PointsSet< 2, X >

Class for handling sets of 2D Points.

Parameters

X	Must be a type for which operations +, -, $*$ and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of $*$

Definition at line 35 of file pointsset.hpp.

6.23.2 Member Typedef Documentation

6.23.2.1 template<class X > typedef list::const_iterator SemSolver::PointsSet< 2, X >::ConstIterator

Definition at line 46 of file pointsset.hpp.

6.23.2.2 template<class X > typedef list::iterator SemSolver::PointsSet< 2, X >::Iterator [protected]

Definition at line 40 of file pointsset.hpp.

6.23.2.3 template<class X > typedef Point<2, X> SemSolver::PointsSet< 2, X >::KeyType

Definition at line 43 of file pointsset.hpp.

6.23.2.4 template<class X > typedef Point<2, X> SemSolver::PointsSet< 2, X >::MappedType

Definition at line 44 of file pointsset.hpp.

6.23.2.5 template<class X > typedef list::size_type SemSolver::PointsSet< 2, X >::SizeType

Definition at line 47 of file pointsset.hpp.

6.23.2.6 template<class X > typedef list::value_type SemSolver::PointsSet< 2, X >::ValueType

Definition at line 45 of file pointsset.hpp.

6.23.3 Constructor & Destructor Documentation

6.23.3.1 template<class X > SemSolver::PointsSet< 2, X >::PointsSet (const X & *tol*) [inline]

Constructor.

Parameters

tol	Points whose distance is below this value are treated as the same point

Definition at line 101 of file pointsset.hpp.

6.23.4 Member Function Documentation

6.23.4.1 template<class X > SemSolver::PointsSet< 2, X >::Iterator SemSolver::PointsSet< 2, X >::begin ( ) [inline, protected]

Definition at line 108 of file pointsset.hpp.

6.23.4.2 template<class X > SemSolver::PointsSet< 2, X >::ConstIterator SemSolver::PointsSet< 2, X >::begin ( ) const [inline]

Get ConstIterator to first element.

Returns

ConstIterator to the first element of the set

Definition at line 117 of file pointsset.hpp.

6.23.4.3 template<class X > void SemSolver::PointsSet< 2, X >::clear ( ) [inline]

Clear set constent.

Definition at line 124 of file pointsset.hpp.

6.23.4.4 template<class X > SemSolver::PointsSet< 2, X >::Iterator SemSolver::PointsSet< 2, X >::end ( ) [inline, protected]

Definition at line 147 of file pointsset.hpp.

6.23.4.5 template<class X > SemSolver::PointsSet< 2, X >::ConstIterator SemSolver::PointsSet< 2, X >::end ( ) const [inline]

Get ConstIterator to past-the-end location.

Returns

ConstIterator to the location succeeding the last element in a set

Definition at line 156 of file pointsset.hpp.

6.23.4.6 template<class X > void SemSolver::PointsSet< 2, X >::erase (Iterator *position*) [inline]

Erase an entry.

Parameters

position	Iterator to the element to be removed

Definition at line 164 of file pointsset.hpp.

6.23.4.7 template<class X > SemSolver::PointsSet< 2, X >::SizeType SemSolver::PointsSet< 2, X >::erase (const KeyType & *x*) [inline]

Erase entries if exist.

Parameters

x	The point value corresponding to the entry to be removed

Returns

The number of elements erased: 1 or 0

Definition at line 174 of file pointsset.hpp.

6.23.4.8 template<class X > void SemSolver::PointsSet< 2, X >::erase (Iterator *first*, Iterator *last*) [inline]

Erase entries in a range.

Parameters

first	Iterator to the first entry to be removed
last	Iterator to the position just beyond the last element to be removed

Definition at line 183 of file pointsset.hpp.

6.23.4.9 template<class X > SemSolver::PointsSet< 2, X >::Iterator SemSolver::PointsSet< 2, X >::find (const KeyType & *x*) [inline, protected]

Definition at line 210 of file pointsset.hpp.

6.23.4.10 template<class X > SemSolver::PointsSet< 2, X >::ConstIterator SemSolver::PointsSet< 2, X >::find (const KeyType & *x*) const [inline]

Find a point entry.

Parameters

x	The point value to search

Returns

ConstIterator to the entry if found, to end otherwise

Definition at line 193 of file pointsset.hpp.

6.23.4.11 template<class X > bool SemSolver::PointsSet< 2, X >::has (const KeyType & *x*) const [inline]

Check if an entry exists.

Parameters

x	The point to be found

Returns

Whether the entry exists or not

Definition at line 133 of file pointsset.hpp.

6.23.4.12 template<class X > template<class InputIterator > void SemSolver::PointsSet< 2, X >::insert (InputIterator *first*, InputIterator *last*)

Insert multiple pairs if they don't exist.

Parameters

first	iterator to the first pair
last	iterator to the last pair

Definition at line 267 of file pointsset.hpp.

6.23.4.13 template<class X > SemSolver::PointsSet< 2, X >::Iterator SemSolver::PointsSet< 2, X >::insert (const ValueType & *x*) [inline]

Insert a point if it doesn't exists otherwise do nothing.

Parameters

x	the point to be inserted

Returns

the iterator to the point inserted

Definition at line 230 of file pointsset.hpp.

6.23.4.14 template<class X > SemSolver::PointsSet< 2, X >::Iterator SemSolver::PointsSet< 2, X >::insert (Iterator *it*, const ValueType & *x*)

Insert a point if it doesn't exists otherwise do nothing.

Parameters

x	the point to be inserted
it	iterator to the position guess where to insert new entry

Returns

the iterator to the point inserted

Definition at line 241 of file pointsset.hpp.

6.23.4.15 template<class X > bool SemSolver::PointsSet< 2, X >::isEmpty ( ) const [inline]

Check if there are entries.

Returns

Whether the set is empty or not

Definition at line 141 of file pointsset.hpp.

6.23.4.16 template<class X > SemSolver::PointsSet< 2, X >::MappedType & SemSolver::PointsSet< 2, X >::operator[] (const KeyType & *x*) [inline]

Access element.

If x matches the key of an element in the container, the function returns a reference to its mapped value. If x does not match the key of any element in the container, the function inserts a new element with that key and returns a reference to its mapped value.

Parameters

x	Key value of the element whose mapped value is accessed.

Returns

A reference to the element with a key value equal to x.

Definition at line 293 of file pointsset.hpp.

6.23.4.17 template<class X > SemSolver::PointsSet< 2, X >::SizeType SemSolver::PointsSet< 2, X >::size ( ) const [inline]

Get the set size.

Returns

The set size

Definition at line 278 of file pointsset.hpp.

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

- pointsset.hpp

6.24 SemSolver::Polygon< 2, X > Class Template Reference

Class for handling 2D polygons.

`#include <polygon.hpp>`

Public Member Functions

- Polygon ()

  Default constructor.

- Polygon (CGAL_point const ∗first, CGAL_point const ∗last)

  Construct Polygon from a Point's sequence.

- bool contains (Point< 2, X > const &point) const

  Test if a point lies on the Polygon.

6.24.1 Detailed Description

template<class X> class SemSolver::Polygon< 2, X >

Class for handling 2D polygons.

Parameters

X	Must be a type for which operations +, -, ∗ and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of ∗

Definition at line 37 of file polygon.hpp.

6.24.2 Constructor & Destructor Documentation

6.24.2.1 template<class X > SemSolver::Polygon< 2, X >::Polygon ( ) [inline]

Default constructor.

Definition at line 70 of file polygon.hpp.

6.24.2.2 template<class X > SemSolver::Polygon< 2, X >::Polygon (CGAL_point const * *first*, CGAL_point const * *last*) [inline]

Construct Polygon from a Point's sequence.

Parameters

first	Pointer to the first Point of the sequence
last	Pointer to the last Point of the sequence

Definition at line 75 of file polygon.hpp.

6.24.3 Member Function Documentation

6.24.3.1 template<class X > bool SemSolver::Polygon< 2, X >::contains (Point< 2, X > const & *point*) const [inline]

Test if a point lies on the Polygon.

Parameters

point	The Point to test with

Returns

The test result

Definition at line 81 of file polygon.hpp.

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

- polygon.hpp

6.25 SemSolver::Polygonation< 2, X > Class Template Reference

Class for handling 2D polygonations.

`#include <polygonation.hpp>`

Classes

- class Element

  A Polygonation element.

Public Member Functions

- bool isQuadrangulation () const

Test if Polygonation is a quadrangualtion.

- void refine ()

  Refine the Polygonation.

- void clear ()

  Clear the Polygonation.

- const Element & element (const unsigned &index) const
- unsigned size () const
- void addElement (Polygon< 2, X > const &polygon, std::vector< int > const &neighbours)
- std::vector< unsigned > elementIndicesAt (Point< 2, X > const &point) const

6.25.1 Detailed Description

template<class X> class SemSolver::Polygonation< 2, X >

Class for handling 2D polygonations.

Parameters

X	Must be a type for which operations +, -, $*$ and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of $*$

Definition at line 36 of file polygonation.hpp.

6.25.2 Member Function Documentation

6.25.2.1 template<class X > void SemSolver::Polygonation< 2, X >::addElement (Polygon< 2, X > const & *polygon*, std::vector< int > const & *neighbours*) [inline]

Add an Element to the Polygonation

Parameters

polygon	The geometry of the element
neighbours	Vector of neighbour element ids in counterclokwise order

Definition at line 251 of file polygonation.hpp.

6.25.2.2 template<class X > void SemSolver::Polygonation< 2, X >::clear ( ) [inline]

Clear the Polygonation.

Definition at line 228 of file polygonation.hpp.

6.25.2.3 template<class X > const SemSolver::Polygonation< 2, X >::Element & SemSolver::Polygonation< 2, X >::element (const unsigned & *index*) const [inline]

Get an Element

Parameters

index	Element's position

Returns

reference to Element

Definition at line 235 of file polygonation.hpp.

6.25.2.4 template<class X > std::vector< unsigned > SemSolver::Polygonation< 2, X >::elementIndicesAt (SemSolver::Point< 2, X > const & *point*) const

Get on which Elements lies a Point

Parameters

point	The Point to check

Returns

Vector of position of that elements that contain the given point

Definition at line 258 of file polygonation.hpp.

6.25.2.5 template<class X > bool SemSolver::Polygonation< 2, X >::isQuadrangulation ( ) const

Test if Polygonation is a quadrangualtion.

Returns

The test result

Definition at line 162 of file polygonation.hpp.

6.25.2.6 template<class X > void SemSolver::Polygonation< 2, X >::refine ( )

Refine the Polygonation.

Split all elements into subelements, one for each vertex, by adding as vertices the centroid of each element and the midpoint of each segment

Definition at line 174 of file polygonation.hpp.

6.25.2.7 template<class X > unsigned SemSolver::Polygonation< 2, X >::size ( ) const [inline]

Get Polygonation size

Returns

The number of elements

Definition at line 245 of file polygonation.hpp.

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

- polygonation.hpp

6.26 SemSolver::PolygonWithHoles< 2, X > Class Template Reference

Class for handling 2D polygons with holes.

`#include <polygonwithholes.hpp>`

Public Member Functions

- CGAL_Polygon_with_holes const & cgal ()

6.26.1 Detailed Description

template<class X> class SemSolver::PolygonWithHoles< 2, X >

Class for handling 2D polygons with holes.

Parameters

X	Must be a type for which operations +, -, $*$ and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of $*$

Definition at line 37 of file polygonwithholes.hpp.

6.26.2 Member Function Documentation

6.26.2.1 template<class X > CGAL_Polygon_with_holes const& SemSolver::PolygonWithHoles< 2, X >::cgal ( ) [inline]

Definition at line 46 of file polygonwithholes.hpp.

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

- polygonwithholes.hpp

6.27 SemSolver::Polynomial< X > Class Template Reference

Class for handing the mathematical concept of polynomials over X.

`#include <polynomial.hpp>`

Public Member Functions

- Polynomial ()

  Default constructor.

- Polynomial (X const &scalar)

  Construct a constant polynomial.

- Polynomial (std::vector< X > const &coefficients)

  Constuct a polynomial from its coefficients.

- Polynomial (Polynomial const &poly)

  Copy constructor.

- ~Polynomial ()

  Destructor.

- Polynomial & operator= (Polynomial const &poly)

  Assignment operator.

- Polynomial & operator+= (Polynomial const &poly)

  Addition assignment operator.

- Polynomial & operator-= (Polynomial const &poly)

  Subtraction assignment operator.

- Polynomial & operator*= (Polynomial const &poly)

  Multiplication assignment operator.

- Polynomial & operator/= (Polynomial const &poly)

  Division assignment operator.

- Polynomial & operator%= (const Polynomial &poly)

  Modulo assignment operator.

- Polynomial operator+ () const

  Unary plus.

- Polynomial operator- () const

  Unary minus.

- Polynomial operator+ (const Polynomial &poly) const

  Addition operator.

- Polynomial operator- (const Polynomial &poly) const

  Subtraction operator.

- Polynomial operator* (const Polynomial &poly) const

  Multiplication operator.

- Polynomial operator/ (const Polynomial &poly) const

  Division operator.

- Polynomial operator% (const Polynomial &poly) const

  Modulo operator.

- bool operator== (const Polynomial &poly) const

  Equal to operator.

- bool operator!= (const Polynomial &poly) const

  Not equal to operator.

- double operator() (const X &variable) const
- double coefficient (const int &order) const
- int degree () const

  Get the degree of a polynomial.

- void setCoefficient (const int &order, X const &coefficient)

  Set the coefficient of given order.

- void setDegree (const int &degree)

  Set the polynomial degree.

- Polynomial derivative () const

  Compute polynomial derivative.

- Polynomial ruffini (X const &zero) const
- std::vector< X > zeros () const

    Compute the vector of polynomial roots.

6.27.1 Detailed Description

template<class X> class SemSolver::Polynomial< X >

Class for handing the mathematical concept of polynomials over X.

Definition at line 14 of file polynomial.hpp.

6.27.2 Constructor & Destructor Documentation

6.27.2.1 template<class X> SemSolver::Polynomial< X >::Polynomial ( ) [inline]

Default constructor.

Definition at line 22 of file polynomial.hpp.

6.27.2.2 template<class X> SemSolver::Polynomial< X >::Polynomial (X const & *scalar*) [inline]

Construct a constant polynomial.

Parameters

scalar	The scalar value of the polynomial

Definition at line 29 of file polynomial.hpp.

6.27.2.3 template<class X> SemSolver::Polynomial< X >::Polynomial (std::vector< X > const & *coefficients*) [inline]

Constuct a polynomial from its coefficients.

Parameters

coefficients	Vector of coefficients from that of order 0

Definition at line 36 of file polynomial.hpp.

6.27.2.4 template<class X> SemSolver::Polynomial< X >::Polynomial (Polynomial< X > const & *poly*) [inline]

Copy constructor.

Parameters

poly	the polynomial to be copied

Definition at line 43 of file polynomial.hpp.

6.27.2.5 template<class X> SemSolver::Polynomial< X >::~Polynomial ( ) [inline]

Destructor.

Definition at line 49 of file polynomial.hpp.

6.27.3 Member Function Documentation

6.27.3.1 template<class X> double SemSolver::Polynomial< X >::coefficient (const int & *order*) const [inline]

Access coefficient

Parameters

order	The order of the coefficient to access

Definition at line 237 of file polynomial.hpp.

6.27.3.2 template<class X> int SemSolver::Polynomial< X >::degree ( ) const [inline]

Get the degree of a polynomial.

Definition at line 248 of file polynomial.hpp.

6.27.3.3 template<class X> Polynomial SemSolver::Polynomial< X >::derivative ( ) const [inline]

Compute polynomial derivative.

Definition at line 282 of file polynomial.hpp.

6.27.3.4 template<class X> bool SemSolver::Polynomial< X >::operator!= (const Polynomial< X > & *poly*) const

Not equal to operator.

6.27.3.5 template<class X> Polynomial SemSolver::Polynomial< X >::operator% (const Polynomial< X > & *poly*) const [inline]

Modulo operator.

Parameters

poly	the divisor used for computing the reminder

Definition at line 205 of file polynomial.hpp.

6.27.3.6 template<class X> Polynomial& SemSolver::Polynomial< X >::operator%= (const Polynomial< X > & *poly*) [inline]

Modulo assignment operator.

Parameters

poly	the divisor used for computing the reminder

Definition at line 127 of file polynomial.hpp.

6.27.3.7 template<class X> double SemSolver::Polynomial< X >::operator() (const X & *variable*) const [inline]

Definition at line 227 of file polynomial.hpp.

6.27.3.8 template<class X> Polynomial SemSolver::Polynomial< X >::operator* (const Polynomial< X > & *poly*) const [inline]

Multiplication operator.

Parameters

poly	the polynomial to be multiplied

Definition at line 187 of file polynomial.hpp.

6.27.3.9 template<class X> Polynomial& SemSolver::Polynomial< X >::operator*= (Polynomial< X > const & *poly*) [inline]

Multiplication assignment operator.

Parameters

poly	the polynomial to be multiplied

Definition at line 86 of file polynomial.hpp.

6.27.3.10 template<class X> Polynomial SemSolver::Polynomial< X >::operator+ (const Polynomial< X > & *poly*) const [inline]

Addition operator.

Parameters

poly	the polynomial to be summed

Definition at line 169 of file polynomial.hpp.

6.27.3.11 template<class X> Polynomial SemSolver::Polynomial< X >::operator+ ( ) const [inline]

Unary plus.

Definition at line 152 of file polynomial.hpp.

6.27.3.12 template<class X> Polynomial& SemSolver::Polynomial< X >::operator+= (Polynomial< X > const & *poly*) [inline]

Addition assignment operator.

Parameters

poly	the polynomial to be summed

Definition at line 62 of file polynomial.hpp.

6.27.3.13 template<class X> Polynomial SemSolver::Polynomial< X >::operator- ( ) const [inline]

Unary minus.

Definition at line 158 of file polynomial.hpp.

6.27.3.14 template< class X > Polynomial SemSolver::Polynomial< X >::operator- (const Polynomial< X > & *poly*) const [inline]

Subtraction operator.

Parameters

poly	the polynomial to be subtracted

Definition at line 178 of file polynomial.hpp.

6.27.3.15 template< class X > Polynomial& SemSolver::Polynomial< X >::operator-= (Polynomial< X > const & *poly*) [inline]

Subtraction assignment operator.

Parameters

poly	the polynomial to be subtracted

Definition at line 74 of file polynomial.hpp.

6.27.3.16 template< class X > Polynomial SemSolver::Polynomial< X >::operator/ (const Polynomial< X > & *poly*) const [inline]

Division operator.

Parameters

poly	the polynomial by which divide

Definition at line 196 of file polynomial.hpp.

6.27.3.17 template< class X > Polynomial& SemSolver::Polynomial< X >::operator/= (Polynomial< X > const & *poly*) [inline]

Division assignment operator.

Parameters

poly	the polynomial by which divide

Definition at line 101 of file polynomial.hpp.

6.27.3.18 template<class X> Polynomial& SemSolver::Polynomial< X >::operator= (Polynomial< X > const & *poly*) [inline]

Assignment operator.

Parameters

poly	the polynomial to be copied

Definition at line 54 of file polynomial.hpp.

6.27.3.19 template<class X> bool SemSolver::Polynomial< X >::operator== (const Polynomial< X > & *poly*) const [inline]

Equal to operator.

Definition at line 213 of file polynomial.hpp.

6.27.3.20 template<class X> Polynomial SemSolver::Polynomial< X >::ruffini (X const & *zero*) const [inline]

Compute quotient of the division by a monic binomial with Ruffini's rule zero the 0-order coefficient of the divisor

Definition at line 296 of file polynomial.hpp.

6.27.3.21 template<class X> void SemSolver::Polynomial< X >::setCoefficient (const int & *order*, X const & *coefficient*) [inline]

Set the coefficient of given order.

Parameters

order	Must be non negative and not greater than degree
coefficient	The value to be assigned

Definition at line 259 of file polynomial.hpp.

6.27.3.22 template<class X> void SemSolver::Polynomial< X >::setDegree (const int & *degree*) [inline]

Set the polynomial degree.

Parameters

degree	Must be not less than -1

Definition at line 271 of file polynomial.hpp.

6.27.3.23 template<class X> std::vector<X> SemSolver::Polynomial< X >::zeros ( ) const [inline]

Compute the vector of polynomial roots.

Definition at line 307 of file polynomial.hpp.

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

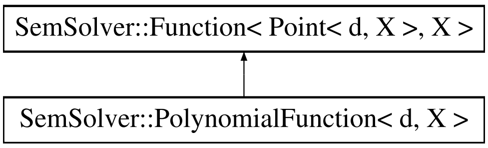
- polynomial.hpp

6.28 SemSolver::PolynomialFunction< d, X > Class Template Reference

Class for handling polynomial separable functions.

```
#include <polynomialfunction.hpp>
```

Inheritance diagram for SemSolver::PolynomialFunction< d, X >:



Public Member Functions

- PolynomialFunction ()
  Default Constructor.
- PolynomialFunction (Polynomial< X > const *polynomials)
  Construct Polynomial function from polynomials.
- PolynomialFunction (PolynomialFunction< d, X > const &polynomial_function)
  Copy constructor.
- ~PolynomialFunction ()
  Destructor.
- PolynomialFunction< d, X > & operator= (PolynomialFunction< d, X > const &polynomial_function)
  Assigment operator.
- Polynomial< X > const & polynomial (int const &index) const
- void setPolynomial (int const &index, Polynomial< X > const &poly)

Set function projection at index.

- X evaluate (Point< d, X > const &x) const

  Compute function value at Point x.

6.28.1 Detailed Description

template<int d, class X> class SemSolver::PolynomialFunction< d, X >

Class for handling polynomial separable functions. The image of a point X of dimension d is the product of d polynomial evaluated at each coordinate of X, i.e. for d=3 F(X) = f1(x1)∗f2(x2)∗f3(x3)

Parameters

d	Dimension of the space
X	Must be a type for which operations +, -, ∗ and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of ∗

Definition at line 22 of file polynomialfunction.hpp.

6.28.2 Constructor & Destructor Documentation

6.28.2.1 template<int d, class X> SemSolver::PolynomialFunction< d, X >::PolynomialFunction ( ) [inline]

Default Constructor.

Definition at line 29 of file polynomialfunction.hpp.

6.28.2.2 template<int d, class X> SemSolver::PolynomialFunction< d, X >::PolynomialFunction (Polynomial< X > const ∗ *polynomials*) [inline]

Construct Polynomial function from polynomials.

Definition at line 36 of file polynomialfunction.hpp.

6.28.2.3 template<int d, class X> SemSolver::PolynomialFunction< d, X >::PolynomialFunction (PolynomialFunction< d, X > const & *polynomial_function*) [inline]

Copy constructor.

Definition at line 43 of file polynomialfunction.hpp.

6.28.2.4 template<int d, class X> SemSolver::PolynomialFunction< d, X >::~PolynomialFunction ( ) [inline]

Destructor.

Definition at line 52 of file polynomialfunction.hpp.

6.28.3 Member Function Documentation

6.28.3.1 template<int d, class X> X SemSolver::PolynomialFunction< d, X >::evaluate (Point< d, X > const & *x*) const [inline, virtual]

Compute function value at Point x.

Reimplemented from SemSolver::Function< Point< d, X >, X >.

Definition at line 87 of file polynomialfunction.hpp.

6.28.3.2 template<int d, class X> PolynomialFunction<d,X>& SemSolver::PolynomialFunction< d, X >::operator= (PolynomialFunction< d, X > const & *polynomial_function*) [inline]

Assigment operator.

Definition at line 56 of file polynomialfunction.hpp.

6.28.3.3 template<int d, class X> Polynomial<X> const& SemSolver::PolynomialFunction< d, X >::polynomial (int const & *index*) const [inline]

Access polynomial projection over a component

Parameters

index	The index of the conmpont on wich to project

Definition at line 65 of file polynomialfunction.hpp.

6.28.3.4 template<int d, class X> void SemSolver::PolynomialFunction< d, X >::setPolynomial (int const & *index*, Polynomial< X > const & *poly*) [inline]

Set function projection at index.

Definition at line 76 of file polynomialfunction.hpp.

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

- polynomialfunction.hpp

6.29 SemSolver::Problem< d, X > Class Template Reference

Class for handling a mathematical problem given by an equation, boundary conditions, geometry and parameters.

`#include <problem.hpp>`

Public Member Functions

- Problem ()

  Default constructor.

- ~Problem ()

  Destructor.

- void setGeometry (const SemGeometry< d, X > ∗geometry)

  Set the geometry on wich the problem is defined.

- void setEquation (const Equation< d, X > ∗equation)

  Set the equation describing the problem.

- void setBoundaryConditions (const BoundaryConditions< d, X > ∗boundary_-conditions)

  Set the boundary conditions for the problem.

- void setParameters (const SemParameters< X > ∗parameters)

  Set the parameters for the problem.

- void clearGeometry ()

  Clear problem geometry.

- void clearEquation ()

  Clear problem equation.

- void clearBoundaryConditions ()

  Clear problem boundary conditions.

- void clearParameters ()

  Clear problem parameters.

- const SemGeometry< d, X > ∗ geometry () const

  Access problem geometry.

- const Equation< d, X > ∗ equation () const

  Access problem equation.

- const BoundaryConditions< d, X > * boundaryConditions () const

  Access problem boundary conditions.

- const SemParameters< X > * parameters () const

  Access problem parameters.

- bool isDefined ()

6.29.1 Detailed Description

template<int d, class X> class SemSolver::Problem< d, X >

Class for handling a mathematical problem given by an equation, boundary conditions, geometry and parameters.

Definition at line 21 of file problem.hpp.

6.29.2 Constructor & Destructor Documentation

6.29.2.1 template<int d, class X > SemSolver::Problem< d, X >::Problem ( )

Default constructor.

Definition at line 53 of file problem.hpp.

6.29.2.2 template<int d, class X > SemSolver::Problem< d, X >::~Problem ( )

Destructor.

Definition at line 63 of file problem.hpp.

6.29.3 Member Function Documentation

6.29.3.1 template<int d, class X > const SemSolver::BoundaryConditions< d, X > * SemSolver::Problem< d, X >::boundaryConditions ( ) const [inline]

Access problem boundary conditions.

Definition at line 149 of file problem.hpp.

6.29.3.2 template<int d, class X > void SemSolver::Problem< d, X >::clearBoundaryConditions ( ) [inline]

Clear problem boundary conditions.

Definition at line 118 of file problem.hpp.

6.29.3.3 template< int d, class X > void SemSolver::Problem< d, X >::clearEquation ( ) [inline]

Clear problem equation.

Definition at line 110 of file problem.hpp.

6.29.3.4 template< int d, class X > void SemSolver::Problem< d, X >::clearGeometry ( ) [inline]

Clear problem geometry.

Definition at line 102 of file problem.hpp.

6.29.3.5 template< int d, class X > void SemSolver::Problem< d, X >::clearParameters ( ) [inline]

Clear problem parameters.

Definition at line 126 of file problem.hpp.

6.29.3.6 template< int d, class X > const SemSolver::Equation< d, X > * SemSolver::Problem< d, X >::equation ( ) const [inline]

Access problem equation.

Definition at line 141 of file problem.hpp.

6.29.3.7 template< int d, class X > const SemSolver::SemGeometry< d, X > * SemSolver::Problem< d, X >::geometry ( ) const [inline]

Access problem geometry.

Definition at line 134 of file problem.hpp.

6.29.3.8 template< int d, class X > bool SemSolver::Problem< d, X >::isDefined ( ) [inline]

Check if problem is well defined i.e. if geometry, equation, boundary conditions and parameters are defined for the problem

Definition at line 164 of file problem.hpp.

6.29.3.9 template< int d, class X > const SemSolver::SemParameters< X > * SemSolver::Problem< d, X >::parameters ( ) const [inline]

Access problem parameters.

Definition at line 156 of file problem.hpp.

6.29.3.10 template<int d, class X > void SemSolver::Problem< d, X >::setBoundaryConditions (const BoundaryConditions< d, X > * *boundary_conditions*) [inline]

Set the boundary conditions for the problem.

Definition at line 87 of file problem.hpp.

6.29.3.11 template<int d, class X > void SemSolver::Problem< d, X >::setEquation (const Equation< d, X > * *equation*) [inline]

Set the equation describing the problem.

Definition at line 80 of file problem.hpp.

6.29.3.12 template<int d, class X > void SemSolver::Problem< d, X >::setGeometry (const SemGeometry< d, X > * *geometry*) [inline]

Set the geometry on wich the problem is defined.

Definition at line 73 of file problem.hpp.

6.29.3.13 template<int d, class X > void SemSolver::Problem< d, X >::setParameters (const SemParameters< X > * *parameters*) [inline]

Set the parameters for the problem.

Definition at line 95 of file problem.hpp.

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

- problem.hpp

6.30 SemSolver::PSLG< X > Class Template Reference

Class for handing Planar Straight Line Graphs.

`#include <pslg.hpp>`

Classes

- struct Hole

  PSLG Hole struct.

- struct Segment

  PSLG Segment struct.

- struct Vertex

PSLG Vertex struct.

Public Member Functions

- PSLG ()

  Default constructor.

- ~PSLG ()

  Destructor.

- void clear ()

  Clear PSLG content.

- Vertex const & vertex (const unsigned &index) const
- Segment const & segment (const unsigned &index) const
- Hole const & hole (const unsigned &index) const
- unsigned const & vertices () const
- unsigned const & segments () const
- unsigned const & holes () const
- void setNumberOfVertices (const unsigned &number)
- void setNumberOfSegments (const unsigned &number)
- void setNumberOfHoles (const unsigned &number)
- void setNumberOfVerticesAttributes (const unsigned &number)
- void setNumberOfVerticesBoundaryMarkers (const unsigned &number)
- void setVertex (unsigned const &index, int const &number, const X &x, const X &y, double *attributes=0, int const &marker=0)
- void setSegment (unsigned const &index, int const &number, int const &source, int const &target, int const &marker=0)
- void setHole (unsigned const &index, int const &number, const X &x, const X &y)

6.30.1 Detailed Description

template<class X> class SemSolver::PSLG< X >

Class for handing Planar Straight Line Graphs. By definition, a PSLG is just a list of vertices and segments. It can also contain information about holes and concavities, as well as regional attributes and constraints on the areas of triangles.

Parameters

X	The type of coordinates

Definition at line 24 of file pslg.hpp.

6.30.2 Constructor & Destructor Documentation

6.30.2.1 template<class X > SemSolver::PSLG< X >::PSLG ( )

Default constructor.

Definition at line 176 of file pslg.hpp.

6.30.2.2 template<class X > SemSolver::PSLG< X >::~PSLG ( )

Destructor.

Definition at line 191 of file pslg.hpp.

6.30.3 Member Function Documentation

6.30.3.1 template<class X > void SemSolver::PSLG< X >::clear ( )

Clear PSLG content.

Definition at line 204 of file pslg.hpp.

6.30.3.2 template<class X > const SemSolver::PSLG< X >::Hole & SemSolver::PSLG< X >::hole (const unsigned & *index*) const [inline]

Get a hole

Parameters

index	The hole position

Returns

The hole

Definition at line 249 of file pslg.hpp.

6.30.3.3 template<class X > const unsigned & SemSolver::PSLG< X >::holes ( ) const [inline]

Get number of holes

Returns

Holes number

Definition at line 271 of file pslg.hpp.

6.30.3.4 template<class X > const SemSolver::PSLG< X >::Segment & SemSolver::PSLG< X >::segment (const unsigned & *index*) const [inline]

Get a segment

Parameters

index	The segment position

Returns

The segment

Definition at line 238 of file pslg.hpp.

6.30.3.5 template<class X > const unsigned & SemSolver::PSLG< X >::segments ( ) const [inline]

Get number of segments

Returns

Segments number

Definition at line 265 of file pslg.hpp.

6.30.3.6 template<class X > void SemSolver::PSLG< X >::setHole (unsigned const & *index*, int const & *number*, const X & *x*, const X & *y*)

Set a hole

Parameters

index	The hole position
number	The hole number
x	Abscissa
y	Ordinate

Definition at line 400 of file pslg.hpp.

6.30.3.7 template<class X > void SemSolver::PSLG< X >::setNumberOfHoles (const unsigned & *number*)

Set number of holes

Parameters

number	Holes number

Definition at line 312 of file pslg.hpp.

6.30.3.8 template<class X > void SemSolver::PSLG< X >::setNumberOfSegments (const unsigned & *number*)

Set number of segments

Parameters

number	Segments number

Definition at line 301 of file pslg.hpp.

6.30.3.9 template<class X > void SemSolver::PSLG< X >::setNumberOfVertices (const unsigned & *number*)

Set number of vertices

Parameters

number	Vertices number

Definition at line 277 of file pslg.hpp.

6.30.3.10 template<class X > void SemSolver::PSLG< X >::setNumberOfVerticesAttributes (const unsigned & *number*)

Set number of vertices attributes

Parameters

number	Vertices attributes number

Definition at line 323 of file pslg.hpp.

6.30.3.11 template<class X > void SemSolver::PSLG< X >::setNumberOfVerticesBoundaryMarkers (const unsigned & *number*) [inline]

Set number of vertices boundary markers

Parameters

number	Vertices boundary markers number

Definition at line 345 of file pslg.hpp.

6.30.3.12 template<class X > void SemSolver::PSLG< X >::setSegment (unsigned const & *index*, int const & *number*, int const & *source*, int const & *target*, int const & *marker* = 0)

Set a segment

Parameters

index	The segment position
number	The segment number
source	First end-point number
target	Second end-point number
marker	Boundary marker

Definition at line 383 of file pslg.hpp.

6.30.3.13 template<class X > void SemSolver::PSLG< X >::setVertex (unsigned const & *index*, int const & *number*, const X & *x*, const X & *y*, double * *attributes* = 0, int const & *marker* = 0)

Set a vertex

Parameters

index	The vertex position
number	The vertex number
x	Abscissa
y	Ordinate
attributes	Attributes array
marker	Boundary marker

Definition at line 356 of file pslg.hpp.

6.30.3.14 template<class X > const SemSolver::PSLG< X >::Vertex & SemSolver::PSLG< X >::vertex (const unsigned & *index*) const [inline]

Get a vertex

Parameters

index	The vertex position

Returns

The vertex

Definition at line 227 of file pslg.hpp.

6.30.3.15 template<class X > const unsigned & SemSolver::PSLG< X >::vertices ( ) const [inline]

Get number of vertices

Returns

Vertices number

Definition at line 259 of file pslg.hpp.

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

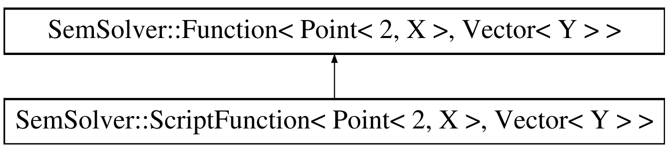
- pslg.hpp

6.31 SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > > Class Template Reference

Class for handling function from 2D Euclidean space X^2 to Vectorial space Y^n defined in scripts.

```
#include <scriptfunction.hpp>
```

Inheritance diagram for SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > >:



Public Member Functions

- ScriptFunction (QStringList const &strings)

  Constructor.

- ~ScriptFunction ()

  Destructor.

- Vector< Y > evaluate (const Point< 2, X > &point) const

  Compute function value at a point.

- QString mml () const

  Get function definition in Mathematical Markup Language notation.

6.31.1 Detailed Description

template<class X, class Y> class SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > >

Class for handling function from 2D Euclidean space X^2 to Vectorial space Y^n defined in scripts. Such functions are evaluated at run time and allow users to speecify the function definition during execution for example by using external ECMA Script files.

Definition at line 122 of file scriptfunction.hpp.

6.31.2 Constructor & Destructor Documentation

6.31.2.1 template<class X , class Y > SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > >::ScriptFunction (QStringList const & *strings*)

Constructor.

Parameters

strings	Function component definitions in ECMA Script language. Variables are assumed to be x and y. To the function is associated the ECMA Script "function f0(x, y) { return string[0] }, function f1(x, y) { return string[1] }, ..., function fn(x, y) { return string[n] }". For ease of use, mathematical function of Math object are redifined, in this way sin(x) can be used instead of Math.sin(x)

6.31.2.2 template<class X , class Y > SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > >::~ScriptFunction ( )

Destructor.

6.31.3 Member Function Documentation

6.31.3.1 template<class X , class Y > Vector<Y> SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > >::evaluate (const Point< 2, X > & *point*) const `[virtual]`

Compute function value at a point.

Reimplemented from SemSolver::Function< Point< 2, X >, Vector< Y > >.

6.31.3.2 template<class X , class Y > QString SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > >::mml ( ) const `[virtual]`

Get function definition in Mathematical Markup Language notation.

Returns

QString of function definition in MathML format

Reimplemented from SemSolver::Function< Point< 2, X >, Vector< Y > >.

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

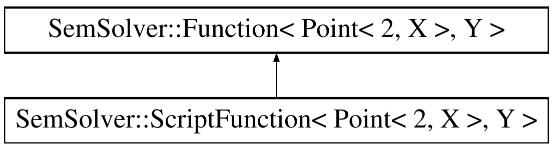
- scriptfunction.hpp

6.32 SemSolver::ScriptFunction< Point< 2, X >, Y > Class Template Reference

Class for handling function from 2D Euclidean space X^2 to scalar space Y defined in scripts.

```
#include <scriptfunction.hpp>
```

Inheritance diagram for SemSolver::ScriptFunction< Point< 2, X >, Y >:



Public Member Functions

- ScriptFunction (QString const &string)
  Constructor.
- ~ScriptFunction ()
  Destructor.
- Y evaluate (const Point< 2, X > &point) const
  Compute function value at a point.
- QString mml () const
  Get function definition in Mathematical Markup Language notation.

6.32.1 Detailed Description

template<class X, class Y> class SemSolver::ScriptFunction< Point< 2, X >, Y >

Class for handling function from 2D Euclidean space X^2 to scalar space Y defined in scripts. Such functions are evaluated at run time and allow users to speecify the function definition during execution for example by using external ECMA Script files.

Definition at line 91 of file scriptfunction.hpp.

6.32.2 Constructor & Destructor Documentation

6.32.2.1 template< class X , class Y > SemSolver::ScriptFunction< Point< 2, X >, Y >::ScriptFunction (QString const & *string*)

Constructor.

Parameters

string	Function definition in ECMA Script language. Variables are assumed to be x, y. To the function associated the ECMA Script "function f(x,y) { return string }". For ease of use, mathematical function of Math object are redifined, in this way sin(x) can be used instead of Math.sin(x)

6.32.2.2 template< class X , class Y > SemSolver::ScriptFunction< Point< 2, X >, Y >::~ScriptFunction ( )

Destructor.

6.32.3 Member Function Documentation

6.32.3.1 template< class X , class Y > Y SemSolver::ScriptFunction< Point< 2, X >, Y >::evaluate (const Point< 2, X > & *point*) const [virtual]

Compute function value at a point.

Reimplemented from SemSolver::Function< Point< 2, X >, Y >.

6.32.3.2 template< class X , class Y > QString SemSolver::ScriptFunction< Point< 2, X >, Y >::mml ( ) const [virtual]

Get function definition in Mathematical Markup Language notation.

Returns

QString of function definition in MathML format

Reimplemented from SemSolver::Function< Point< 2, X >, Y >.

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

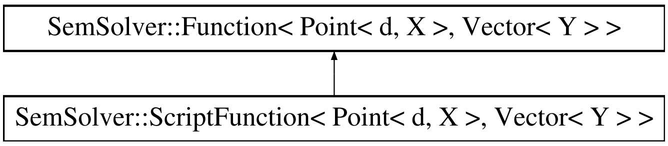
- scriptfunction.hpp

6.33 SemSolver::ScriptFunction< Point< d, X >, Vector< Y > > Class Template Reference

Class for handling function from Euclidean space X^d to Vectorial space Y^n defined inscripts.

```
#include <scriptfunction.hpp>
```

Inheritance diagram for SemSolver::ScriptFunction< Point< d, X >, Vector< Y > >:



Public Member Functions

- ScriptFunction (QStringList const &strings)

  Constructor.

- ~ScriptFunction ()

  Destructor.

- Vector< Y > evaluate (const Point< d, X > &point) const

  Compute function value at a point.

- QString mml () const

  Get function definition in Mathematical Markup Language notation.

6.33.1 Detailed Description

template<int d, class X, class Y> class SemSolver::ScriptFunction< Point< d, X >, Vector< Y > >

Class for handling function from Euclidean space X^d to Vectorial space Y^n defined inscripts. Such functions are evaluated at run time and allow users to speecify the function definition during execution for example by using external ECMA Script files.

Definition at line 58 of file scriptfunction.hpp.

6.33.2 Constructor & Destructor Documentation

6.33.2.1 template<int d, class X , class Y > SemSolver::ScriptFunction< Point< d, X >, Vector< Y > >::ScriptFunction (QStringList const & *strings*)

Constructor.

Parameters

strings	Function component definitions in ECMA Script language. Variables are assumed to be x0, x1, ..., xd-1. To the function is associated the ECMA Script "function f0(x0, x1, ..., xd-1) { return string[0] }, function f1(x0, x1, ..., xd-1) { return string[1] }, ..., function fn(x0, x1, ..., xd-1) { return string[n] }". For ease of use, mathematical function of Math object are redifined, in this way sin(x) can be used instead of Math.sin(x)

6.33.2.2 template<int d, class X , class Y > SemSolver::ScriptFunction< Point< d, X >, Vector< Y > >::~ScriptFunction ( )

Destructor.

6.33.3 Member Function Documentation

6.33.3.1 template<int d, class X , class Y > Vector<Y> SemSolver::ScriptFunction< Point< d, X >, Vector< Y > >::evaluate (const Point< d, X > & *point*) const [virtual]

Compute function value at a point.

Reimplemented from SemSolver::Function< Point< d, X >, Vector< Y > >.

6.33.3.2 template<int d, class X , class Y > QString SemSolver::ScriptFunction< Point< d, X >, Vector< Y > >::mml ( ) const [virtual]

Get function definition in Mathematical Markup Language notation.

Returns

QString of function definition in MathML format

Reimplemented from SemSolver::Function< Point< d, X >, Vector< Y > >.

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

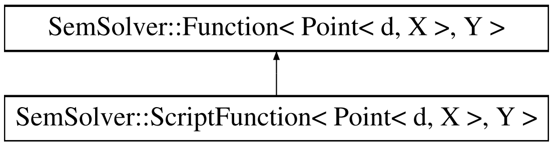
- scriptfunction.hpp

6.34 SemSolver::ScriptFunction< Point< d, X >, Y > Class Template Reference

Class for handling function from Euclidean space X^d to scalar space Y defined in scripts.

```
#include <scriptfunction.hpp>
```

Inheritance diagram for SemSolver::ScriptFunction< Point< d, X >, Y >:



Public Member Functions

- ScriptFunction (QString const &string)

  Constructor.

- ~ScriptFunction ()

  Destructor.

- Y evaluate (const Point< d, X > &point) const

  Compute function value at a point.

- QString mml () const

  Get function definition in Mathematical Markup Language notation.

6.34.1 Detailed Description

template<int d, class X, class Y> class SemSolver::ScriptFunction< Point< d, X >, Y >

Class for handling function from Euclidean space X^d to scalar space Y defined in scripts. Such functions are evaluated at run time and allow users to speecify the function definition during execution for example by using external ECMA Script files.

Definition at line 27 of file scriptfunction.hpp.

6.34.2 Constructor & Destructor Documentation

6.34.2.1 template<int d, class X , class Y > SemSolver::ScriptFunction< Point< d, X >, Y >::ScriptFunction (QString const & *string*)

Constructor.

Parameters

string	Function definition in ECMA Script language. Variables are assumed to be x0, x1, ..., xd-1. To the function associated the ECMA Script "function f(x0, x1, ..., xd-1) { return string }". For ease of use, mathematical function of Math object are redifined, in this way sin(x) can be used instead of Math.sin(x)

6.34.2.2 template<int d, class X , class Y > SemSolver::ScriptFunction< Point< d, X >, Y >::~ScriptFunction ( )

Destructor.

6.34.3 Member Function Documentation

6.34.3.1 template<int d, class X , class Y > Y SemSolver::ScriptFunction< Point< d, X >, Y >::evaluate (const Point< d, X > & *point*) const `[virtual]`

Compute function value at a point.

Reimplemented from SemSolver::Function< Point< d, X >, Y >.

6.34.3.2 template<int d, class X , class Y > QString SemSolver::ScriptFunction< Point< d, X >, Y >::mml ( ) const `[virtual]`

Get function definition in Mathematical Markup Language notation.

Returns

QString of function definition in MathML format

Reimplemented from SemSolver::Function< Point< d, X >, Y >.

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

- scriptfunction.hpp

6.35 SemSolver::PSLG< X >::Segment Struct Reference

PSLG Segment struct.

`#include <pslg.hpp>`

Public Attributes

- int number
- int source

- int target
- int marker

6.35.1 Detailed Description

template<class X> struct SemSolver::PSLG< X >::Segment

PSLG Segment struct. Segment is specified by listing the indices of its two endpoints. This means that you must include its endpoints in the vertex list. Each segment, like each vertex, may have a boundary marker.

Definition at line 47 of file pslg.hpp.

6.35.2 Member Data Documentation

6.35.2.1 template<class X> int SemSolver::PSLG< X >::Segment::marker

Definition at line 52 of file pslg.hpp.

6.35.2.2 template<class X> int SemSolver::PSLG< X >::Segment::number

Definition at line 49 of file pslg.hpp.

6.35.2.3 template<class X> int SemSolver::PSLG< X >::Segment::source

Definition at line 50 of file pslg.hpp.

6.35.2.4 template<class X> int SemSolver::PSLG< X >::Segment::target

Definition at line 51 of file pslg.hpp.

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

- pslg.hpp

6.36 SemSolver::Segment< 2, X > Class Template Reference

```
#include <segment.hpp>
```

Classes

- struct less

Public Member Functions

- Segment (CGAL_segment const &s)
- Segment (Point< 2, X > const &source, Point< 2, X > const &target)

    Construct an oriented segment from source point to target point.

- CGAL_segment const & cgal () const
- bool intersect (Segment< 2, X > const &segment) const

    Check if two segments intersect.

- bool intersectInteriorly (Segment< 2, X > const &segment) const

    Check if two segment intersect themselves not in their endpoints.

6.36.1 Detailed Description

template<class X> class SemSolver::Segment< 2, X >

Definition at line 21 of file segment.hpp.

6.36.2 Constructor & Destructor Documentation

6.36.2.1 template<class X > SemSolver::Segment< 2, X >::Segment (CGAL_segment const & *s*) [inline]

Definition at line 31 of file segment.hpp.

6.36.2.2 template<class X > SemSolver::Segment< 2, X >::Segment (Point< 2, X > const & *source*, Point< 2, X > const & *target*) [inline]

Construct an oriented segment from source point to target point.

Definition at line 34 of file segment.hpp.

6.36.3 Member Function Documentation

6.36.3.1 template<class X > CGAL_segment const& SemSolver::Segment< 2, X >::cgal ( ) const [inline]

Definition at line 37 of file segment.hpp.

6.36.3.2 template<class X > bool SemSolver::Segment< 2, X >::intersect (Segment< 2, X > const & *segment*) const [inline]

Check if two segments intersect.

Definition at line 43 of file segment.hpp.

6.36.3.3 template<class X > bool SemSolver::Segment< 2, X >::intersectInteriorly (Segment< 2, X > const & *segment*) const [inline]

Check if two segment intersect themselves not in their endpoints.

Definition at line 49 of file segment.hpp.

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

- segment.hpp

6.37 SemSolver::SegmentsMap< d, X > Class Template Reference

```
#include <segmentsmap.hpp>
```

Public Types

- typedef Base::const_iterator ConstIterator

Public Member Functions

- SegmentsMap ()

  Default constructor.

- bool contains (Segment< d, X > const &segment) const

  Check if map contains a segment.

- bool has (int const &segment_id) const

  Check id an id is already used.

- Segment< d, X > segmentFrom (Point< d, X > const &source) const

  Get first segment in the map whith specified source.

- Segment< d, X > segmentTo (Point< d, X > const &target) const

  Get first segment in the map whith specified target.

- Segment< d, X > const & segment (int const &id) const

  Access segment by key.

- int id (Segment< d, X > const &segment) const
- int add (Segment< d, X > const &segment)
- void insert (int const &id, Segment< d, X > const &segment)

  Insert a id-segment pair in map If id already exists do nothing.

- void remove (int const &id)

  Remove entry with specified id if exists.

- void modify (int const &id, Segment< d, X > const &segment)
- int segments () const

  Get size of map.

- virtual bool intersect (Segment< d, X > const &segment) const

  Check if there is a segment in map intesecting specified segment.

- virtual bool intersectInteriorly (Segment< d, X > const &segment) const
- bool isConsistentWith (Segment< d, X > const &segment) const
- bool haveOn (Point< d, X > const &point) const

  Check if given point lies on a segment of the map or not.

Protected Types

- typedef Base::iterator Iterator

6.37.1 Detailed Description

template<int d, class X> class SemSolver::SegmentsMap< d, X >

Definition at line 26 of file segmentsmap.hpp.

6.37.2 Member Typedef Documentation

6.37.2.1 template<int d, class X > typedef Base::const_iterator SemSolver::SegmentsMap< d, X >::ConstIterator

Definition at line 36 of file segmentsmap.hpp.

6.37.2.2 template<int d, class X > typedef Base::iterator SemSolver::SegmentsMap< d, X >::Iterator [protected]

Definition at line 33 of file segmentsmap.hpp.

6.37.3 Constructor & Destructor Documentation

6.37.3.1 template<int d, class X > SemSolver::SegmentsMap< d, X >::SegmentsMap ( ) [inline]

Default constructor.

Definition at line 46 of file segmentsmap.hpp.

6.37.4 Member Function Documentation

6.37.4.1 template<int d, class X > int SemSolver::SegmentsMap< d, X >::add (Segment< d, X > const & *segment*) [inline]

Inset a segment in map

Returns

Id associated with segment

Definition at line 111 of file segmentsmap.hpp.

6.37.4.2 template<int d, class X > bool SemSolver::SegmentsMap< d, X >::contains (Segment< d, X > const & *segment*) const [inline]

Check if map contains a segment.

Definition at line 52 of file segmentsmap.hpp.

6.37.4.3 template<int d, class X > bool SemSolver::SegmentsMap< d, X >::has (int const & *segment_id*) const [inline]

Check id an id is already used.

Definition at line 61 of file segmentsmap.hpp.

6.37.4.4 template<int d, class X > bool SemSolver::SegmentsMap< d, X >::haveOn (Point< d, X > const & *point*) const [inline]

Check if given point lies on a segment of the map or not.

Definition at line 180 of file segmentsmap.hpp.

6.37.4.5 template<int d, class X > int SemSolver::SegmentsMap< d, X >::id (Segment< d, X > const & *segment*) const [inline]

Get key corresponding to a segment

Parameters

segment	Segment to find, it is assumed to exist

Definition at line 100 of file segmentsmap.hpp.

6.37.4.6 template<int d, class X > void SemSolver::SegmentsMap< d, X >::insert (int const & *id*, Segment< d, X > const & *segment*) [inline]

Insert a id-segment pair in map If id already exists do nothing.

Definition at line 119 of file segmentsmap.hpp.

6.37.4.7 template<int d, class X > virtual bool SemSolver::SegmentsMap< d, X >::intersect (Segment< d, X > const & *segment*) const [inline, virtual]

Check if there is a segment in map intesecting specified segment.

Definition at line 150 of file segmentsmap.hpp.

6.37.4.8 template<int d, class X > virtual bool SemSolver::SegmentsMap< d, X >::intersectInteriorly (Segment< d, X > const & *segment*) const [inline, virtual]

Check if there is a segment in map intesecting specified segment not in a endpoint

Definition at line 160 of file segmentsmap.hpp.

6.37.4.9 template<int d, class X > bool SemSolver::SegmentsMap< d, X >::isConsistentWith (Segment< d, X > const & *segment*) const [inline]

Check if segment has consisten orientation with segments in map ie if there are no two segments with the same target nor source

Definition at line 170 of file segmentsmap.hpp.

6.37.4.10 template<int d, class X > void SemSolver::SegmentsMap< d, X >::modify (int const & *id*, Segment< d, X > const & *segment*) [inline]

Modify segment with specified id

Parameters

id	The key to find, it is assumed to exist in map
segment	Value to be assigned

Definition at line 133 of file segmentsmap.hpp.

6.37.4.11 template<int d, class X > void SemSolver::SegmentsMap< d, X >::remove (int const & *id*) [inline]

Remove entry with specified id if exists.

Definition at line 125 of file segmentsmap.hpp.

6.37.4.12 template<int d, class X > Segment<d,X> const& SemSolver::SegmentsMap< d, X >::segment (int const & *id*) const `[inline]`

Access segment by key.

Definition at line 88 of file segmentsmap.hpp.

6.37.4.13 template<int d, class X > Segment<d,X> SemSolver::SegmentsMap< d, X >::segmentFrom (Point< d, X > const & *source*) const `[inline]`

Get first segment in the map whith specified source.

Definition at line 70 of file segmentsmap.hpp.

6.37.4.14 template<int d, class X > int SemSolver::SegmentsMap< d, X >::segments ( ) const `[inline]`

Get size of map.

Definition at line 144 of file segmentsmap.hpp.

6.37.4.15 template<int d, class X > Segment<d,X> SemSolver::SegmentsMap< d, X >::segmentTo (Point< d, X > const & *target*) const `[inline]`

Get first segment in the map whith specified target.

Definition at line 79 of file segmentsmap.hpp.

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

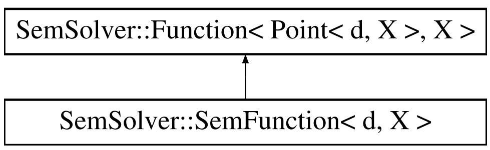
- segmentsmap.hpp

6.38 SemSolver::SemFunction< d, X > Class Template Reference

Class for handling spectral elements functions.

```
#include <semfunction.hpp>
```

Inheritance diagram for SemSolver::SemFunction< d, X >:



6.38.1 Detailed Description

template<int d, class X> class SemSolver::SemFunction< d, X >

Class for handling spectral elements functions. For each element of the partition of the domain geometry there are defined a tranformation to a canonical element, and a polynomial function

Definition at line 19 of file semfunction.hpp.

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

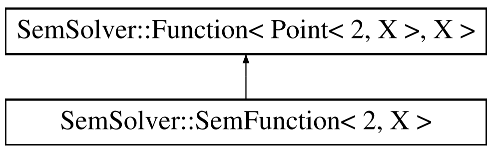
- semfunction.hpp

6.39 SemSolver::SemFunction< 2, X > Class Template Reference

Class for handling 2D spectral elements functions.

```
#include <semfunction.hpp>
```

Inheritance diagram for SemSolver::SemFunction< 2, X >:



Public Member Functions

- SemFunction (SemGeometry< 2, X > const &geometry, PolynomialFunctionVector const &polynomials, BilinearTransformationsVector const &maps)
- SemFunction (SemFunction const &sem_function)

  Copy constructor.

- PolynomialFunction< 2, X > const & polynomial (const unsigned &index) const

  Access polynomial transformed restriction on a subdomain element.

- void setPolynomialComponent (const unsigned &index, const unsigned &component, const Polynomial< X > &poly)

  Set polynomial transformed restriction component on a subdomain element.

- BilinearTransformation< X > const & map (const unsigned &index) const

  Access transformation from a subdomain element to canonical element.

- double evaluate (Point< 2, X > const &x) const

Compute function value at a point.

- Vector< X > evaluateRestrictionGradient (int const &element_index, Point< 2, X > const &P) const

  Compute gradient of function restriction on a subdomain element.

6.39.1 Detailed Description

template<class X> class SemSolver::SemFunction< 2, X >

Class for handling 2D spectral elements functions. For each element of the partition of the domain geometry there are defined a tranformation to a canonical element, and a polynomial function

Definition at line 27 of file semfunction.hpp.

6.39.2 Constructor & Destructor Documentation

6.39.2.1 template<class X > SemSolver::SemFunction< 2, X >::SemFunction (SemGeometry< 2, X > const & *geometry*, PolynomialFunctionVector const & *polynomials*, BilinearTransformationsVector const & *maps*) [inline]

Construct a spectral element function on a SemGeometry from a PloynomialFunction vector and a BilinearTranformation vector

Definition at line 41 of file semfunction.hpp.

6.39.2.2 template<class X > SemSolver::SemFunction< 2, X >::SemFunction (SemFunction< 2, X > const & *sem_function*) [inline]

Copy constructor.

Definition at line 51 of file semfunction.hpp.

6.39.3 Member Function Documentation

6.39.3.1 template<class X > double SemSolver::SemFunction< 2, X >::evaluate (Point< 2, X > const & *x*) const [inline, virtual]

Compute function value at a point.

Reimplemented from SemSolver::Function< Point< 2, X >, X >.

Definition at line 90 of file semfunction.hpp.

6.39.3.2 template< class X > Vector<X> SemSolver::SemFunction< 2, X >::evaluateRestrictionGradient (int const & *element_index*, Point< 2, X > const & *P*) const [inline]

Compute gradient of function restriction on a subdomain element.

Definition at line 100 of file semfunction.hpp.

6.39.3.3 template< class X > BilinearTransformation<X> const& SemSolver::SemFunction< 2, X >::map (const unsigned & *index*) const [inline]

Access transformation from a subdomain element to canonical element.

Definition at line 80 of file semfunction.hpp.

6.39.3.4 template< class X > PolynomialFunction<2,X> const& SemSolver::SemFunction< 2, X >::polynomial (const unsigned & *index*) const [inline]

Access polynomial transformed restriction on a subdomain element.

Definition at line 58 of file semfunction.hpp.

6.39.3.5 template< class X > void SemSolver::SemFunction< 2, X >::setPolynomialComponent (const unsigned & *index*, const unsigned & *component*, const Polynomial< X > & *poly*) [inline]

Set polynomial transformed restriction component on a subdomain element.

Definition at line 68 of file semfunction.hpp.

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

- semfunction.hpp

6.40 SemSolver::SemGeometry< d, X > Class Template Reference

Class for describing the geometry of a SemProblem.

`#include <semgeometry.hpp>`

6.40.1 Detailed Description

template<int d, class X> class SemSolver::SemGeometry< d, X >

Class for describing the geometry of a SemProblem. Consist of the description of the whole domain, of a set of its subdomains and of its boundary edges

Parameters

d	Dimension of the space
X	Must be a type for which operations +, -, * and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of *

Definition at line 18 of file semgeometry.hpp.

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

- semgeometry.hpp

6.41 SemSolver::SemGeometry< 2, X > Class Template Reference

```
#include <semgeometry.hpp>
```

Public Member Functions

- PSLG< X > const & domain () const

  Access geometry description.

- Polygonation< 2, X > const & subDomains () const

  Access geometry partition.

- void setDomain (PSLG< X > const &domain)

  Set geometry description.

- void setSubDomains (Polygonation< 2, X > const &sub_domains)

  Set geometry partition.

- bool contains (const Point< 2, X > &point) const

  Check if a point doesn't lie outside the domain.

6.41.1 Detailed Description

template<class X> class SemSolver::SemGeometry< 2, X >

Definition at line 30 of file semgeometry.hpp.

6.41.2 Member Function Documentation

6.41.2.1 template<class X > bool SemSolver::SemGeometry< 2, X >::contains (const Point< 2, X > & *point*) const [inline]

Check if a point doesn't lie outside the domain.

Definition at line 63 of file semgeometry.hpp.

6.41.2.2 template<class X > PSLG<X> const& SemSolver::SemGeometry< 2, X >::domain ( ) const [inline]

Access geometry description.

Definition at line 39 of file semgeometry.hpp.

6.41.2.3 template<class X > void SemSolver::SemGeometry< 2, X >::setDomain (PSLG< X > const & *domain*) [inline]

Set geometry description.

Definition at line 51 of file semgeometry.hpp.

6.41.2.4 template<class X > void SemSolver::SemGeometry< 2, X >::setSubDomains (Polygonation< 2, X > const & *sub_domains*) [inline]

Set geometry partition.

Definition at line 57 of file semgeometry.hpp.

6.41.2.5 template<class X > Polygonation<2,X> const& SemSolver::SemGeometry< 2, X >::subDomains ( ) const [inline]

Access geometry partition.

Definition at line 45 of file semgeometry.hpp.

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

- semgeometry.hpp

6.42 SemSolver::SemParameters< X > Class Template Reference

```
#include <semparameters.hpp>
```

Public Member Functions

- SemParameters ()

Default constructor.

- SemParameters (int const &degree, X const &tolerance, X const &penality)

  Construct Parameters from degree, tolerance and penality values.
- int const & degree () const

  Access degree parameter.
- X const & tolerance () const

  Access tolerance parameter.
- X const & penality () const

  Access penality parameter.
- void setDegree (const int &d)

  Set degree parameter.
- void setTolerance (const X &t)

  Set tolerance parameter.
- void setPenality (const X &p)

  Set penality parameter.

6.42.1 Detailed Description

template<class X> class SemSolver::SemParameters< X >

Class used for stroing the parameters of the spectral element method It consist of polynomial degree to be used, tolerance and penality coefficient

Definition at line 15 of file semparameters.hpp.

6.42.2 Constructor & Destructor Documentation

6.42.2.1 template<class X> SemSolver::SemParameters< X >::SemParameters ( ) [inline]

Default constructor.

Definition at line 23 of file semparameters.hpp.

6.42.2.2 template<class X> SemSolver::SemParameters< X >::SemParameters (int const & *degree*, X const & *tolerance*, X const & *penality*) [inline]

Construct Parameters from degree, tolerance and penality values.

Definition at line 27 of file semparameters.hpp.

6.42.3 Member Function Documentation

6.42.3.1 template<class X> int const& SemSolver::SemParameters< X >::degree ( ) const [inline]

Access degree parameter.

Definition at line 36 of file semparameters.hpp.

6.42.3.2 template<class X> X const& SemSolver::SemParameters< X >::penality ( ) const [inline]

Access penality parameter.

Definition at line 48 of file semparameters.hpp.

6.42.3.3 template<class X> void SemSolver::SemParameters< X >::setDegree (const int & *d*) [inline]

Set degree parameter.

Definition at line 54 of file semparameters.hpp.

6.42.3.4 template<class X> void SemSolver::SemParameters< X >::setPenality (const X & *p*) [inline]

Set penality parameter.

Definition at line 66 of file semparameters.hpp.

6.42.3.5 template<class X> void SemSolver::SemParameters< X >::setTolerance (const X & *t*) [inline]

Set tolerance parameter.

Definition at line 60 of file semparameters.hpp.

6.42.3.6 template<class X> X const& SemSolver::SemParameters< X >::tolerance ( ) const [inline]

Access tolerance parameter.

Definition at line 42 of file semparameters.hpp.

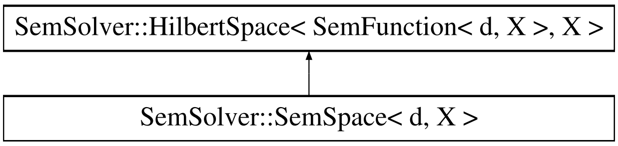
The documentation for this class was generated from the following file:

- semparameters.hpp

6.43 SemSolver::SemSpace< d, X > Class Template Reference

```
#include <semspace.hpp>
```

Inheritance diagram for SemSolver::SemSpace< d, X >:



6.43.1 Detailed Description

template< int d, class X > class SemSolver::SemSpace< d, X >

Class used for modeling the mathematical concept of Functional Space of Spectral Element Functions

Parameters

d	Dimension of the domain
X	Must be a type for which operations +, -, $*$ and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of $*$

Definition at line 32 of file semspace.hpp.

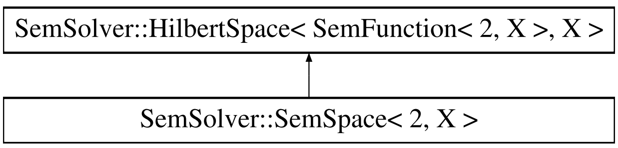
The documentation for this class was generated from the following file:

- semspace.hpp

6.44 SemSolver::SemSpace< 2, X > Class Template Reference

```
#include <semspace.hpp>
```

Inheritance diagram for SemSolver::SemSpace< 2, X >:



Classes

- class Element

  Class for handling members of the space as Fourier coefficients.

- class Node

Public Types

- typedef MultiIndex< 2 >::less Index2Order
- typedef MultiIndex< 3 >::less Index3Order
- typedef std::vector< Node > NodesVector
- typedef std::vector< SemFunction< 2, X > * > SemFunctionsVector
- typedef PointsMap< 2, X, int > NodesMap
- typedef NodesMap::ConstIterator NodeConstIterator
- typedef std::map< MultiIndex< 3 >, int, Index3Order > ElementsMap
- typedef ElementsMap::const_iterator ElementConstIterator
- typedef std::map< MultiIndex< 2 >, MultiIndex< 3 >, Index2Order > Border-sMap
- typedef std::vector< int > BordersVector
- typedef std::map< MultiIndex< 3 >, double, Index3Order > WeightsMap
- typedef Polygonation< 2, X >::Element SubDomain

Public Member Functions

- SemSpace (SemGeometry< 2, X > const &geometry, SemParameters< X > const &parameters)

  Construct SpectralElement Spece on Spectral Element Geometry and Parameters.

- unsigned nodes () const

  Get number of space nodes.

- const Node & node (const unsigned &index) const

  Access index-th node.

- int subDomains () const

  Get number of subdomains.

- int const & subDomainIndex (MultiIndex< 3 > const &index) const

  Get node index correrponding to an element multindex.

- Node const & subDomainNode (MultiIndex< 3 > const &index) const

  Get node correrponding to an element multindex.

- double const & subDomainWeight (MultiIndex< 3 > const &index) const

  Get weight index correrponding to an element multindex.

- unsigned borders () const

  Get number of geometry borders.

- int border (int const &i) const

  Get index of i-th border.

- int borderId (int const &border) const

  Get id of border whith index border.

- MultiIndex< 3 > const & borderSubDomainIndex (MultiIndex< 2 > const &index) const

  Get element index corresponding to a border index.

- int const & borderIndex (MultiIndex< 2 > const &index) const

  Get node index corresponding to a border index.

- Node const & borderNode (MultiIndex< 2 > const &index) const

  Get node corresponding to a border index.

- double const & borderWeight (MultiIndex< 2 > const &index) const

  Get weight corresponding to a border index.

- double scalarProduct (Element first, Element second)

  NOT YET IMPLEMENTED.

- SemFunction< 2, X > const * baseFunction (const unsigned &index) const

  Access base function.

- int const & degree () const

  Get space degree.

Protected Attributes

- SemParameters< X > const & _parameters
- SemGeometry< 2, X > const & _geometry

6.44.1 Detailed Description

template<class X> class SemSolver::SemSpace< 2, X >

Class used for modeling the mathematical concept of Functional Space of Spectral Element Functions on 2D Euclidean space

Parameters

X	Must be a type for which operations +, -, $*$ and / are defined with semantics (approximately) corresponding to those of a field in a mathematical sense. Note that, strictly speaking, the built-in type int does not fullfil the requirements on a field type, since ints correspond to elements of a ring rather than a field, especially operation / is not the inverse of $*$

Definition at line 45 of file semspace.hpp.

6.44.2 Member Typedef Documentation

6.44.2.1 template<class X > typedef std::map< MultiIndex<2>, MultiIndex<3>, Index2Order > SemSolver::SemSpace< 2, X >::BordersMap

Definition at line 182 of file semspace.hpp.

6.44.2.2 template<class X > typedef std::vector<int> SemSolver::SemSpace< 2, X >::BordersVector

Definition at line 183 of file semspace.hpp.

6.44.2.3 template<class X > typedef ElementsMap::const_iterator SemSolver::SemSpace< 2, X >::ElementConstIterator

Definition at line 180 of file semspace.hpp.

6.44.2.4 template<class X > typedef std::map< MultiIndex<3>, int, Index3Order > SemSolver::SemSpace< 2, X >::ElementsMap

Definition at line 179 of file semspace.hpp.

6.44.2.5 template<class X > typedef MultiIndex<2>::less SemSolver::SemSpace< 2, X >::Index2Order

Definition at line 173 of file semspace.hpp.

6.44.2.6 template<class X > typedef MultiIndex<3>::less SemSolver::SemSpace< 2, X >::Index3Order

Definition at line 174 of file semspace.hpp.

6.44.2.7 template<class X > typedef NodesMap::ConstIterator SemSolver::SemSpace< 2, X >::NodeConstIterator

Definition at line 178 of file semspace.hpp.

6.44.2.8 template<class X > typedef PointsMap<2, X, int> SemSolver::SemSpace< 2, X >::NodesMap

Definition at line 177 of file semspace.hpp.

6.44.2.9 template<class X > typedef std::vector<Node> SemSolver::SemSpace< 2, X >::NodesVector

Definition at line 175 of file semspace.hpp.

6.44.2.10 template<class X > typedef std::vector< SemFunction<2,X> * > SemSolver::SemSpace< 2, X >::SemFunctionsVector

Definition at line 176 of file semspace.hpp.

6.44.2.11 template<class X > typedef Polygonation<2,X>::Element SemSolver::SemSpace< 2, X >::SubDomain

Definition at line 185 of file semspace.hpp.

6.44.2.12 template<class X > typedef std::map< MultiIndex<3>, double, Index3Order > SemSolver::SemSpace< 2, X >::WeightsMap

Definition at line 184 of file semspace.hpp.

6.44.3 Constructor & Destructor Documentation

6.44.3.1 template<class X > SemSolver::SemSpace< 2, X >::SemSpace (SemGeometry< 2, X > const & *geometry*, SemParameters< X > const & *parameters*) [inline]

Construct SpectralElement Spece on Spectral Element Geometry and Parameters.

Definition at line 256 of file semspace.hpp.

6.44.4 Member Function Documentation

6.44.4.1 template<class X > SemFunction<2,X> const* SemSolver::SemSpace< 2, X >::baseFunction (const unsigned & *index*) const [inline]

Access base function.

Definition at line 730 of file semspace.hpp.

6.44.4.2 template<class X > int SemSolver::SemSpace< 2, X >::border (int const & *i*) const [inline]

Get index of i-th border.

Definition at line 677 of file semspace.hpp.

6.44.4.3 template<class X > int SemSolver::SemSpace< 2, X >::borderId (int const & *border*) const [inline]

Get id of border whith index border.

Definition at line 683 of file semspace.hpp.

6.44.4.4 template<class X > int const& SemSolver::SemSpace< 2, X >::borderIndex (MultiIndex< 2 > const & *index*) const [inline]

Get node index corresponding to a border index.

Definition at line 705 of file semspace.hpp.

6.44.4.5 template<class X > Node const& SemSolver::SemSpace< 2, X >::borderNode (MultiIndex< 2 > const & *index*) const [inline]

Get node corresponding to a border index.

Definition at line 711 of file semspace.hpp.

6.44.4.6 template<class X > unsigned SemSolver::SemSpace< 2, X >::borders ( ) const [inline]

Get number of geometry borders.

Definition at line 671 of file semspace.hpp.

6.44.4.7 template<class X > MultiIndex<3> const& SemSolver::SemSpace< 2, X >::borderSubDomainIndex (MultiIndex< 2 > const & *index*) const [inline]

Get element index corresponding to a border index.

Definition at line 692 of file semspace.hpp.

6.44.4.8 template<class X > double const& SemSolver::SemSpace< 2, X >::borderWeight (MultiIndex< 2 > const & *index*) const [inline]

Get weight corresponding to a border index.

Definition at line 717 of file semspace.hpp.

6.44.4.9 template<class X > int const& SemSolver::SemSpace< 2, X >::degree ( ) const [inline]

Get space degree.

Definition at line 741 of file semspace.hpp.

6.44.4.10 template<class X > const Node& SemSolver::SemSpace< 2, X >::node (const unsigned & *index*) const [inline]

Access index-th node.

Definition at line 625 of file semspace.hpp.

6.44.4.11 template<class X > unsigned SemSolver::SemSpace< 2, X >::nodes ( ) const [inline]

Get number of space nodes.

Definition at line 619 of file semspace.hpp.

6.44.4.12 template<class X > double SemSolver::SemSpace< 2, X >::scalarProduct (Element *first*, Element *second*) [inline]

NOT YET IMPLEMENTED.

Definition at line 723 of file semspace.hpp.

6.44.4.13 template<class X > int const& SemSolver::SemSpace< 2, X >::subDomainIndex (MultiIndex< 3 > const & *index*) const [inline]

Get node index correrponding to an element multindex.

Definition at line 639 of file semspace.hpp.

6.44.4.14 template<class X > Node const& SemSolver::SemSpace< 2, X >::subDomainNode (MultiIndex< 3 > const & *index*) const [inline]

Get node correrponding to an element multindex.

Definition at line 652 of file semspace.hpp.

6.44.4.15 template<class X > int SemSolver::SemSpace< 2, X >::subDomains ( ) const [inline]

Get number of subdomains.

Definition at line 633 of file semspace.hpp.

6.44.4.16 template<class X > double const& SemSolver::SemSpace< 2, X >::subDomainWeight (MultiIndex< 3 > const & *index*) const [inline]

Get weight index correrponding to an element multindex.

Definition at line 658 of file semspace.hpp.

6.44.5 Member Data Documentation

6.44.5.1 template<class X > SemGeometry<2,X> const& SemSolver::SemSpace< 2, X >::_geometry [protected]

Definition at line 190 of file semspace.hpp.

6.44.5.2 template<class X > SemParameters<X> const& SemSolver::SemSpace< 2, X >::_parameters [protected]

Definition at line 189 of file semspace.hpp.

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

- semspace.hpp

6.45 SemSolver::Vector< X > Class Template Reference

`#include <vector.hpp>`

Public Member Functions

- Vector ()

  Construct empty vector.

- Vector (TNT_array_1d const &vector)

  Construct vector from TNT::Array1D.

- Vector (int dimension)

  Construct vector of dimension elements.

- Vector (int dimension, X const &value)
- template<int d>
  Vector (Point< d, X > const &point)

  Construct vector from point coordinates.

- int rows () const

  Get vector dimension.

6.45.1 Detailed Description

template<class X> class SemSolver::Vector< X >

Definition at line 18 of file vector.hpp.

6.45.2 Constructor & Destructor Documentation

6.45.2.1 template<class X> SemSolver::Vector< X >::Vector ( ) [inline]

Construct empty vector.

Definition at line 24 of file vector.hpp.

6.45.2.2 template<class X> SemSolver::Vector< X >::Vector (TNT_array_1d const & *vector*) [inline]

Construct vector from TNT::Array1D.

Definition at line 27 of file vector.hpp.

6.45.2.3 template<class X> SemSolver::Vector< X >::Vector (int *dimension*) [inline]

Construct vector of dimension elements.

Definition at line 30 of file vector.hpp.

6.45.2.4 template<class X> SemSolver::Vector< X >::Vector (int *dimension*, X const & *value*) [inline]

Definition at line 31 of file vector.hpp.

6.45.2.5 template<class X> template<int d> SemSolver::Vector< X >::Vector (Point< d, X > const & *point*) [inline]

Construct vector from point coordinates.

Definition at line 35 of file vector.hpp.

6.45.3 Member Function Documentation

6.45.3.1 template<class X> int SemSolver::Vector< X >::rows ( ) const [inline]

Get vector dimension.

Definition at line 43 of file vector.hpp.

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

- vector.hpp

6.46 SemSolver::PSLG< X >::Vertex Struct Reference

PSLG Vertex struct.

`#include <pslg.hpp>`

Public Attributes

- int number
- X x
- X y
- double ∗ attributes
- int marker

6.46.1 Detailed Description

template<class X> struct SemSolver::PSLG< X >::Vertex

PSLG Vertex struct. Contain coordinates x and y, vertex number, attributes - which are typically values of physical quantities (such as mass or conductivity) associated with the nodes of a finite element mesh and boundary markers. Boundary markers are used to identify boundary vertices and vertices resting on PSLG segments.

Definition at line 34 of file pslg.hpp.

6.46.2 Member Data Documentation

6.46.2.1 template<class X> double∗ SemSolver::PSLG< X >::Vertex::attributes

Definition at line 39 of file pslg.hpp.

6.46.2.2 template<class X> int SemSolver::PSLG< X >::Vertex::marker

Definition at line 40 of file pslg.hpp.

6.46.2.3 template<class X> int SemSolver::PSLG< X >::Vertex::number

Definition at line 36 of file pslg.hpp.

6.46.2.4 template<class X> X SemSolver::PSLG< X >::Vertex::x

Definition at line 37 of file pslg.hpp.

6.46.2.5 template<class X> X SemSolver::PSLG< X >::Vertex::y

Definition at line 38 of file pslg.hpp.

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

- pslg.hpp

Chapter 7

File Documentation

7.1 archive.hpp File Reference

```
#include <QFile>
#include <QString>
#include <QStringList>
#include <QTemporaryFile>
#include <QTextStream>
#include <SemSolver/IO/carchive>
```

Classes

- class SemSolver::IO::Archive

  Class for handling tar uncompressed archives.

Namespaces

- namespace SemSolver

  Project main namespace.

- namespace SemSolver::IO

  Namespace for Input/Output operations on SemSolver Classes.

7.2 archive.hpp

```
#ifndef IO_ARCHIVE_HPP
```

```
#define IO_ARCHIVE_HPP

#include <QFile>
#include <QString>
#include <QStringList>
#include <QTemporaryFile>
#include <QTextStream>

#include  <SemSolver/IO/carchive>

namespace SemSolver
{
    namespace IO
    {
        class Archive
        {
            enum Status
            {
                CLOSED,
                OPENREAD,
                OPENWRITE
            };

            archive::archive *archive;
            QFile            *file;
            Status           status;

        public:
            Archive(QFile *file);

            bool openRead();

            bool openWrite();

            bool closeRead();

            bool closeWrite();

            QStringList entries();

            template<class T>
            bool addValue(T const &value,
                          QString const &name)
            {
            #if SEMDEBUG
                if(status!=OPENWRITE)
                    qFatal("You must openWrite archive before adding entries");
            #endif
                if(!file->exists())
                    return false;
                QTemporaryFile *temp_file = new QTemporaryFile();
                if(!temp_file->open(QIODevice::WriteOnly | QIODevice::Text))
                    return false;
                QTextStream out(temp_file);
                out << value;
                temp_file->close();
                addFile(temp_file, name);
                delete temp_file;
                return true;
            };

            bool addFile(QFile *file);
```

```
            bool addFile(QFile *file,
                         QString const &name);

            bool extractFile(QString const &name);

            bool extractFile(QString const &name, QFile *file);
        };
    };
};

#endif // IO_ARCHIVE_HPP
```

7.3 bilineartransformation.hpp File Reference

```
#include <cmath>
#include <SemSolver/homeomorphism.hpp>
#include <SemSolver/matrix.hpp>
#include <SemSolver/point.hpp>
#include <SemSolver/polygon.hpp>
```

Classes

- class SemSolver::BilinearTransformation< X >

  Class representing a bilinear tranformation of 2D euclidean space.

Namespaces

- namespace SemSolver

  Project main namespace.

7.4 bilineartransformation.hpp

```
#ifndef BILINEARTRANSFORMATION_HPP
#define BILINEARTRANSFORMATION_HPP

namespace SemSolver
{
    template<class X>
    class BilinearTransformation;
};

#include <cmath>

#include <SemSolver/homeomorphism.hpp>
#include <SemSolver/matrix.hpp>
```

```cpp
#include <SemSolver/point.hpp>
#include <SemSolver/polygon.hpp>

namespace SemSolver
{

    template<class X>
    class BilinearTransformation
        : public Homeomorphism< Point<2,X>, Point<2,X> >
    {
        // domain
        Polygon<2,X> _omega;

        // coefficients
        X _Px, _Qx, _Rx, _Sx;
        X _Py, _Qy, _Ry, _Sy;
        X _PQ, _PR, _PS, _QR, _QS, _RS;

        // tolerance for inverse evaluation
        X _tolerance;

    public:

        BilinearTransformation();

        BilinearTransformation(Point<2,X> const &A,
                               Point<2,X> const &B,
                               Point<2,X> const &C,
                               Point<2,X> const &D);

        BilinearTransformation(Polygon<2,X> const &omega,
                               X const &tolerance);

        ~BilinearTransformation();

        void setOmega(Polygon<2,X> const &omega);

        void setTolerance(X const &tolerance);

        inline Polygon<2,X> const &omega() const;

        inline X const &tolerance() const;

        Point<2,X> evaluate(Point<2,X> const &point) const;

        Point<2,X> evaluateInverse(Point<2,X> const &point) const;

        inline X evaluateJacobianDeterminant(Point<2,X> const &point) const;

        Matrix<double>
                evaluateTransposeInverseJacobian(Point<2,X> const &point) const;
    };
};

template<class X>
SemSolver::BilinearTransformation<X>::BilinearTransformation()
{
    _Px =  0., _Py =  0.;
    _Qx = -1., _Qy =  0.;
    _Rx =  0., _Ry = -1.;
    _Sx =  0., _Sy =  0.;
    _PQ =  0., _PR =  0.;
```

```
    _PS =  0., _QR =  1.;
    _QS =  0., _RS =  0.;
    _tolerance = 0;
};

template<class X>
SemSolver::BilinearTransformation<X>::BilinearTransformation(Point<2,X> const &A,

                                                             Point<2,X> const &B,

                                                             Point<2,X> const &C,

                                                             Point<2,X> const &D)

{
    _Px = (A.x()+B.x()+C.x()+D.x())/4.;
    _Py = (A.y()+B.y()+C.y()+D.y())/4.;
    _Qx = (A.x()-B.x()-C.x()+D.x())/4.;
    _Qy = (A.y()-B.y()-C.y()+D.y())/4.;
    _Rx = (A.x()+B.x()-C.x()-D.x())/4.;
    _Ry = (A.y()+B.y()-C.y()-D.y())/4.;
    _Sx = (A.x()-B.x()+C.x()-D.x())/4.;
    _Sy = (A.y()-B.y()+C.y()-D.y())/4.;

    _PQ = _Px*_Qy - _Py*_Qx;
    _PR = _Px*_Ry - _Py*_Rx;
    _PS = _Px*_Sy - _Py*_Sx;
    _QR = _Qx*_Ry - _Qy*_Rx;
    _QS = _Qx*_Sy - _Qy*_Sx;
    _RS = _Rx*_Sy - _Ry*_Sx;

    _tolerance = 0.;
};

template<class X>
SemSolver::BilinearTransformation<X>::BilinearTransformation(Polygon<2,X> const &
omega,
                                                             X const &tolerance)
{
    setOmega(omega);
    setTolerance(tolerance);
};

template<class X>
SemSolver::BilinearTransformation<X>::~BilinearTransformation()
{
}

template<class X>
void SemSolver::BilinearTransformation<X>::setOmega(Polygon<2,X> const &omega)
{
#ifdef SEMDEBUG
    if(omega.size()!=4)
        qFatal("SemSolver::BilinearTransformation::setOmega - ERROR : omega must
be a qu"\
               "adrangle.");
    if(!omega.is_simple())
        qFatal("SemSolver::BilinearTransformation::setOmega - ERROR : omega must
be simp"\
               "le.");
    if(!omega.is_convex())
        qFatal("SemSolver::BilinearTransformation::setOmega - ERROR : omega must
```

```
be conv"\
                "ex.");
    if(!omega.is_counterclockwise_oriented())
        qFatal("SemSolver::BilinearTransformation::setOmega - ERROR : omega must
be coun"\
                "terclockwise oriented.");
#endif // SEMDEBUG
    _omega = omega;

    Point<2,X> const &A = omega.vertex(0);
    Point<2,X> const &B = omega.vertex(1);
    Point<2,X> const &C = omega.vertex(2);
    Point<2,X> const &D = omega.vertex(3);

    _Px = (A.x()+B.x()+C.x()+D.x())/4.;
    _Py = (A.y()+B.y()+C.y()+D.y())/4.;
    _Qx = (A.x()-B.x()-C.x()+D.x())/4.;
    _Qy = (A.y()-B.y()-C.y()+D.y())/4.;
    _Rx = (A.x()+B.x()-C.x()-D.x())/4.;
    _Ry = (A.y()+B.y()-C.y()-D.y())/4.;
    _Sx = (A.x()-B.x()+C.x()-D.x())/4.;
    _Sy = (A.y()-B.y()+C.y()-D.y())/4.;

    _PQ = _Px*_Qy - _Py*_Qx;
    _PR = _Px*_Ry - _Py*_Rx;
    _PS = _Px*_Sy - _Py*_Sx;
    _QR = _Qx*_Ry - _Qy*_Rx;
    _QS = _Qx*_Sy - _Qy*_Sx;
    _RS = _Rx*_Sy - _Ry*_Sx;
};

template<class X>
void SemSolver::BilinearTransformation<X>::setTolerance(X const &tolerance)
{
#ifdef SEMDEBUG
    if(tolerance<0)
        qFatal("SemSolver::BilinearTransformation::setTolerance - ERROR : toleran
ce must"\
                "be non negative.");
#endif // SEMDEBUG
    _tolerance = tolerance;
};

template<class X>
inline SemSolver::Polygon<2,X> const &
SemSolver::BilinearTransformation<X>::omega() const
{
    return _omega;
}


template<class X>
inline X const &SemSolver::BilinearTransformation<X>::tolerance() const
{
    return _tolerance;
};

template<class X>
SemSolver::Point<2,X>
        SemSolver::BilinearTransformation<X>::evaluate(Point<2,X> const &point) c
onst
{
```

```
    X x = _Px - _Qx*point.x() - _Rx*point.y() + _Sx*point.x()*point.y();
    X y = _Py - _Qy*point.x() - _Ry*point.y() + _Sy*point.x()*point.y();
    return Point<2,X>(x,y);
};



template<class X>
SemSolver::Point<2,X>
        SemSolver::BilinearTransformation<X>::evaluateInverse(
                Point<2,X> const &point) const
{
    X x, y;
    X XQ = point.x()*_Qy - point.y()*_Qx;
    X XR = point.x()*_Ry - point.y()*_Rx;
    X XS = point.x()*_Sy - point.y()*_Sx;
    if( std::abs(_QS)>_tolerance && std::abs(_RS)>_tolerance )
    {
        // Omega is not a parallelogram nor a trapezoid
        X psi = pow(XS-_PS+_QR, 2.) + 2.*(XR*_QS+XQ*_RS-XS*_QR) - \
                4.*_PQ*_RS;
        X n0 = XS-_PS;
        X n1 = _QR - sqrt(psi);
        X d0 = 2.*_QS;
        X d1 = 2.*_RS;
        x = -(n0-n1)/d0;
        y = -(n0+n1)/d1;
    }
    else if(std::abs(_RS)>_tolerance) // QS trascurabile
    {
        // Omega is a vertical trapezoid
        x = _Qx*(XR-_PR)/(_Sx*(XQ-_PQ)-_Qx*_QR);
        y = (XQ-_PQ)/_QR;
    }
    else if(std::abs(_QS)>_tolerance)
    {
        // Omega is a horizontal trapezoid
        x = (_PR-XR)/_QR;
        y = _Rx*(_PQ-XQ)/(_Sx*(_PR-XR)-_Rx*_QR);
    }
    else
    {
        // Omega is a parallelogram
        x = (_PR-XR)/_QR;
        y = (XQ-_PQ)/_QR;
    }
    return Point<2,X>(x,y);
};

template<class X>
inline X SemSolver::BilinearTransformation<X>::evaluateJacobianDeterminant(
        Point<2,X> const &point) const
{
    return _QR - point.x()*_QS + point.y()*_RS;
};

template<class X>
SemSolver::Matrix<double>
        SemSolver::BilinearTransformation<X>::evaluateTransposeInverseJacobian(
                Point<2,X> const &point) const
{
    Matrix<double> tiJ(2,2,1/evaluateJacobianDeterminant(point));
```

```
    tiJ[0][0] *= point.x()*_Sy - _Ry;
    tiJ[0][1] *= _Rx - point.x()*_Sx;
    tiJ[1][0] *= _Qy - point.y()*_Sy;
    tiJ[1][1] *= point.y()*_Sx - _Qx;
    return tiJ;
};

#endif // BILINEARTRANSFORMATION_HPP
```

7.5 boundaryconditions.hpp File Reference

```
#include <QString>
#include <QStringList>
#include <map>
#include <vector>
#include <SemSolver/function.hpp>
#include <SemSolver/point.hpp>
#include <SemSolver/semgeometry.hpp>
```

Classes

- class SemSolver::BoundaryConditions< d, X >

  Class for handling boundary conditions on a SemGeometry.

- class SemSolver::BoundaryConditions< 2, X >

  Class for handling boundary conditions on a 2D SemGeometry.

Namespaces

- namespace SemSolver

  Project main namespace.

7.6 boundaryconditions.hpp

```
#ifndef BOUNDARYCONDITIONS_HPP
#define BOUNDARYCONDITIONS_HPP

#include <QString>
#include <QStringList>

namespace SemSolver
{
    template<int d, class X>
```

```
    class BoundaryConditions;
};

#include <map>
#include <vector>

#include <SemSolver/function.hpp>
#include <SemSolver/point.hpp>
#include <SemSolver/semgeometry.hpp>

namespace SemSolver
{

    template<int d, class X>
    class BoundaryConditions
    {
    };


    template<class X>
    class BoundaryConditions<2,X>
    {
    public:
        enum Type
        {
            UNDEFINED,
            DIRICHLET,
            NEUMANN,
            ROBIN
        };

        typedef Function< Point<2,X>, X > const * FunctionPtr;
        typedef std::map<int, FunctionPtr> FunctionsMap;
        typedef std::map<int, Type > TypesMap;
        typedef typename FunctionsMap::const_iterator FunctionConstIterator;

    private:
        // types map
        TypesMap _types;

        // ceofficients
        FunctionsMap _g;
        FunctionsMap _h;
        FunctionsMap _gamma;
        FunctionsMap _r;

    public:

        inline BoundaryConditions();

        BoundaryConditions(TypesMap const &types,
                           FunctionsMap const *g,
                           FunctionsMap const *h,
                           FunctionsMap const *gamma,
                           FunctionsMap const *r);

        template<class Y>
        BoundaryConditions(BoundaryConditions<2,Y> const &conditions);

        inline int conditions() const;

        inline Type const &borderType(int const &index) const;
```

```

        inline FunctionPtr dirichletData(int const &index) const;

        inline FunctionPtr neumannData(int const &index) const;

        inline FunctionPtr robinCoefficient(int const &index) const;

        inline FunctionPtr robinData(int const &index) const;

        void setBorder(int const &index,
                       Type const &type,
                       FunctionPtr const f1=0,
                       FunctionPtr const f2=0);

        QStringList labels() const;

        QStringList mmls() const;

        void clear();
    };
};

template<class X>
SemSolver::BoundaryConditions<2, X>::BoundaryConditions()
{
};

template<class X>
SemSolver::BoundaryConditions<2, X>::BoundaryConditions(TypesMap const &types,
                                                         FunctionsMap const *g,
                                                         FunctionsMap const *h,
                                                         FunctionsMap const *gamma,
                                                         FunctionsMap const *r)
{
    _types(types);
    for(typename TypesMap::const_iterator it=types.begin(); it!=types.end(); ++it)
    {
        switch(it->second)
        {
        case DIRICHLET:
#ifdef SEMDEBUG
            if(!g)
                qFatal("SemSolver::BoundaryConditions::BoundaryConditions - ERROR - tryi"\
                       "ng to access null pointer g.");
#endif
            _g[it->first] = g[it->first];
            break;
        case NEUMANN:
#ifdef SEMDEBUG
            if(!h)
                qFatal("SemSolver::BoundaryConditions::BoundaryConditions - ERROR - tryi"\
                       "ng to access null pointer h.");
#endif
            _h[it->first] = h[it->first];
            break;
        case ROBIN:
#ifdef SEMDEBUG
            if(!gamma)
```

```
                qFatal("SemSolver::BoundaryConditions::BoundaryConditions - ERROR
 - tryi"\
                        "ng to access null pointer gamma.");
            if(!r)
                qFatal("SemSolver::BoundaryConditions::BoundaryConditions - ERROR
 - tryi"\
                        "ng to access null pointer r.");
#endif
            _gamma[it->first] = gamma[it->first];
            _r[it->first] = r[it->first];
            break;
        default:
            _types[it->first] = UNDEFINED;
        }
    }
};

template<class X>
template<class Y>
SemSolver::BoundaryConditions<2, X>::BoundaryConditions(
        BoundaryConditions<2, Y> const &conditions)
{
    _types = conditions._types;
    _g = conditions._g;
    _h = conditions._h;
    _gamma = conditions._gamma;
    _r = conditions._r;
};

template<class X>
inline int SemSolver::BoundaryConditions<2, X>::conditions() const
{
    return _types.size();
};

template<class X>
inline typename SemSolver::BoundaryConditions<2, X>::Type const &
        SemSolver::BoundaryConditions<2, X>::borderType(int const &index) const
{
    typename TypesMap::const_iterator it = _types.find(index);
#ifdef SEMDEBUG
    if(it==_types.end() )
        qFatal("SemSolver::BoundaryConditions::borderType - ERROR : no border at
index.");
#endif
    return it->second;
};

template<class X>
inline SemSolver::Function< SemSolver::Point<2, X>, X > const *
        SemSolver::BoundaryConditions<2, X>::dirichletData(int const &index) cons
t
{
    FunctionConstIterator it = _g.find(index);
#ifdef SEMDEBUG
    if(it==_g.end() )
        qFatal("SemSolver::BoundaryConditions::dirichletData - ERROR : the border
 at ind"\
                "ex is not Dirichlet.");
#endif
    return it->second;
};
```

```
template<class X>
inline SemSolver::Function< SemSolver::Point<2, X>, X > const *
        SemSolver::BoundaryConditions<2, X>::neumannData(int const &index) const
{
    FunctionConstIterator it = _h.find(index);
#ifdef SEMDEBUG
    if(it==_h.end() )
        qFatal("SemSolver::BoundaryConditions::neumannData - ERROR : the border at index"\
               " is not Neumann.");
#endif
    return it->second;
};

template<class X>
inline SemSolver::Function< SemSolver::Point<2, X>, X > const *
        SemSolver::BoundaryConditions<2, X>::robinCoefficient(int const &index) const
{
    FunctionConstIterator it = _gamma.find(index);
#ifdef SEMDEBUG
    if(it==_gamma.end() )
        qFatal("SemSolver::BoundaryConditions::robinCoefficient - ERROR : the border at "\
               "index is not Robin.");
#endif
    return it->second;
};

template<class X>
inline SemSolver::Function< SemSolver::Point<2, X>, X > const *
        SemSolver::BoundaryConditions<2, X>::robinData(int const &index) const
{
    FunctionConstIterator it = _r.find(index);
#ifdef SEMDEBUG
    if(it==_r.end() )
        qFatal("SemSolver::BoundaryConditions::robinData - ERROR : the border at index i"\
               "s not Robin.");
#endif
    return it->second;
};


template<class X>
void SemSolver::BoundaryConditions<2, X>::setBorder(int const &index,
               Type const &type,
               FunctionPtr const f1,
               FunctionPtr const f2)
{
    _types[index] = type;
    switch(type)
    {
    case DIRICHLET:
#ifdef SEMDEBUG
        if(!f1)
            qFatal("SemSolver::BoundaryConditions::setBorder - ERROR : trying to access "\
                   "null pointer f1.");
#endif
        _g[index] = f1;
```

```
        break;
    case NEUMANN:
#ifdef SEMDEBUG
        if(!f1)
            qFatal("SemSolver::BoundaryConditions::setBorder - ERROR : trying to access "\
                   "null pointer f1.");
#endif
        _h[index] = f1;
        break;
    case ROBIN:
#ifdef SEMDEBUG
        if(!f1)
            qFatal("SemSolver::BoundaryConditions::setBorder - ERROR : trying to access "\
                   "null pointer f1.");
        if(!f2)
            qFatal("SemSolver::BoundaryConditions::setBorder - ERROR : trying to access "\
                   "null pointer f2.");
#endif
        _gamma[index] = f1;
        _r[index] = f2;
        break;
    default:
        _types[index] = UNDEFINED;
    }
};


template<class X>
QStringList SemSolver::BoundaryConditions<2, X>::labels() const
{
    QStringList list;
    for(typename TypesMap::const_iterator it=_types.begin(); it!=_types.end(); ++it)
        list << "S" + QString::number(it->first);
    return list;
};


template<class X>
QStringList SemSolver::BoundaryConditions<2, X>::mmls() const
{
    QStringList list;
    for(typename TypesMap::const_iterator it=_types.begin(); it!=_types.end(); ++it)
    {
        QString mml;
        switch(it->second)
        {
        case SemSolver::BoundaryConditions<2, X>::DIRICHLET:
            mml = "<mpadded depth='-2'><msub><mi>u</mi><mrow><mo>&#124;</mo><msub><mi>&G"\
                  "amma;</mi><mi>i</mi></msub></mrow></msub><mo>=</mo><msub><mi>g</mi><m"\
                  "i>i</mi></msub><mtext>,  </mtext><msub><mi>g</mi><mi>i</mi>"\
                  "</msub><mfenced open='(' close=')' separators=','> <mi> x </mi> <mi> "\
                  "y </mi> </mfenced> <mo>=</mo>"
                  + dirichletData(it->first)->mml() +
```

```
                        "</mpadded>";
                break;
            case SemSolver::BoundaryConditions<2, X>::NEUMANN:
                mml = "<mpadded depth='-2'><mo>&nabla;</mo><msub><mi>u</mi><mrow><mo>&#124;<"\
                      "/mo><msub><mi>&Gamma;</mi><mi>i</mi></msub></mrow></msub><mo>&CenterD"\
                      "ot;</mo><mi mathvariant='bold'>n</mi><mo>=</mo><msub><mi>h</mi><mi>i<"\
                      "/mi></msub><mtext>,  </mtext><msub><mi>h</mi><mi>i</mi></ms"\
                      "ub><mfenced open='(' close=')' separators=','> <mi> x </mi> <mi> y </"\
                      "mi> </mfenced><mo>=</mo>"
                      + neumannData(it->first)->mml() +
                      "</mpadded>";
                break;
            case SemSolver::BoundaryConditions<2, X>::ROBIN:
                mml = "<mpadded depth='-2'><mo>&nabla;</mo><msub><mi>u</mi><mrow><mo>&#124;<"\
                      "/mo><msub><mi>&Gamma;</mi><mi>i</mi></msub></mrow></msub><mo>&CenterD"\
                      "ot;</mo><mi mathvariant='bold'>n</mi><mo>+</mo><msub><mi>&gamma;</mi>"\
                      "<mi>i</mi></msub><msub><mi>u</mi><mrow><mo>&#124;</mo><msub><mi>&Gamm"\
                      "a;</mi><mi>i</mi></msub></mrow></msub><mo>=</mo><msub><mi>r</mi><mi>i"\
                      "</mi></msub><mtext>,  </mtext><msub><mi>&gamma;</mi><mi>i</"\
                      "mi></msub><mfenced open='(' close=')' separators=','> <mi> x </mi> <m"\
                      "i> y </mi> </mfenced><mo>=</mo>"
                      + robinCoefficient(it->first)->mml() +
                      "<mtext>,  </mtext><msub><mi>r</mi><mi>i</mi></msub><mfenced"\
                      " open='(' close=')' separators=','> <mi> x </mi> <mi> y </mi> </mfenc"\
                      "ed><mo>=</mo>"
                      + robinData(it->first)->mml() +
                      "</mpadded>";
                break;
            default:
                mml = "<mtext>none</mtext>";
            }
            list << mml;
        }
        return list;
};

template<class X>
void SemSolver::BoundaryConditions<2, X>::clear()
{
    _types.clear();
    _g.clear();
    _h.clear();
    _gamma.clear();
    _r.clear();
};

#endif // BOUNDARYCONDITIONS_HPP
```

7.7 boundaryconditions.hpp File Reference

```
#include <QFile>
#include <SemSolver/boundaryconditions.hpp>
#include <SemSolver/scriptfunction.hpp>
```

Namespaces

- namespace SemSolver

  Project main namespace.

- namespace SemSolver::IO

  Namespace for Input/Output operations on SemSolver Classes.

Functions

- template<class X >
  bool SemSolver::IO::read_boundary_conditions (QFile *file, BoundaryConditions< 2, X > &boundary_conditions)

  Read 2D BoundaryConditions from file.

7.8 boundaryconditions.hpp

```
#ifndef IO_BOUNDARYCONDITIONS_HPP
#define IO_BOUNDARYCONDITIONS_HPP

#include <QFile>

#include <SemSolver/boundaryconditions.hpp>

namespace SemSolver
{
    namespace IO
    {
        template<class X>
        bool read_boundary_conditions(QFile *file,
                                      BoundaryConditions<2, X> &boundary_conditions
);
    };
};

#include <SemSolver/scriptfunction.hpp>

template<class X>
bool SemSolver::IO::read_boundary_conditions(QFile *file,
                                             BoundaryConditions<2, X> &bc)
{
    bc.clear();
```

```
#ifdef SEMDEBUG
    if(!file->open(QIODevice::ReadOnly))
    {
        qWarning("SemSolver::IO::readBoundaryCondition - ERROR : cannot open file.");
        return false;
    }
#endif
    QTextStream input(file);
    QStringList values;
    values = next_non_empty_line_values(input);
    while(!values.isEmpty())
    {
#ifdef SEMDEBUG
        if(values.size()<2)
        {
            qWarning("SemSolver::IO::readBoundaryCondition - ERROR : wrong number of inp"\
                     "uts on a line.");
            file->close();
            return false;
        }
#endif
        if(values[1]=="DIRICHLET")
        {
#ifdef SEMDEBUG
            if(values.size()!=3)
            {
                qWarning("SemSolver::IO::readBoundaryCondition - ERROR : wrong number of"\
                         "inputs on Dirichlet line.");
                file->close();
                return false;
            }
#endif
            bc.setBorder(values[0].toInt(),
                         BoundaryConditions<2,X>::DIRICHLET,
                         new ScriptFunction< Point<2, X>, X >(values[2]));
        }
        else if(values[1]=="NEUMANN")
        {
#ifdef SEMDEBUG
            if(values.size()!=3)
            {
                qWarning("SemSolver::IO::readBoundaryCondition - ERROR : wrong number of"\
                         "inputs on Neumann line.");
                file->close();
                return false;
            }
#endif
            bc.setBorder(values[0].toInt(),
                         BoundaryConditions<2,X>::NEUMANN,
                         new ScriptFunction< Point<2, X>, X >(values[2]));
        }
        else if(values[1]=="ROBIN")
        {
#ifdef SEMDEBUG
            if(values.size()!=4)
            {
                qWarning("SemSolver::IO::readBoundaryCondition - ERROR : wrong number of"\
```

```
                          "inputs on Robin line.");
                 file->close();
                 return false;
             }
#endif
             bc.setBorder(values[0].toInt(),
                          BoundaryConditions<2,X>::ROBIN,
                          new ScriptFunction< Point<2, X>, X >(values[2]),
                          new ScriptFunction< Point<2, X>, X >(values[3]));
         }
#ifdef SEMDEBUG
         else
         {
             qWarning("SemSolver::IO::readBoundaryCondition - ERROR : unknown input line "\
                      "in file.");
             file->close();
             return false;
         };
#endif
         values = next_non_empty_line_values(input);
     }
     file->close();

     return true;
};

#endif // IO_BOUNDARYCONDITIONS_HPP
```

7.9 buildsolution.hpp File Reference

`#include <SemSolver/function.hpp>`

`#include <SemSolver/semspace.hpp>`

`#include <SemSolver/vector.hpp>`

Namespaces

- namespace SemSolver

    Project main namespace.

- namespace SemSolver::PostProcessor

Functions

- template<class X >
  void SemSolver::PostProcessor::build_solution (const SemSpace< 2, X > &space, const Vector< X > &coefficients, Function< Point< 2, X >, X > ∗&solution)

7.10 buildsolution.hpp

```
#ifndef BUILDSOLUTION_HPP
#define BUILDSOLUTION_HPP

#include <SemSolver/function.hpp>
#include <SemSolver/semspace.hpp>
#include <SemSolver/vector.hpp>

namespace SemSolver
{
    namespace PostProcessor
    {
        template<class X>
        void build_solution(const SemSpace<2, X> &space,
                            const Vector<X> &coefficients,
                            Function< Point<2, X>, X > *&solution)
        {
            typedef typename SemSpace<2, X>::Element Element;
            delete solution;
            solution = new Element(&space, coefficients);
        };
    };
};

#endif // BUILDSOLUTION_HPP
```

7.11 choleskysolve.hpp File Reference

```
#include <jama_cholesky.h>
#include <SemSolver/matrix.hpp>
#include <SemSolver/vector.hpp>
```

Namespaces

- namespace SemSolver

  Project main namespace.

- namespace SemSolver::Solver

  Solver namespace.

Functions

- template<class X >
  bool SemSolver::Solver::cholesky_solve (Matrix< X > const &A, Vector< X > const &b, Vector< X > &x)

7.12 choleskysolve.hpp

```
#ifndef CHOLESKYSOLVE_HPP
#define CHOLESKYSOLVE_HPP

#if defined _WIN32 || defined _WIN64
#       include <SemSolver/math_defines>
#endif

#include <jama_cholesky.h>

#include <SemSolver/matrix.hpp>
#include <SemSolver/vector.hpp>

namespace SemSolver
{

    namespace Solver
    {
        template<class X>
        bool cholesky_solve(Matrix<X> const &A,
                            Vector<X> const &b,
                            Vector<X> &x)
        {
            JAMA::Cholesky<X> cholesky(A);
#ifdef SEMDEBUG
            if(!cholesky.is_spd())
            {
                qWarning("SemSolver::Solver::cholesky_solve - ERROR : Matrix A is not a "\
                         "symmetric, positive definite matrix.");
                return false;
            }
#endif
            x = cholesky.solve(b);
            return true;
        };
    };
};

#endif // CHOLESKYSOLVE_HPP
```

7.13 computealgebraicsystem.hpp File Reference

```
#include <SemSolver/semspace.hpp>
#include <SemSolver/problem.hpp>
#include <SemSolver/diffusionconvectionreactionequation.hpp>
#include <SemSolver/matrix.hpp>
#include <SemSolver/vector.hpp>
#include <SemSolver/Assembler/computediffusionmatrix.hpp>
#include <SemSolver/Assembler/computeconvectionmatrix.hpp>
#include <SemSolver/Assembler/computereactionmatrix.hpp>
```

```
#include <SemSolver/Assembler/computebordermatrix.hpp>
#include <SemSolver/Assembler/computeforcingvector.hpp>
#include <SemSolver/Assembler/computebordervector.hpp>
```

Namespaces

- namespace SemSolver

  Project main namespace.

- namespace SemSolver::Assembler

  Assembler namespace.

Functions

- template<class X >
  void SemSolver::Assembler::compute_algebraic_system (const SemSpace< 2, X > &space, const Problem< 2, X > &problem, Matrix< X > &A, Vector< X > &f)

  The computed system is stored in the Matrix and vectored refereced by A and f.

7.14 computealgebraicsystem.hpp

```
#ifndef COMPUTEALGEBRAICSYSTEM_HPP
#define COMPUTEALGEBRAICSYSTEM_HPP

#include <SemSolver/semspace.hpp>
#include <SemSolver/problem.hpp>
#include <SemSolver/diffusionconvectionreactionequation.hpp>
#include <SemSolver/matrix.hpp>
#include <SemSolver/vector.hpp>

#include <SemSolver/Assembler/computediffusionmatrix.hpp>
#include <SemSolver/Assembler/computeconvectionmatrix.hpp>
#include <SemSolver/Assembler/computereactionmatrix.hpp>
#include <SemSolver/Assembler/computebordermatrix.hpp>
#include <SemSolver/Assembler/computeforcingvector.hpp>
#include <SemSolver/Assembler/computebordervector.hpp>

namespace SemSolver
{

    namespace Assembler
    {

        template<class X>
        void compute_algebraic_system(const SemSpace<2, X> &space,
                                      const Problem<2, X> &problem,
                                      Matrix<X> &A,
                                      Vector<X> &f)
```

```
        {
            if( problem.equation()->type()==Equation<2, X>::DIFFUSION_CONVECTION_REACTION )
            {
                const DiffusionConvectionReactionEquation<2, X> *equation =
                        (const DiffusionConvectionReactionEquation<2, X> *)problem.equation();
                int n = space.nodes();
                A = Matrix<X>(n,n,0.);
                f = Vector<X>(n,0.);
                Matrix<X> Ad, Ac, Ar, Ab;
                Vector<X> ff, fb;
                if(equation->diffusion())
                {
#ifdef SEMDEBUG
                    qDebug() << "diffusion matrix";
#endif
                    Assembler::compute_diffusion_matrix(space, equation->diffusion(), Ad);
                    A += Ad;
                }
                if(equation->convection())
                {
#ifdef SEMDEBUG
                    qDebug() << "convection matrix";
#endif
                    Assembler::compute_convection_matrix(space,equation->convection(),Ac);
                    A += Ac;
                }
                if(equation->reaction())
                {
#ifdef SEMDEBUG
                    qDebug() << "reaction matrix";
#endif
                    Assembler::compute_reaction_matrix(space, equation->reaction(), Ar);
                    A += Ar;
                }
#ifdef SEMDEBUG
                qDebug() << "border matrix";
#endif
                Assembler::compute_border_matrix(space,
                                                 problem.boundaryConditions(),
                                                 equation->diffusion(),
                                                 problem.parameters()->penality(),
                                                 Ab);
                A += Ab;
                if(equation->forcing())
                {
#ifdef SEMDEBUG
                    qDebug() << "forcing vector";
#endif
                    Assembler::compute_forcing_vector(space, equation->forcing(), ff);
                    f += ff;
                }
#ifdef SEMDEBUG
                qDebug() << "border vector";
#endif
                Assembler::compute_border_vector(space,
```

```
                                                            problem.boundaryConditions(),
                                                            equation->diffusion(),
                                                            problem.parameters()->penality()
,
                                                            fb);
                    f += fb;
                }
            };
        };
};

#endif // COMPUTEALGEBRAICSYSTEM_HPP
```

7.15 computebordermatrix.hpp File Reference

`#include <SemSolver/semspace.hpp>`

`#include <SemSolver/boundaryconditions.hpp>`

`#include <SemSolver/matrix.hpp>`

`#include <SemSolver/multiindex.hpp>`

Namespaces

- namespace SemSolver

  Project main namespace.

- namespace SemSolver::Assembler

  Assembler namespace.

Functions

- template<class X >
  void SemSolver::Assembler::compute_border_matrix (const SemSpace< 2, X > &space, const BoundaryConditions< 2, X > *boundary_conditions, const Function< Point< 2, X >, X > *diffusion, double const &penality, Matrix< X > &matrix)

  The computed matrix is stored in the Matrix referenced by matrix.

7.16 computebordermatrix.hpp

```
#ifndef COMPUTEBORDERMATRIX_HPP
#define COMPUTEBORDERMATRIX_HPP

#include <SemSolver/semspace.hpp>
#include <SemSolver/boundaryconditions.hpp>
```

```
#include <SemSolver/matrix.hpp>
#include <SemSolver/multiindex.hpp>

namespace SemSolver
{

    namespace Assembler
    {

        template<class X>
        void compute_border_matrix(const SemSpace<2, X> &space,
                                   const BoundaryConditions<2, X> *boundary_conditions,
                                   const Function< Point<2, X>, X > *diffusion,
                                   double const &penality,
                                   Matrix<X> &matrix )
        {
            unsigned n = space.nodes();
            int N = space.degree();
            unsigned Mb = space.borders();

            matrix = Matrix<X>(n,n,0.);
            for(unsigned i=0; i<Mb; ++i)
            {
                for(int j=0; j<=N; ++j)
                {
                    MultiIndex<2> mi;
                    mi.setSubIndex(0,i+1);
                    mi.setSubIndex(1,j);
                    int I = space.borderIndex(mi);
                    int border = space.borderId(space.border(i));
                    if(boundary_conditions->borderType(border) ==
                       BoundaryConditions<2,X>::DIRICHLET)
                    {
                        X const &alpha = space.borderWeight(mi);
                        X const &eta = penality;
                        matrix[I][I] += alpha * eta;
                    }
                    else if(boundary_conditions->borderType(border)==
                            BoundaryConditions<2,X>::ROBIN)
                    {
                        X const &alpha = space.borderWeight(mi);
                        Point<2,X> const &x = space.borderNode(mi).point();
                        X mi = diffusion ? diffusion->evaluate(x) : 0;
                        X gamma = boundary_conditions->robinCoefficient(
                                border)->evaluate(x);
                        matrix[I][I] += alpha * mi * gamma;
                    }
                }
            }
        };
    };
};

#endif // COMPUTECONVECTIONMATRIX_HPP
```

7.17 computebordervector.hpp File Reference

`#include <SemSolver/semspace.hpp>`

```
#include <SemSolver/boundaryconditions.hpp>
#include <SemSolver/vector.hpp>
```

Namespaces

- namespace SemSolver

  Project main namespace.

- namespace SemSolver::Assembler

  Assembler namespace.

Functions

- template<class X >
  void SemSolver::Assembler::compute_border_vector (const SemSpace< 2, X > &space, const BoundaryConditions< 2, X > ∗boundary_conditions, const Function< Point< 2, X >, X > ∗diffusion, const double &penality, Vector< X > &vector)

  The computed matrix is stored in the Matrix referenced by matrix.

7.18 computebordervector.hpp

```
#ifndef COMPUTEBORDERVECTOR_HPP
#define COMPUTEBORDERVECTOR_HPP

#include <SemSolver/semspace.hpp>
#include <SemSolver/boundaryconditions.hpp>
#include <SemSolver/vector.hpp>

namespace SemSolver
{

    namespace Assembler
    {

        template<class X>
        void compute_border_vector(const SemSpace<2, X> &space,
                                   const BoundaryConditions<2, X> *boundary_conditions,
                                   const Function< Point<2, X>, X > *diffusion,
                                   const double &penality,
                                   Vector<X> &vector)
        {
            int n = space.nodes();
            int N = space.degree();
            int Mb = space.borders();
            vector = Vector<X>(n,0.);
            for(int i=0; i<Mb; ++i)
            {
```

```
                for(int j=0; j<=N; ++j)
                {
                    MultiIndex<2> mi;
                    mi.setSubIndex(0,i+1);
                    mi.setSubIndex(1,j);
                    int I = space.borderIndex(mi);
                    X alpha = space.borderWeight(mi);
                    int border = space.borderId(space.border(i));
                    Point<2,X> const &x = space.borderNode(mi).point();
                    if(boundary_conditions->borderType(border) == BoundaryConditions<2,X>::DIRICHLET)
                    {
                        X const &eta = penality;
                        X g = boundary_conditions->dirichletData(border)->evaluate(x);
                        vector[I] += alpha * eta * g;
                    }
                    else if(boundary_conditions->borderType(border)== BoundaryConditions<2,X>::NEUMANN)
                    {
                        X mi = diffusion ? diffusion->evaluate(x) : 0;
                        X h = boundary_conditions->neumannData(border)->evaluate(x);
                        vector[I]+= alpha * mi * h;
                    }
                    else if(boundary_conditions->borderType(border)== BoundaryConditions<2,X>::ROBIN)
                    {
                        X mi = diffusion ? diffusion->evaluate(x) : 0;
                        X r = boundary_conditions->robinData(border)->evaluate(x);
                        vector[I]+= alpha * mi * r;
                    }
                }
            }
        };
    };
};

#endif // COMPUTEBORDERVECTOR_HPP
```

7.19 computeconvectionmatrix.hpp File Reference

```
#include <SemSolver/function.hpp>
#include <SemSolver/point.hpp>
#include <SemSolver/vector.hpp>
#include <SemSolver/matrix.hpp>
```

Namespaces

- namespace SemSolver

  Project main namespace.

- namespace SemSolver::Assembler

Assembler namespace.

Functions

- template<class X >
  void SemSolver::Assembler::compute_convection_matrix (const SemSpace< 2, X > &space, const Function< Point< 2, X >, Vector< X > > *convection, Matrix< X > &matrix)

  The computed matrix is stored in the Matrix referenced by matrix.

7.20 computeconvectionmatrix.hpp

```
#ifndef COMPUTECONVECTIONMATRIX_HPP
#define COMPUTECONVECTIONMATRIX_HPP

#include <SemSolver/function.hpp>
#include <SemSolver/point.hpp>
#include <SemSolver/vector.hpp>
#include <SemSolver/matrix.hpp>

namespace SemSolver
{

    namespace Assembler
    {

        template<class X>
        void compute_convection_matrix(const SemSpace<2, X> &space,
                                       const Function< Point<2, X>,
                                       Vector<X> > *convection,
                                       Matrix<X> &matrix)
        {
            typedef typename SemSpace<2,X>::Node Node;

            int n = space.nodes();

            matrix = Matrix<X>(n,n,0.);

            for (int I0=0; I0<n; ++I0)
            {
                Node const &node0 = space.node(I0);
                for (int I1=0; I1<n; ++I1)
                {
                    Node const &node1 = space.node(I1);
                    int l0=0, l1=0;
                    while(l0<node0.supportSubDomains() && l1<node1.supportSubDomains())
                    {
                        if(node0.subDomainIndex(l0).subIndex(0) <
                           node1.subDomainIndex(l1).subIndex(0))
                            ++l0;
                        else if(node0.subDomainIndex(l0).subIndex(0) >
                                node1.subDomainIndex(l1).subIndex(0))
                            ++l1;
```

```
                        else
                        {
                            MultiIndex<3> mi0 = node0.subDomainIndex(l0);
                            int i = mi0.subIndex(0);
                            X alpha = space.subDomainWeight(mi0);
                            Point<2,X> x0 = node0.point();
                            Vector<X> beta = convection->evaluate(x0);
                            Vector<X> const &grad1 =
                                    space.baseFunction(I1)->\
                                    evaluateRestrictionGradient(i,x0);
                            matrix[I0][I1] += alpha * scalar(beta,grad1);
                            ++l0;
                            ++l1;
                        }
                    }
                }
            }
        };
    };
};

#endif // COMPUTECONVECTIONMATRIX_HPP
```

7.21 computediffusionmatrix.hpp File Reference

```
#include <SemSolver/function.hpp>
#include <SemSolver/point.hpp>
#include <SemSolver/matrix.hpp>
#include <SemSolver/semspace.hpp>
```

Namespaces

- namespace SemSolver

  Project main namespace.

- namespace SemSolver::Assembler

  Assembler namespace.

Functions

- template< class X >
  void SemSolver::Assembler::compute_diffusion_matrix (const SemSpace< 2, X > &space, const Function< Point< 2, X >, X > *diffusion, Matrix< X > &matrix)

  The computed matrix is stored in the Matrix referenced by matrix.

7.22 computediffusionmatrix.hpp

```
#ifndef COMPUTEDIFFUSIONMATRIX_HPP
#define COMPUTEDIFFUSIONMATRIX_HPP

#include <SemSolver/function.hpp>
#include <SemSolver/point.hpp>
#include <SemSolver/matrix.hpp>
#include <SemSolver/semspace.hpp>

namespace SemSolver
{

    namespace Assembler
    {

        template<class X>
        void compute_diffusion_matrix(const SemSpace<2, X> &space,
                                      const Function< Point<2, X>, X> *diffusion,
                                      Matrix<X> &matrix)
        {
            typedef typename SemSpace<2, X>::Node Node;

            int n = space.nodes();
            int N = space.degree();

            matrix = Matrix<X>(n,n,0.);

            for (int I0=0; I0<n; ++I0)
            {
                Node const &node0 = space.node(I0);
                for (int I1=0; I1<n; ++I1)
                {
                    Node const &node1 = space.node(I1);
                    int l0=0, l1=0;
                    while(l0<node0.supportSubDomains() && l1<node1.supportSubDomains())
                    {
                        if(node0.subDomainIndex(l0).subIndex(0) < node1.subDomainIndex(l1).subIndex(0))
                            ++l0;
                        else if(node0.subDomainIndex(l0).subIndex(0) > node1.subDomainIndex(l1).subIndex(0))
                            ++l1;
                        else
                        {
                            int i = node0.subDomainIndex(l0).subIndex(0);
                            for (int j2=0; j2<=N; ++j2)
                            {
                                for (int k2=0; k2<=N; ++k2)
                                {
                                    MultiIndex<3> mi2;
                                    mi2.setSubIndex(0,i);
                                    mi2.setSubIndex(1,j2);
                                    mi2.setSubIndex(2,k2);
                                    X alpha = space.subDomainWeight(mi2);
                                    Point<2,X> x2 = space.subDomainNode(mi2).point();
                                    X mi = diffusion->evaluate(x2);
                                    Vector<X> const &grad0 = space.baseFunction(I0)->evaluateRestrictionGradient(i,x2);
```

```
                                    Vector<X> const &grad1 = space.baseFunction(I1)->evaluateRestrictionGradient(i,x2);
                                    matrix[I0][I1] += alpha * mi * scalar(grad1,grad0);
                                }
                            }
                            ++I0;
                            ++I1;
                        }
                    }
                }
            }
        };
    };
};

#endif // COMPUTEDIFFUSIONMATRIX_HPP
```

7.23 computeforcingvector.hpp File Reference

```
#include <SemSolver/problem.hpp>
#include <SemSolver/vector.hpp>
```

Namespaces

- namespace SemSolver

  Project main namespace.

- namespace SemSolver::Assembler

  Assembler namespace.

Functions

- template< class X >
  void SemSolver::Assembler::compute_forcing_vector (const SemSpace< 2, X > &space, const Function< Point< 2, X >, X > *forcing, Vector< X > &vector)

  The computed matrix is stored in the Matrix referenced by matrix.

7.24 computeforcingvector.hpp

```
#ifndef COMPUTEFORCINGVECTOR_HPP
#define COMPUTEFORCINGVECTOR_HPP

#include <SemSolver/problem.hpp>
#include <SemSolver/vector.hpp>
```

```
namespace SemSolver
{

    namespace Assembler
    {

        template<class X>
        void compute_forcing_vector(const SemSpace<2, X> &space,
                                    const Function< Point<2, X>, X > *forcing,
                                    Vector<X> &vector)
        {
            typedef typename SemSpace<2,X>::Node Node;

            int n = space.nodes();
            vector = Vector<X>(n,0.);
            for(int I=0; I<n; ++I)
            {
                Node const &node = space.node(I);
                for(int l=0; l<node.supportSubDomains(); ++l)
                {
                    MultiIndex<3> Il = node.subDomainIndex(l);
                    X const &alpha = space.subDomainWeight(Il);
                    Point<2,X> const &x = space.subDomainNode(Il).point();
                    X f = forcing->evaluate(x);
                    vector[I] += alpha * f;
                }
            }
        };
    };
};

#endif // COMPUTEFORCINGVECTOR_HPP
```

7.25 computeplotdata.hpp File Reference

```
#include <SemSolver/semspace.hpp>
#include <SemSolver/vector.hpp>
#include <qwt3d_types.h>
```

Namespaces

- namespace SemSolver

    Project main namespace.

- namespace SemSolver::PostProcessor

Functions

- template< class X >
  void SemSolver::PostProcessor::compute_plot_data (const SemSpace< 2, X >

&space, const Vector< X > &u, Qwt3D::TripleField &data, Qwt3D::CellField &poly)

7.26 computeplotdata.hpp

```
#ifndef COMPUTEPLOTDATA_HPP
#define COMPUTEPLOTDATA_HPP

#include <SemSolver/semspace.hpp>
#include <SemSolver/vector.hpp>

#include <qwt3d_types.h>

namespace SemSolver
{
    namespace PostProcessor
    {
        template<class X>
        void compute_plot_data(const SemSpace<2, X> &space,
                               const Vector<X> &u,
                               Qwt3D::TripleField &data,
                               Qwt3D::CellField &poly)

        {
            for(int i=0; i<u.rows(); ++i)
                data.push_back(Qwt3D::Triple(space.node(i).point().x(), space.node(i).point().y(), u[i]));
            for(int i=0; i<space.subDomains(); ++i)
            {
                for(int j=0; j<space.degree(); ++j)
                {
                    for(int k=0; k<space.degree(); ++k)
                    {
                        Qwt3D::Cell cell;
                        MultiIndex<3> mi;
                        mi.setSubIndex(0,i);
                        mi.setSubIndex(1,j);
                        mi.setSubIndex(2,k);
                        cell.push_back(space.subDomainIndex(mi));
                        mi.setSubIndex(0,i);
                        mi.setSubIndex(1,j+1);
                        mi.setSubIndex(2,k);
                        cell.push_back(space.subDomainIndex(mi));
                        mi.setSubIndex(0,i);
                        mi.setSubIndex(1,j+1);
                        mi.setSubIndex(2,k+1);
                        cell.push_back(space.subDomainIndex(mi));
                        mi.setSubIndex(0,i);
                        mi.setSubIndex(1,j);
                        mi.setSubIndex(2,k+1);
                        cell.push_back(space.subDomainIndex(mi));
                        poly.push_back(cell);
                    }
                }
            }
        }
    }
};
```

```
00054 #endif // COMPUTEPLOTDATA_HPP
```

7.27 computepolygonationfrompslg.hpp File Reference

```
#include <boost/shared_ptr.hpp>
#include <CGAL/Cartesian.h>
#include <CGAL/Filtered_kernel.h>
#include <CGAL/Straight_skeleton_2.h>
#include <CGAL/create_straight_skeleton_from_polygon_with_-
holes_2.h>
#include <CGAL/squared_distance_2.h>
#include <SemSolver/pointsset.hpp>
#include <SemSolver/semparameters.hpp>
#include <SemSolver/segment.hpp>
#include <SemSolver/polygonation.hpp>
#include <SemSolver/polygonwithholes.hpp>
#include <SemSolver/pslg.hpp>
#include <SemSolver/PreProcessor/computepolygonwithholesfrompslg.hpp>
```

Namespaces

- namespace SemSolver

  Project main namespace.

- namespace SemSolver::PreProcessor

  PreProcessor namespace.

Functions

- template< class X >
  bool SemSolver::PreProcessor::compute_polygonation_from_pslg (const PSLG< X > &pslg, Polygonation< 2, X > &polygonation)

7.28 computepolygonationfrompslg.hpp

```
00001 #ifndef COMPUTEPOLYGONATIONFROMPSLG_HPP
00002 #define COMPUTEPOLYGONATIONFROMPSLG_HPP
00003
00004 #ifdef __GNUC__
```

```
#   ifdef __GLIBC__
#    if ((__GLIBC__ > 2) || ((__GLIBC__ == 2) && (__GLIBC_MINOR__ >= 1)))
#      include <stdint.h>
#    endif
#   endif
#endif

#if defined _WIN32 || defined _WIN64
#   undef max
#   include <limits>
#endif

#include <boost/shared_ptr.hpp>

#include <CGAL/Cartesian.h>
#include <CGAL/Filtered_kernel.h>
#include <CGAL/Straight_skeleton_2.h>
#include <CGAL/create_straight_skeleton_from_polygon_with_holes_2.h>
#include <CGAL/squared_distance_2.h>
#include <SemSolver/pointsset.hpp>
#include <SemSolver/semparameters.hpp>
#include <SemSolver/segment.hpp>
#include <SemSolver/polygonation.hpp>
#include <SemSolver/polygonwithholes.hpp>
#include <SemSolver/pslg.hpp>
#include <SemSolver/PreProcessor/computepolygonwithholesfrompslg.hpp>

namespace SemSolver
{

    namespace PreProcessor
    {
        template<class X>
        bool compute_polygonation_from_pslg(const PSLG<X> &pslg,
                                            Polygonation<2,X> &polygonation)
        {
            typedef CGAL::Straight_skeleton_2< CGAL::Filtered_kernel<
                    CGAL::Simple_cartesian<X> > > Skeleton;
            typedef typename boost::shared_ptr< Skeleton > SkeletonPtr;

            typedef typename Skeleton::Vertex_iterator VertexIterator;
            typedef typename Skeleton::Face_iterator FaceIterator;
            typedef typename Skeleton::Halfedge_handle HalfedgeHandle;
            typedef typename Skeleton::Halfedge_iterator HalfedgeIterator;
            typedef typename Segment&lt;2,X&gt;::less Segment_less;
            typedef std::map&lt;Segment&lt;2,X&gt;, int, Segment_less> SegmentIdsMap;
            typedef typename SegmentIdsMap::value_type Pair;
            PolygonWithHoles<2,X> domain;
            if(!compute_polygon_with_holes_from_pslg(pslg, domain))
            {
                return false;
            }
            SkeletonPtr skeleton = CGAL::create_interior_straight_skeleton_2(doma
in.cgal());

            polygonation.clear();

            SegmentIdsMap face_id;
            int index = 0;
            for (FaceIterator it = skeleton->faces_begin();
            it!=skeleton->faces_end();++it)
                face_id.insert(Pair(
```

```
                        Segment<2, X>(it->halfedge()->vertex()->point(),
                                      it->halfedge()->opposite()->vertex()->point()),
                        ++index));
            SegmentIdsMap boundary_id;
            index=0;
            for (HalfedgeIterator it = skeleton->halfedges_begin();
            it!= skeleton->halfedges_end(); ++it)
                if(it->is_border())
                    boundary_id.insert(Pair(
                            Segment<2, X>(it->vertex()->point(),
                                          it->opposite()->vertex()->point()), ++index));
            for (FaceIterator it = skeleton->faces_begin();
            it != skeleton->faces_end(); ++it)
            {
                Polygon<2, X> polygon;
                std::vector<int> neighbours;
                HalfedgeHandle halfedge = it->halfedge();
                do
                {
                    polygon.push_back(halfedge->vertex()->point());
                    if(halfedge->is_bisector())
                        neighbours.push_back(face_id[
                                Segment<2, X>(halfedge->opposite()->face()->halfedge()->vertex()->point(),
                                              halfedge->opposite()->face()->halfedge()->opposite()->vertex()->point())]);
                    else
                        neighbours.push_back(-boundary_id[
                                Segment<2, X>(halfedge->opposite()->vertex()->point(),
                                              halfedge->opposite()->opposite()->vertex()->point())]);
                    halfedge = halfedge->next();
                }
                while(halfedge->vertex()->point()!=it->halfedge()->vertex()->point());
                polygonation.addElement(polygon, neighbours);
            }
            return true;
        }
    }
}

#endif // COMPUTEPOLYGONATIONFROMPSLG_HPP
```

7.29 computepolygonwithholesfrompslg.hpp File Reference

`#include <SemSolver/polygonwithholes.hpp>`

`#include <SemSolver/pslg.hpp>`

`#include <SemSolver/sequenceslist.hpp>`

Namespaces

- namespace SemSolver

Project main namespace.

- namespace SemSolver::PreProcessor

  PreProcessor namespace.

Functions

- template< class X >
  bool SemSolver::PreProcessor::compute_vertices_sequences_from_pslg (PSLG< X > const &pslg, Sequences_list &vertices_sequences)

  Get sequences of vertices representing the set of polygons given by a PSLG.

- template< class X >
  bool SemSolver::PreProcessor::compute_polygon_with_holes_from_pslg (PSLG< X > const &pslg, PolygonWithHoles< 2, X > &polygon)

  Compute the polygon with holes given by a PSLG.

7.30 computepolygonwithholesfrompslg.hpp

```
#ifndef COMPUTEPOLYGONWITHHOLESFROMPSLG_HPP
#define COMPUTEPOLYGONWITHHOLESFROMPSLG_HPP

#include <SemSolver/polygonwithholes.hpp>
#include <SemSolver/pslg.hpp>
#include <SemSolver/sequenceslist.hpp>

namespace SemSolver
{

    namespace PreProcessor
    {
        template<class X>
        bool compute_vertices_sequences_from_pslg(PSLG<X> const &pslg,
                                                                                    Sequences_list &vertices_sequences)
        {
            for(unsigned i=0; i<pslg.segments(); ++i)
            {
                bool segment_added = false;
                for(Sequences_list::iterator it = vertices_sequences.begin();
                it!=vertices_sequences.end() && !segment_added;
                ++it)
                {
                    if(pslg.segment(i).source==it->back())
                    {
                        it->push_back(pslg.segment(i).target);
                        segment_added = true;
                    }
                    else if(pslg.segment(i).target==it->back())
                    {
                        it->push_back(pslg.segment(i).source);
                        segment_added = true;
```

```
                    }
                }
                if(!segment_added)
                {
                    Sequence sequence;
                    sequence.push_back(pslg.segment(i).source);
                    sequence.push_back(pslg.segment(i).target);
                    vertices_sequences.push_back(sequence);
                }
            }
            Sequences_list::iterator begin = vertices_sequences.begin();
            Sequences_list::iterator end = vertices_sequences.end();
            Sequences_list::iterator before_end = --vertices_sequences.end();
            for(Sequences_list::iterator it_0 = begin; it_0!=before_end && it_0 !=end;
                ++it_0)
            {
                bool is_closed = (it_0->front()==it_0->back());
                if(!is_closed)
                {
                    Sequences_list::iterator after_it_0 = it_0; ++after_it_0;
                    for(Sequences_list::iterator it_1 = after_it_0; it_1!=end &&
                        !is_closed; ++it_1)
                    {
                        if(it_0->back()==it_1->front())
                        {
                            for(Sequence::const_iterator it_2=it_1->begin();
                            ++it_2!=it_1->end(); )
                                it_0->push_back(*it_2);
                            vertices_sequences.erase(it_1);
                            end = vertices_sequences.end();
                            is_closed = (it_0->front()==it_0->back());
                            it_1 = it_0;
                        }
                        else if(it_0->back()==it_1->back())
                        {
                            for(Sequence::const_reverse_iterator it_2=it_1->rbegin();
                            ++it_2!=it_1->rend(); )
                                it_0->push_back(*it_2);
                            vertices_sequences.erase(it_1);
                            end = vertices_sequences.end();
                            is_closed = (it_0->front()==it_0->back());
                            it_1 = it_0;
                        }
                        else if(it_0->front()==it_1->back())
                        {
                            for(Sequence::const_reverse_iterator it_2=it_1->rbegin();
                            ++it_2!=it_1->rend(); )
                                it_0->push_front(*it_2);
                            vertices_sequences.erase(it_1);
                            end = vertices_sequences.end();
                            is_closed = (it_0->front()==it_0->back());
                            it_1 = it_0;
                        }
                        else if(it_0->front()==it_1->front())
                        {
                            for(Sequence::const_iterator it_2=it_1->begin();
                            ++it_2!=it_1->end(); )
                                it_0->push_front(*it_2);
                            vertices_sequences.erase(it_1);
```

```
                            end = vertices_sequences.end();
                            is_closed = (it_0->front()==it_0->back());
                            it_1 = it_0;
                        }
                    }
                }
                before_end = --vertices_sequences.end();
            }
            return true;
        }

        template<class X>
        bool compute_polygon_with_holes_from_pslg(PSLG<X> const &pslg,
                                                  PolygonWithHoles<2,X> &polygon)

        {
            typedef std::list< Polygon<2,X> > Polygons_list;
            typedef typename Polygons_list::iterator Polygon_iterator;
            typedef typename Polygons_list::const_iterator Polygon_const_iterator;

            typedef std::map<int,int> Vertices_map;
            typedef typename Polygon<2,X>::Vertex_const_iterator Vertex_const_iterator;

            polygon.clear();
            Sequences_list vertices_sequences;
            compute_vertices_sequences_from_pslg(pslg,vertices_sequences);
            if(vertices_sequences.size()<1)
                return false;
            bool *is_used_vertex = new bool[pslg.vertices()];
            Vertices_map vertices_map;
            for(unsigned i=0; i<pslg.vertices(); ++i)
            {
                is_used_vertex[i] = false;
                vertices_map.insert(Vertices_map::value_type(pslg.vertex(i).number,i));
            }
            Polygons_list polygons;
            for(Sequences_list::const_iterator it_0=vertices_sequences.begin();
            it_0!=vertices_sequences.end(); ++it_0)
            {
#ifdef SEMDEBUG
                if(it_0->front()!=it_0->back())
                {
                    qWarning("PSLG::to_polygon : segments must define closed polygons");
                    return false;
                }
#endif
                Polygon<2,X> subpolygon;
                for(Sequence::const_iterator it_1=it_0->begin(); it_1!=--it_0->end(); ++it_1)
                {
#ifdef SEMDEBUG
                    if(is_used_vertex[vertices_map[*it_1]])
                    {
                        qWarning("PSLG::to_polygon : no more than two segments must pass through each vertex");
                        return false;
                    }
                    else
```

```
                    {
#endif
                        double const &x = pslg.vertex(vertices_map[*it_1]).x;
                        double const &y = pslg.vertex(vertices_map[*it_1]).y;
                        subpolygon.push_back(Point<2,X>(x,y));
                        is_used_vertex[vertices_map[*it_1]] = true;
#ifdef SEMDEBUG
                    }
#endif
                }
#ifdef SEMDEBUG
                if(subpolygon.is_empty())
                {
                    qWarning("PSLG::to_polygon : segments must define non empty p
olygons");
                    return false;
                }
                if(!subpolygon.is_simple())
                {
                    qWarning("PSLG::to_polygon : segments must define simple poly
gons");
                    return false;
                }
#endif
                polygons.push_back(subpolygon);
            }
#ifdef SEMDEBUG
            for(unsigned i=0; i<pslg.vertices(); ++i)
            {
                if(!is_used_vertex[i])
                {
                    qWarning("PSLG::to_polygon : there are unused vertex");
                    return false;
                }
            }
#endif
            Polygon_iterator outer;
            bool found = false;
            for(Polygon_iterator it_0 = polygons.begin(); it_0!=polygons.end() &&
 !found;
            ++it_0)
            {
                bool is_outer = true;
                for(Polygon_const_iterator it_1 = polygons.begin(); it_1!=polygon
s.end()
                    &&is_outer; ++it_1)
                    if(it_0!=it_1)
                        for (Vertex_const_iterator it_2 = it_1->vertices_begin();

                it_2 != it_1->vertices_end() &&is_outer; ++it_2)
                            if(!it_0->has_on_bounded_side(*it_2))
                                is_outer = false;
                if(is_outer)
                {
                    outer = it_0;
                    found = true;
                }
            }
            if(outer->is_clockwise_oriented())
                outer->reverse_orientation();
            polygon.outer_boundary() = *outer;
            polygons.erase(outer);
```

```
#ifdef SEMDEBUG
            if(polygons.size()<pslg.holes())
            {
                qWarning("PSLG::to_polygon : there are too many holes");
                return false;
            }
            if(polygons.size()>pslg.holes())
            {
                qWarning("PSLG::to_polygon : there are not enough holes");
                return false;
            }
#endif
            for(unsigned i=0; i<pslg.holes(); ++i)
            {
                Polygon_iterator it=polygons.begin();
                Point<2,X> hole(pslg.hole(i).x,pslg.hole(i).y);
#ifdef SEMDEBUG
                while(!it->has_on_bounded_side(hole))
                {
                    ++it;
                    if(it==polygons.end())
                    {
                        qWarning("PSLG::to_polygon : there is no hole inside inner polygon");
                        return false;
                    }
                }
#endif
                if(it->is_counterclockwise_oriented())
                    it->reverse_orientation();
                polygon.add_hole(*it);
                polygons.erase(it);
            }
            return true;
        }
    }
}

#endif // COMPUTEPOLYGONWITHHOLESFROMPSLG_HPP
```

7.31 computereactionmatrix.hpp File Reference

```
#include <SemSolver/semspace.hpp>
#include <SemSolver/matrix.hpp>
```

Namespaces

- namespace SemSolver

  Project main namespace.

- namespace SemSolver::Assembler

  Assembler namespace.

Functions

- template<class X >
  void SemSolver::Assembler::compute_reaction_matrix (const SemSpace< 2, X > &space, Function< Point< 2, X >, X > const ∗reaction, Matrix< double > &matrix)

  The computed matrix is stored in the Matrix referenced by matrix.

7.32 computereactionmatrix.hpp

```
#ifndef COMPUTEREACTIONMATRIX_HPP
#define COMPUTEREACTIONMATRIX_HPP

#include <SemSolver/semspace.hpp>
#include <SemSolver/matrix.hpp>

namespace SemSolver
{

    namespace Assembler
    {

        template<class X>
        void compute_reaction_matrix(const SemSpace<2, X> &space,
                                     Function< Point<2, X>, X> const *reaction,
                                     Matrix<double> &matrix)
        {
            typedef typename SemSpace<2,X>::Node Node;

            int n = space.nodes();
            matrix = Matrix<X>(n,n,0.);
            for(int I0=0; I0<n; ++I0)
            {
                Node const &node0 = space.node(I0);
                Point<2,X> x = node0.point();
                for(int l=0; l<node0.supportSubDomains(); ++l)
                {
                    MultiIndex<3> mi1 = node0.subDomainIndex(l);
                    int I1 = space.subDomainIndex(mi1);
                    X alpha = space.subDomainWeight(mi1);
                    X gamma = reaction->evaluate(x);
                    matrix[I0][I1] += alpha * gamma;
                }
            }
        };
    };
};

#endif // COMPUTEREACTIONMATRIX_HPP
```

7.33 computesolutionhull.hpp File Reference

`#include <SemSolver/vector.hpp>`

```
#include <SemSolver/semspace.hpp>
```

Namespaces

- namespace SemSolver

  Project main namespace.

- namespace SemSolver::PostProcessor

Functions

- template<class X >
  void SemSolver::PostProcessor::compute_solution_hull (const SemSpace< 2, X > &space, const Vector< X > &coefficients, double &xmin, double &ymin, double &zmin, double &xmax, double &ymax, double &zmax)

7.34 computesolutionhull.hpp

```
#ifndef COMPUTESOLUTIONHULL_HPP
#define COMPUTESOLUTIONHULL_HPP

#include <SemSolver/vector.hpp>
#include <SemSolver/semspace.hpp>

namespace SemSolver
{
    namespace PostProcessor
    {
        template<class X>
        void compute_solution_hull(const SemSpace<2, X> &space,
                                   const Vector<X> &coefficients,
                                   double &xmin,
                                   double &ymin,
                                   double &zmin,
                                   double &xmax,
                                   double &ymax,
                                   double &zmax)
        {
            if(!space.nodes())
                return;
            if(!coefficients.rows())
                return;
            xmin = xmax = space.node(0).point().x();
            ymin = ymax = space.node(0).point().y();
            zmin = zmax = coefficients[0];
            for(unsigned i=1; i<space.nodes(); ++i)
            {
                if(space.node(i).point().x() < xmin)
                    xmin = space.node(i).point().x();
                if(space.node(i).point().x() > xmax)
                    xmax = space.node(i).point().x();
                if(space.node(i).point().y() < ymin)
                    ymin = space.node(i).point().y();
                if(space.node(i).point().y() > ymax)
```

```
                        ymax = space.node(i).point().y();
                }
                for(int i=1; i<coefficients.rows(); ++i)
                {
                    /********* Not precise **********/
                    if(coefficients[i] < zmin)
                        zmin = coefficients[i];
                    if(coefficients[i] > zmax)
                        zmax = coefficients[i];
                }
            };
        };
};

#endif // COMPUTESOLUTIONHULL_HPP
```

7.35 diffusionconvectionreactionequation.hpp File Reference

```
#include <QDebug>
#include <SemSolver/equation.hpp>
#include <SemSolver/function.hpp>
#include <SemSolver/point.hpp>
#include <SemSolver/vector.hpp>
```

Classes

- class SemSolver::DiffusionConvectionReactionEquation< d, X >

  Class for handling Diffusion-Convection-Reaction steady equation.

- class SemSolver::DiffusionConvectionReactionEquation< 2, X >

  Class for handling Diffusion-Convection-Reaction steady equation on 2D spaces.

Namespaces

- namespace SemSolver

  Project main namespace.

7.36 diffusionconvectionreactionequation.hpp

```
#ifndef DIFFUSIONCONVECTIONREACTIONEQUATION_HPP
#define DIFFUSIONCONVECTIONREACTIONEQUATION_HPP

#include <QDebug>

namespace SemSolver
```

```
{
    template<int d, class X>
    class DiffusionConvectionReactionEquation;
};

#include <SemSolver/equation.hpp>
#include <SemSolver/function.hpp>
#include <SemSolver/point.hpp>
#include <SemSolver/vector.hpp>

namespace SemSolver
{

    template<int d, class X>
    class DiffusionConvectionReactionEquation
        : public Equation<d, X>
    {
        // coefficients
        Function< Point<d, X>, X >          *_diffusion;
        Function< Point<d, X>, Vector<X> > *_convection;
        Function< Point<d, X>, X >          *_reaction;
        Function< Point<d, X>, X >          *_forcing;

    public:

        typedef typename Equation<d, X>::Type Type;

        DiffusionConvectionReactionEquation()
        {
            _diffusion = 0;
            _convection = 0;
            _reaction = 0;
            _forcing = 0;
        };

        ~DiffusionConvectionReactionEquation()
        {
            delete _diffusion;
            delete _convection;
            delete _reaction;
            delete _forcing;
        };

        inline Type type() const
        {
            return Equation<d, X>::DIFFUSION_CONVECTION_REACTION;
        };

        QString mml() const;

        inline void setDiffusion(Function< Point<d, X>, X > *diffusion)
        {
            delete _diffusion;
            _diffusion = diffusion;
        };

        inline void setConvection(Function< Point<d, X>, Vector<X> > *convection)

        {
            delete _convection;
            _convection = convection;
        };
```

```
        inline void setReaction(Function< Point<d, X>, X > *reaction)
        {
            delete _reaction;
            _reaction = reaction;
        };

        inline void setForcing(Function< Point<d, X>, X > *forcing)
        {
            delete _forcing;
            _forcing = forcing;
        };

        inline Function< Point<d, X>, X > *diffusion() const
        {
            return _diffusion;
        };

        inline Function< Point<d, X>, Vector<X> > *convection() const
        {
            return _convection;
        };

        inline Function< Point<d, X>, X > *reaction() const
        {
            return _reaction;
        }

        inline Function< Point<d, X>, X > *forcing() const
        {
            return _forcing;
        };
    };

    template<class X>
    class DiffusionConvectionReactionEquation<2, X>
        : public Equation<2, X>
    {
        // coefficients
        Function< Point<2, X>, X >          *_diffusion;
        Function< Point<2, X>, Vector<X> > *_convection;
        Function< Point<2, X>, X >          *_reaction;
        Function< Point<2, X>, X >          *_forcing;

    public:

        typedef typename Equation<2, X>::Type Type;

        DiffusionConvectionReactionEquation()
        {
            _diffusion = 0;
            _convection = 0;
            _reaction = 0;
            _forcing = 0;
        };

        ~DiffusionConvectionReactionEquation()
        {
            delete _diffusion;
            delete _convection;
            delete _reaction;
            delete _forcing;
```

```
        };

        inline Type type() const
        {
            return Equation<2, X>::DIFFUSION_CONVECTION_REACTION;
        };

        QString mml() const
        {
            QString mml = "<mtable><mtr><mtd>";
            if(_diffusion && _convection && _reaction)
                mml += "<mo>-</mo><mo>&Del;&CenterDot;</mo><mfenced open='(' close=')'><mrow><mi"\
                       ">&mu;</mi><mo>&Del;</mo> <mi>u</mi></mrow></mfenced><mo>+</mo><mi mathva"\
                       "riant='bold'>b</mi><mo>&CenterDot;</mo><mo>&Del;</mo> <mi>u</mi><mo>+</m"\
                       "o><mi>&sigma;</mi><mi>u</mi>";
            else if(_diffusion && _convection)
                mml += "<mo>-</mo><mo>&Del;&CenterDot;</mo><mfenced open='(' close=')'><mrow><mi"\
                       ">&mu;</mi><mo>&Del;</mo> <mi>u</mi></mrow></mfenced><mo>+</mo><mi mathva"\
                       "riant='bold'>b</mi><mo>&CenterDot;</mo><mo>&Del;</mo> <mi>u</mi>";
            else if(_diffusion && _reaction)
                mml += "<mo>-</mo><mo>&Del;&CenterDot;</mo><mfenced open='(' close=')'><mrow><mi"\
                       ">&mu;</mi><mo>&Del;</mo> <mi>u</mi></mrow></mfenced><mo>+</mo><mi>&sigma"\
                       ";</mi><mi>u</mi>";
            else if(_convection && _reaction)
                mml += "<mi mathvariant='bold'>b</mi><mo>&CenterDot;</mo><mo>&Del;</mo> <mi>u</m"\
                       "i><mo>+</mo><mi>&sigma;</mi><mi>u</mi>";
            else if(_diffusion)
                mml += "<mo>-</mo><mo>&Del;&CenterDot;</mo><mfenced open='(' close=')'><mrow><mi"\
                       ">&mu;</mi><mo>&Del;</mo> <mi>u</mi></mrow></mfenced>";
            else if(_convection)
                mml += "<mi mathvariant='bold'>b</mi><mo>&CenterDot;</mo><mo>&Del;</mo> <mi>u</m"\
                       "i>";
            else if(_reaction)
                mml += "<mi>&sigma;</mi><mi>u</mi>";
            else
                mml += "<mn>0</mn>";
            if(_forcing)
                mml += "<mo>=</mo><mi>f</mi></mtd></mtr>";
            else
                mml += "<mo>=</mo><mn>0</mn></mtd></mtr>";
            if(_diffusion || _convection || _reaction || _forcing)
            {
                mml += "<mtr><mtd><mtable><mtr>";
                if(_diffusion)
                    mml += "<mtd><mi>&mu;</mi> <mfenced open='(' close=')' separators=','> <mi> "\
                           "x </mi> <mi> y </mi> </mfenced> <mo>=</mo>"
                           + _diffusion->mml()
                    + "</mtd>";
                if(_convection)
                    mml += "<mtd><mi mathvariant='bold'>b</mi> <mfenced open='('
```

```
close=')' separ"\
                              "ators=','> <mi> x </mi> <mi> y </mi> </mfenced> <mo>=
</mo>"
                              + _convection->mml()
                       + "</mtd>";
                   if(_reaction)
                       mml += "<mtd><mi>&sigma;</mi> <mfenced open='(' close=')' sep
arators=','> <m"\
                              "i> x </mi> <mi> y </mi> </mfenced> <mo>=</mo>"
                              + _reaction->mml()
                       + "</mtd>";
                   if(_forcing)
                       mml += "<mtd><mi>f</mi> <mfenced open='(' close=')' separator
s=','> <mi> x <"\
                              "/mi> <mi> y </mi> </mfenced> <mo>=</mo>"
                              + _forcing->mml()
                       + "</mtd>";
                   mml += "</mtr></mtable></mtd></mtr>";
               }
               mml += "</mtable>";
               return mml;
           };

           inline void setDiffusion(Function< Point<2, X>, X > *diffusion)
           {
               delete _diffusion;
               _diffusion = diffusion;
           };

           inline void setConvection(Function< Point<2, X>, Vector<X> > *convection)

           {
               delete _convection;
               _convection = convection;
           };

           inline void setReaction(Function< Point<2, X>, X > *reaction)
           {
               delete _reaction;
               _reaction = reaction;
           };

           inline void setForcing(Function< Point<2, X>, X > *forcing)
           {
               delete _forcing;
               _forcing = forcing;
           };

           inline Function< Point<2, X>, X > *diffusion() const
           {
               return _diffusion;
           };

           inline Function< Point<2, X>, Vector<X> > *convection() const
           {
               return _convection;
           };

           inline Function< Point<2, X>, X > *reaction() const
           {
               return _reaction;
           }
```

```
        inline Function< Point<2, X>, X > *forcing() const
        {
            return _forcing;
        };

    };
};

#endif // DIFFUSIONCONVECTIONREACTIONEQUATION_HPP
```

7.37 equation.hpp File Reference

```
#include <QString>
```

Classes

- class SemSolver::Equation< d, X >

  Virtual class for handling general equation.

Namespaces

- namespace SemSolver

  Project main namespace.

7.38 equation.hpp

```
#ifndef EQUATION_HPP
#define EQUATION_HPP

#include <QString>

namespace SemSolver
{
    template<int d, class X>
    class Equation;
}


namespace SemSolver
{

    template<int d, class X>
    class Equation
    {
    public:
        enum Type
        {
            NONE,
```

```
            DIFFUSION_CONVECTION_REACTION
        };

        Equation() {};

        virtual ~Equation() {};

        virtual Type type() const { return NONE; };

        virtual QString mml() const { return ""; };
    };
};

#endif // EQUATION_HPP
```

7.39 equation.hpp File Reference

```
#include <QFile>
#include <SemSolver/equation.hpp>
#include <SemSolver/diffusionconvectionreactionequation.hpp>
#include <SemSolver/scriptfunction.hpp>
```

Namespaces

- namespace SemSolver

  Project main namespace.

- namespace SemSolver::IO

  Namespace for Input/Output operations on SemSolver Classes.

Functions

- template<class X >
  bool SemSolver::IO::read_equation (QFile ∗file, Equation< 2, X > ∗&equation)

  Read 2D Equation from file.

7.40 equation.hpp

```
#ifndef IO_EQUATION_HPP
#define IO_EQUATION_HPP

#include <QFile>

#include <SemSolver/equation.hpp>
```

```

namespace SemSolver
{
    namespace IO
    {
        template<class X>
        bool read_equation(QFile *file,
                           Equation<2, X> *&equation);
    };
};

#include <SemSolver/diffusionconvectionreactionequation.hpp>
#include <SemSolver/scriptfunction.hpp>

template<class X>
bool SemSolver::IO::read_equation(QFile *file,
                                  Equation<2, X> *&equation)
{
#ifdef SEMDEBUG
    if(!file->open(QIODevice::ReadOnly))
    {
        qWarning("SemSolver::IO::readEquation - ERROR : cannot open file.");
        return false;
    }
#else
    file->open(QIODevice::ReadOnly);
#endif
    QTextStream input(file);
    QStringList values;
    values = next_non_empty_line_values(input);
#ifdef SEMDEBUG
    if(values.isEmpty())
    {
        qWarning("SemSolver::IO::readEquation - ERROR : missing type value in file.");
        file->close();
        return false;
    }
    if(values.size()>1)
    {
        qWarning("SemSolver::IO::readEquation - ERROR : too many inputs in first line of"\
                 " file.");
        file->close();
        return false;
    }
#endif
    if(values[0]=="DIFFUSION_CONVECTION_REACTION")
    {
        DiffusionConvectionReactionEquation<2, X> *new_equation =
                new DiffusionConvectionReactionEquation<2, X>();

        values = next_non_empty_line_values(input);
        while(!values.isEmpty())
        {
            if(values[0]=="DIFFUSION")
            {
#ifdef SEMDEBUG
                if(values.size()!=2)
                {
                    qWarning("SemSolver::IO::readEquation - ERROR : wrong number of inpu"\
```

```
                             "ts on diffusion line.");
                    file->close();
                    return false;
                }
#endif
                new_equation->setDiffusion(
                        new ScriptFunction< Point<2, X>, X >(values[1]));
            }
            else if(values[0]=="CONVECTION")
            {
#ifdef SEMDEBUG
                if(values.size()!=3)
                {
                    qWarning("SemSolver::IO::readEquation - ERROR : wrong number of inpu"\
                             "ts on convection line.");
                    file->close();
                    return false;
                }
#endif
                new_equation->setConvection(
                        new ScriptFunction< Point<2, X>, Vector<X> >(values.mid(1, 2)));
            }
            else if(values[0]=="REACTION")
            {
#ifdef SEMDEBUG
                if(values.size()!=2)
                {
                    qWarning("SemSolver::IO::readEquation - ERROR : wrong number of inpu"\
                             "ts on reaction line.");
                    file->close();
                    return false;
                }
#endif
                new_equation->setReaction(new ScriptFunction< Point<2, X>, X >(values[1]));
            }
            else if(values[0]=="FORCING")
            {
#ifdef SEMDEBUG
                if(values.size()!=2)
                {
                    qWarning("SemSolver::IO::readEquation - ERROR : wrong number of inpu"\
                             "ts on forcing line.");
                    file->close();
                    return false;
                }
#endif
                new_equation->setForcing(new ScriptFunction< Point<2, X>, X >(values[1]));
            }
#ifdef SEMDEBUG
            else
            {
                qWarning("SemSolver::IO::readEquation - ERROR : unknown input line in fi"\
                         "le.");
                file->close();
                return false;
```

```
            };
#endif
            values = next_non_empty_line_values(input);
        }
        delete equation;
        equation = new_equation;
    }
#ifdef SEMDEBUG
    else
    {
        qWarning("SemSolver::IO::readEquation - ERROR : unknown type value in fil
e.");
        file->close();
        return false;
    }
#endif
    file->close();

    return true;
};

#endif // IO_EQUATION_HPP
```

7.41 function.hpp File Reference

Classes

- class SemSolver::Function< X, Y >

  Prototype class for mathematical functions : $X \rightarrow Y$.

Namespaces

- namespace SemSolver

  Project main namespace.

7.42 function.hpp

```
#ifndef FUNCTION_HPP
#define FUNCTION_HPP

namespace SemSolver
{
    template<class X, class Y>
    class Function;
};


namespace SemSolver
{

    template<class X, class Y>
```

```
    class Function
    {

    public:

        Function() {};

        virtual ~Function() {};

        virtual Y evaluate(X const &) const { return Y(); };

        virtual QString mml() const { return ""; };
    };
};

#endif // FUNCTION_HPP
```

7.43 geometry.hpp File Reference

```
#include <QFile>
#include <SemSolver/semgeometry.hpp>
#include <SemSolver/IO/archive.hpp>
#include <SemSolver/IO/pslg.hpp>
#include <SemSolver/IO/subdomains.hpp>
```

Namespaces

- namespace SemSolver

  Project main namespace.

- namespace SemSolver::IO

  Namespace for Input/Output operations on SemSolver Classes.

Functions

- template< class X >
  bool SemSolver::IO::read_geometry (QFile ∗file, SemGeometry< 2, X > &geometry)

  Read SemGeometry form file.

- bool SemSolver::IO::write_geometry (QFile ∗pslg_file, QFile ∗domains_file, QFile ∗file)

  Write Geometry archive from pslg and subdomains files.

7.44 geometry.hpp

```
#ifndef IO_GEOMETRY_HPP
#define IO_GEOMETRY_HPP

#include <QFile>

#include <SemSolver/semgeometry.hpp>

#include <SemSolver/IO/archive.hpp>
#include <SemSolver/IO/pslg.hpp>
#include <SemSolver/IO/subdomains.hpp>

namespace SemSolver
{
    namespace IO
    {
        template<class X>
        bool read_geometry(QFile *file,
                          SemGeometry<2, X> &geometry)
{
    Archive archive(file);
    if(!archive.openRead())
        return false;
    QTemporaryFile *poly_file, *domains_file;
    poly_file = new QTemporaryFile();
    domains_file = new QTemporaryFile();
    if(!archive.extractFile("pslg.poly", poly_file))
    {
        delete poly_file;
        delete domains_file;
        return false;
    }
    if(!archive.extractFile("domains.semsub", domains_file))
    {
        delete poly_file;
        delete domains_file;
        return false;
    }
    if(!archive.closeRead())
    {
        delete poly_file;
        delete domains_file;
        return false;
    }
    PSLG<double> pslg;
    if(!SemSolver::IO::read_PSLG(poly_file, pslg))
    {
        delete poly_file;
        delete domains_file;
        return false;
    }
    Polygonation<2,double> sub_domains;
    if(!SemSolver::IO::read_subdomains(domains_file, sub_domains))
    {
        delete poly_file;
        delete domains_file;
        return false;
    }
    geometry.setDomain(pslg);
    geometry.setSubDomains(sub_domains);
    delete poly_file;
```

```
    delete domains_file;
    return true;
};

        bool write_geometry(QFile *pslg_file,
                            QFile *domains_file,
                            QFile *file);
    };
};

#endif // IO_GEOMETRY_HPP
```

7.45 hilbertspace.hpp File Reference

#include <cmath>

#include <vector>

Classes

- class SemSolver::HilbertSpace< Function, X >

  prototype class for handling the concept of Hilbert Space

- class SemSolver::HilbertSpace< Function, X >::Element

  Class for handling space elements as Fourier coefficients.

Namespaces

- namespace SemSolver

  Project main namespace.

7.46 hilbertspace.hpp

```
#ifndef HILBERTSPACE_HPP
#define HILBERTSPACE_HPP

#include <cmath>

#include <vector>

namespace SemSolver
{
    template<class Function, class X>
    class HilbertSpace
    {
        std::vector<Function> _base;

    public:
```

```
        class Element
        : std::vector<X>
        {
            HilbertSpace<Function, X> const *_space;

        public:
            Element(const HilbertSpace<Function, X> *space) : _space(space) { };

            virtual ~Element() {};
        };

        virtual Function const *baseFunction(int const &index) const
        {
#if SEMDEBUG
            if(index<0 || index>=dimension())
                qFatal("SemSolver::HilbertSpace::baseFunction - ERROR : index out
 of ran"\
                       "ge.");
#endif
            return &_base[index];
        };

        virtual X scalarProduct(Element const &,
                                Element const &) const
        {
            return X();
        };

        virtual inline X norm(Element element)
        {
            return std::sqrt(scalarProduct(element, element));
        };

        virtual inline int dimension() const
        {
            return _base.size();
        };

        virtual Element projection(Function const *) const
        {
            return Element(this);
        };
    };
};

#endif // HILBERTSPACE_HPP
```

7.47 homeomorphism.hpp File Reference

```
#include <SemSolver/function.hpp>
```

Classes

- class SemSolver::Homeomorphism< X, Y >

  Prototype class for mathematical homemorphism $: X \to Y$.

Namespaces

- namespace SemSolver

  Project main namespace.

7.48 homeomorphism.hpp

```
#ifndef HOMEOMORPHISM_HPP
#define HOMEOMORPHISM_HPP

#include <SemSolver/function.hpp>

namespace SemSolver
{
    template<class X, class Y>
    class Homeomorphism
    : public Function<X,Y>
    {
    public:
        Homeomorphism() {};

        virtual ~Homeomorphism() {};

        virtual X evaluateInverse(Y const &)
        {
            return X();
        };
    };
};

#endif // HOMEOMORPHISM_HPP
```

7.49 lusolve.hpp File Reference

```
#include <jama_lu.h>
#include <SemSolver/matrix.hpp>
#include <SemSolver/vector.hpp>
```

Namespaces

- namespace SemSolver

  Project main namespace.

- namespace SemSolver::Solver

  Solver namespace.

Functions

- template<class X >
  bool SemSolver::Solver::lu_solve (Matrix< X > const &A, Vector< X > const &b, Vector< X > &x)

7.50 lusolve.hpp

```
#ifndef LUSOLVER_HPP
#define LUSOLVER_HPP

#if defined _WIN32 || defined _WIN64
#       include <SemSolver/math_defines>
#endif

#include <jama_lu.h>

#include <SemSolver/matrix.hpp>
#include <SemSolver/vector.hpp>

namespace SemSolver
{

    namespace Solver
    {
        template<class X>
        bool lu_solve(Matrix<X> const &A,
                      Vector<X> const &b,
                      Vector<X> &x)
        {
            JAMA::LU<X> lu(A);
#ifdef SEMDEBUG
            if(!lu.isNonsingular())
            {
                qWarning("SemSolver::Solver::lu_solve - ERROR : Matrix A is singular.");
                return false;
            }
#endif
            x = lu.solve(b);
            return true;
        };
    };
};

#endif // LUSOLVER_HPP
```

7.51 matrix.hpp File Reference

```
#include <tnt_array2d.h>
#include <jama_eig.h>
#include <SemSolver/vector.hpp>
```

Classes

- class SemSolver::Matrix< X >

  Class for handling mathematical matrices.

Namespaces

- namespace SemSolver

  Project main namespace.

Functions

- template<class X >
  Matrix< X > SemSolver::operator* (Matrix< X > const &mat1, Matrix< X > const &mat2)

  Matrix multiplication.

- template<class X >
  Matrix< X > SemSolver::operator+ (Matrix< X > const &mat1, Matrix< X > const &mat2)

  Matrix summation.

7.52 matrix.hpp

```
#ifndef MATRIX_HPP
#define MATRIX_HPP

namespace SemSolver
{
    template<class X>
    class Matrix;
};

#include <tnt_array2d.h>
#include <jama_eig.h>

#include <SemSolver/vector.hpp>

namespace SemSolver
{
    template<class X>
    class Matrix
        : public TNT::Array2D<X>
    {
        typedef TNT::Array2D<X> Base;

    public:
        Matrix();
```

```
        Matrix(int rows, int columns);

        Matrix(int rows, int columns, X const &value);

        inline int rows() const;

        inline int columns() const;

        template<class Y>
        friend Matrix<Y> operator+(Matrix<Y> const &, Matrix<Y> const &);

        Vector<X> realEigenvalues() const;
    };

    template<class X>
    Matrix<X> operator *(Matrix<X> const &mat1, Matrix<X> const &mat2);

    template<class X>
    Matrix<X> operator +(Matrix<X> const &mat1, Matrix<X> const &mat2);
};


template<class X>
SemSolver::Matrix<X>::Matrix()
    : Base()
{
};

template<class X>
SemSolver::Matrix<X>::Matrix(int rows, int columns)
    : Base(rows, columns)
{
};

template<class X>
SemSolver::Matrix<X>::Matrix(int rows, int columns, X const &value)
    : Base(rows, columns, value)
{
};

template<class X>
inline int SemSolver::Matrix<X>::rows() const
{
    return Base::dim1();
};

template<class X>
inline int SemSolver::Matrix<X>::columns() const
{
    return Base::dim2();
};

template<class X>
SemSolver::Vector<X> SemSolver::Matrix<X>::realEigenvalues() const
{
    Vector<X> eigenvalues;
    JAMA::Eigenvalue<double> eig(*this);
    eig.getRealEigenvalues(eigenvalues);
    return eigenvalues;
};

```

```
template<class X>
SemSolver::Matrix<X> SemSolver::operator *(Matrix<X> const &mat1,
                                            Matrix<X> const &mat2)
{
    int r1 = mat1.dim1();
    int c1 = mat1.dim2();
    int r2 = mat2.dim1();
    int c2 = mat2.dim2();
#ifdef SEMDEBUG
    if(c1 != r2)
        qFatal("SemSolver::operator * - ERROR : the number of columns of mat1 must match"\
               " the number of rows of mat2.");
#endif
    Matrix<X> product(r1, c2,0);
    for (int i=0; i<r1; i++)
        for (int j=0; j<c2; j++)
            for (int k=0; k<c1; k++)
                product += mat1[i][k] * mat2[k][j];
    return product;
};

template<class X>
SemSolver::Matrix<X> SemSolver::operator +(Matrix<X> const &mat1,
                                            Matrix<X> const &mat2)
{
#ifdef SEMDEBUG
    int r1 = mat1.dim1();
    int c1 = mat1.dim2();
    int r2 = mat2.dim1();
    int c2 = mat2.dim2();
    if(c1 != c2)
        qFatal("SemSolver::operator * - ERROR : the number of columns of mat1 must match"\
               "the number of columns of mat2.");
    if(r1 != r2)
        qFatal("SemSolver::Matrix::operator * - ERROR : the number of rows of mat1 must "\
               "match the number of the rows of mat2.");
#endif
    return mat1+mat2;
};

#endif // MATRIX_HPP
```

7.53 multiindex.hpp File Reference

Classes

- class SemSolver::MultiIndex< N >

  Class multi-index notation.

- struct SemSolver::MultiIndex< N >::less

  Partial order.

Namespaces

- namespace SemSolver

  Project main namespace.

7.54 multiindex.hpp

```
#ifndef MULTIINDEX_HPP
#define MULTIINDEX_HPP

namespace SemSolver
{
    template<int N>
    class MultiIndex
    {
        int _indices[N];

    public:
        MultiIndex()
        {
            for(int i=0; i<N; ++i)
                _indices[i] = 0;
        };

        MultiIndex(int const sub_indices[N])
        {
            for(int i=0; i<N; ++i)
                _indices[i] = sub_indices[i];
        };

        MultiIndex(MultiIndex<N> const &index)
        {
            for(int i=0; i<N; ++i)
                _indices[i] = index.subIndex(i);
        };

        int const &subIndex(int const &index) const
        {
            if(index<0 || N<=index)
                qFatal("SemSolver::MultiIndex::subIndex - ERROR : index out of range.");
            return _indices[index];
        };

        void setSubIndex(int const &index, int const &sub_index)
        {
            if(index<0 || N<=index)
                qFatal("SemSolver::MultiIndex::subIndex - ERROR : index out of range.");
            _indices[index] = sub_index;
        };

        struct less
        {
            bool operator() (MultiIndex<N> const &mi0,
                             MultiIndex<N> const &mi1) const
            {
                for(int i=0; i<N; ++i)
```

```
                {
                    if(mi0.subIndex(i)<mi1.subIndex(i))
                        return true;
                    if(mi0.subIndex(i)>mi1.subIndex(i))
                        return false;
                }
                return false;
            };
        };
    };
};

#endif // MULTIINDEX_HPP
```

7.55 nextnonemptlinevalues.hpp File Reference

`#include <QStringList>`

`#include <QTextStream>`

Namespaces

- namespace SemSolver

    Project main namespace.

- namespace SemSolver::IO

    Namespace for Input/Output operations on SemSolver Classes.

Functions

- QStringList SemSolver::IO::next_non_empty_line_values (QTextStream &text_-stream)

    Get list of values on next non empty line in a text stream skipping comments.

7.56 nextnonemptlinevalues.hpp

```
#ifndef NEXTNONEMPTLINEVALUES_HPP
#define NEXTNONEMPTLINEVALUES_HPP

#include <QStringList>
#include <QTextStream>

namespace SemSolver
{
    namespace IO
    {
        QStringList next_non_empty_line_values(QTextStream &text_stream);
    };
```

```
};

#endif // NEXTNONEMPTLINEVALUES_HPP
```

7.57 parameters.hpp File Reference

`#include <QFile>`

`#include <QTextStream>`

`#include <SemSolver/semparameters.hpp>`

Namespaces

- namespace SemSolver

    Project main namespace.

- namespace SemSolver::IO

    Namespace for Input/Output operations on SemSolver Classes.

Functions

- template<class X >
  bool SemSolver::IO::read_parameters (QFile ∗file, SemParameters< X > &parameters)

    Read SemParameters from file.

7.58 parameters.hpp

```
#ifndef IO_PARAMETERS_HPP
#define IO_PARAMETERS_HPP

#include <QFile>
#include <QTextStream>

#include <SemSolver/semparameters.hpp>

namespace SemSolver
{
    namespace IO
    {
        template<class X>
        bool read_parameters(QFile *file,
                             SemParameters<X> &parameters);
    };
};


```

```
template<class X>
bool SemSolver::IO::read_parameters(QFile *file,
                                    SemParameters<X> &parameters)
{
    bool degree = false, tolerance = false, penality = false;
#ifdef SEMDEBUG
    if(!file->open(QIODevice::ReadOnly))
    {
        qWarning("SemSolver::IO::readParameters - ERROR : cannot open file.");
        return false;
    }
#else
    file->open(QIODevice::ReadOnly);
#endif
    QTextStream input(file);
    QStringList values;
    values = next_non_empty_line_values(input);
    while(!values.isEmpty())
    {
        if(values[0]=="DEGREE")
        {
#ifdef SEMDEBUG
            if(values.size()!=2)
            {
                qWarning("SemSolver::IO::readParameters - ERROR : wrong number of
  inpu"\
                         "ts on degree line.");
                file->close();
                return false;
            }
#endif
            parameters.setDegree(values[1].toInt(&degree));
        }
        else if(values[0]=="TOLERANCE")
        {
#ifdef SEMDEBUG
            if(values.size()!=2)
            {
                qWarning("SemSolver::IO::readParameters - ERROR : wrong number of
  inpu"\
                         "ts on tolerance line.");
                file->close();
                return false;
            }
#endif
            parameters.setTolerance(values[1].toDouble(&tolerance));
        }
        else if(values[0]=="PENALITY")
        {
#ifdef SEMDEBUG
            if(values.size()!=2)
            {
                qWarning("SemSolver::IO::readParameters - ERROR : wrong number of
  inpu"\
                         "ts on tolerance line.");
                file->close();
                return false;
            }
#endif
            parameters.setPenality(values[1].toDouble(&penality));
        }
#ifdef SEMDEBUG
```

```
        else
        {
            qWarning("SemSolver::IO::readParameters - ERROR : unknown input line
in fi"\
                     "le.");
            file->close();
            return false;
        };
#endif
        values = next_non_empty_line_values(input);
    }
    file->close();
    return (degree && tolerance && penality);
};


#endif // IO_PARAMETERS_HPP
```

7.59 point.hpp File Reference

```
#include <cmath>
#include <CGAL/Cartesian.h>
#include <CGAL/Filtered_kernel.h>
#include <CGAL/Point_2.h>
#include <CGAL/centroid.h>
```

Classes

- class SemSolver::Point< 2, X >

  Class for handling 2D euclidean points.

Namespaces

- namespace SemSolver

  Project main namespace.

7.60 point.hpp

```
#ifndef POINT_HPP
#define POINT_HPP

namespace SemSolver
{

    template <int d, class X>
    class Point;
```

```
};

#include <cmath>

#include <CGAL/Cartesian.h>
#include <CGAL/Filtered_kernel.h>
#include <CGAL/Point_2.h>
#include <CGAL/centroid.h>

namespace SemSolver
{


    template<class X>
    class Point<2,X>
        : public CGAL::Point_2< CGAL::Filtered_kernel< CGAL::Simple_cartesian<X> > >
    {

        typedef CGAL::Point_2< CGAL::Filtered_kernel< CGAL::Simple_cartesian<X> > >
                CGAL_Point;

    public:

        inline Point();

        inline Point(CGAL_Point const &cgal_point);

        inline Point(X const &x,
                     X const &y);
    };

};

template<class X>
inline SemSolver::Point<2, X>::Point()
    : CGAL_Point()
{};

template<class X>
inline SemSolver::Point<2, X>::Point(const CGAL_Point &cgal_point)
    : CGAL_Point(cgal_point)
{};

template<class X>
inline SemSolver::Point<2, X>::Point(const X &x,
                                     const X &y)
    : CGAL_Point(x, y)
{};

#endif // POINT_HPP
```

7.61 pointsbimap.hpp File Reference

```
#include <SemSolver/point.hpp>
#include <SemSolver/segment.hpp>
#include <SemSolver/pointsmap.hpp>
```

Classes

- class SemSolver::PointsBimap< 2, X >

    Class for handling bi-directional maps between 2D Points and Integers.

Namespaces

- namespace SemSolver

    Project main namespace.

7.62 pointsbimap.hpp

```
#ifndef POINTSBIMAP_HPP
#define POINTSBIMAP_HPP

namespace SemSolver
{
    template<int d, class X>
    class PointsBimap;
};

#include <SemSolver/point.hpp>
#include <SemSolver/segment.hpp>
#include <SemSolver/pointsmap.hpp>

namespace SemSolver
{

    template<int d, class X>
    class PointsBimap;


    template<class X>
    class PointsBimap<2, X>
        : private PointsMap<2, X, int>
    {
        typedef PointsMap<2, X, int> Base;
        typedef typename Base::Iterator Iterator;

    public:
        typedef typename Base::KeyType KeyType;
        typedef typename Base::MappedType MappedType;
        typedef typename Base::ValueType ValueType;
        typedef typename Base::ConstIterator ConstIterator;
        typedef typename Base::SizeType SizeType;

    private:
        int last_id;

        inline int new_id();

    public:

        inline PointsBimap(const X &tolerance);
```

```
        Iterator insert(const ValueType &x);

        inline MappedType insertPoint(const KeyType &x);

        inline const KeyType &point(const MappedType &y) const;

        inline const MappedType &id(const KeyType &x) const;

        inline void modifyPoint(const MappedType &y, const KeyType &x);

        inline void modifyId(const KeyType &x, const MappedType &y);

        inline void erasePoint(const KeyType &x);

        inline void eraseId(const MappedType &y);

        ConstIterator findPoint(const KeyType &x) const;

        Iterator findPoint(const KeyType &x);

        ConstIterator findId(const MappedType &y) const;

        Iterator findId(const MappedType &y);

        inline bool hasPoint(const KeyType &x) const;

        bool hasId(const MappedType &y) const;

        bool hasPointOn(const Segment<2, X> &s) const;

        using Base::begin;
        using Base::clear;
        using Base::has;
        using Base::isEmpty;
        using Base::end;
        using Base::erase;
        using Base::size;
    };
};

template<class X>
inline int SemSolver::PointsBimap<2, X>::new_id()
{
    return ++last_id;
};


template<class X>
inline SemSolver::PointsBimap<2, X>::PointsBimap(const X &tol)
    : Base(tol), last_id(0)
{
};

template<class X>
typename SemSolver::PointsBimap<2, X>::Iterator
        SemSolver::PointsBimap<2, X>::insert(const ValueType &x)
{
    if(!hasId(x.second))
        return Base::insert(x);
    return Base::end();
};
```

```
template<class X>
inline typename SemSolver::PointsBimap<2, X>::MappedType
        SemSolver::PointsBimap<2, X>::insertPoint(const KeyType &x)
{
    return Base::insert(ValueType(x,new_id()))->second;
};
template<class X>
const typename SemSolver::PointsBimap<2, X>::KeyType &
        SemSolver::PointsBimap<2, X>::point(const MappedType &y) const
{
    ConstIterator it = begin();
    while(it!=end())
    {
        if(it->second == y)
            break;
        else
            ++it;
    }
#ifdef SEMDEBUG
    if(it==end())
        qFatal("SemSolver::PointsBimap::point: no mapped value y.");
#endif
    return it->first;
};
template<class X>
const typename SemSolver::PointsBimap<2, X>::MappedType &
        SemSolver::PointsBimap<2, X>::id(const KeyType &x) const
{
    ConstIterator it = find(x);
#ifdef SEMDEBUG
    if(it==end())
        qFatal("SemSolver::PointsBimap::point: no key value x");
#endif
    return it->second;
};
template<class X>
inline void SemSolver::PointsBimap<2, X>::modifyPoint(const MappedType &y,
                                                       const KeyType &x)
{
#ifdef SEMDEBUG
    if(findPoint(x)!=end())
        qFatal("SemSolver:PointsBimap<2, X>::modifyPoint - ERROR : key value x must be u"\
               "nique.");
#endif
    eraseId(y);
    insert(ValueType(x,y));
};
template<class X>
inline void SemSolver::PointsBimap<2, X>::modifyId(const KeyType &x,
                                                    const MappedType &y)
{
#ifdef SEMDEBUG
    if(findId(y)!=end())
        qFatal("SemSolver:PointsBimap<2, X>::modifyId - ERROR : mapped value y must be u"\
               "nique.");
```

```
#endif
    erasePoint(x);
    insert(ValueType(x,y));
};

template<class X>
inline void SemSolver::PointsBimap<2, X>::erasePoint(const KeyType &x)
{
    Iterator it = findPoint(x);
    if(it!=end())
        erase(it);
};

template<class X>
inline void SemSolver::PointsBimap<2, X>::eraseId(const MappedType &y)
{
    Iterator it = findId(y);
    if(it!=end())
        erase(it);
};

template<class X>
inline typename SemSolver::PointsBimap<2, X>::ConstIterator
        SemSolver::PointsBimap<2, X>::findPoint(const KeyType &x) const
{
    return Base::find(x);
};

template<class X>
inline typename SemSolver::PointsBimap<2, X>::Iterator
        SemSolver::PointsBimap<2, X>::findPoint(const KeyType &x)
{
    return Base::find(x);
};

template<class X>
typename SemSolver::PointsBimap<2, X>::ConstIterator
        SemSolver::PointsBimap<2, X>::findId(const MappedType &id) const
{
    ConstIterator it = begin();
    while(it != end())
    {
        if(it->second == id)
            break;
        else
            ++it;
    }
    return it;
};

template<class X>
typename SemSolver::PointsBimap<2, X>::Iterator
        SemSolver::PointsBimap<2, X>::findId(const MappedType &id)
{
    Iterator it = begin();
    while(it != end())
    {
        if(it->second == id)
            break;
        else
            ++it;
    }
```

```
    return it;
};

template<class X>
inline bool SemSolver::PointsBimap<2, X>::hasPoint(const KeyType &x) const
{
    return findPoint(x)!=end();
};

template<class X>
bool SemSolver::PointsBimap<2, X>::hasId(const MappedType &y) const
{
    return findId(y)!=end();
};

template<class X>
inline bool SemSolver::PointsBimap<2, X>::hasPointOn(const Segment<2, X> &segment
) const
{
    for (ConstIterator it=begin(); it!=end(); ++it)
        if(it->first!=segment.source() && it->first!=segment.target() && segment.
has_on(it->first))
            return true;
    return false;
};

#endif // POINTSBIMAP_HPP
```

7.63 pointsmap.hpp File Reference

`#include <SemSolver/point.hpp>`

Classes

- class SemSolver::PointsMap< 2, X, Y >

  Class for handling maps with 2D Point as KeyType.

Namespaces

- namespace SemSolver

  Project main namespace.

7.64 pointsmap.hpp

```
#ifndef POINTSMAP_HPP
#define POINTSMAP_HPP

namespace SemSolver
{
```

```
    template<int d, class X, class Y>
    class PointsMap;
};

#include <SemSolver/point.hpp>

namespace SemSolver
{

    template<int d, class X, class Y>
    class PointsMap;

    template<class X, class Y>
    class PointsMap<2, X, Y>
    {
        typedef QList< QPair< Point<2, X>, Y> > list;

    protected:
        typedef typename list::iterator Iterator;

    public:
        typedef Point<2, X> KeyType;
        typedef Y MappedType;
        typedef typename list::value_type ValueType;
        typedef typename list::const_iterator ConstIterator;
        typedef typename list::size_type SizeType;

    private:
        list points;
        const double tolerance;
        bool less(const KeyType &x, const KeyType &y) const;
        inline bool equal(const KeyType &x, const KeyType &y) const;

    public:
        inline Iterator begin();
        inline Iterator end();
        inline Iterator find(const KeyType &x);

        inline PointsMap(const X &tolerance);
        inline ConstIterator begin() const;
        inline void clear();
        inline bool has(const KeyType &x) const;
        inline bool isEmpty() const;
        inline ConstIterator end() const;
        inline void erase(Iterator position);
        inline SizeType erase(const KeyType &x);
        inline void erase(Iterator first, Iterator last);
        inline ConstIterator find(const KeyType &x) const;
        inline Iterator insert(const ValueType &x);
        inline Iterator insert(const KeyType &x, const MappedType &y);
        Iterator insert(Iterator position, const ValueType &x);
        template<class InputIterator>
        void insert(InputIterator first, InputIterator last);
        inline SizeType size() const;
        inline MappedType &operator[](const KeyType &key);
    };
};
```

```
template<class X, class Y>
bool SemSolver::PointsMap<2, X, Y>::less(const KeyType &x, const KeyType &y) const
{
    if(x.x()+tolerance < y.x())
        return true;
    if(x.x()-tolerance > y.x())
        return false;
    if(x.y()+tolerance < y.y())
        return true;
    return false;
};

template<class X, class Y>
inline bool SemSolver::PointsMap<2, X, Y>::equal(const KeyType &x, const KeyType &y) const
{
    return !less(x,y) && !less(y,x);
};


template<class X, class Y>
inline SemSolver::PointsMap<2, X, Y>::PointsMap(const X &tol)
    : tolerance(tol)
{
};

template<class X, class Y>
inline typename SemSolver::PointsMap<2, X, Y>::Iterator
        SemSolver::PointsMap<2, X, Y>::begin()
{
    return points.begin();
};

template<class X, class Y>
inline typename SemSolver::PointsMap<2, X, Y>::ConstIterator
        SemSolver::PointsMap<2, X, Y>::begin() const
{
    return points.begin();
};

template<class X, class Y>
inline void SemSolver::PointsMap<2, X, Y>::clear()
{
    points.clear();
};

template<class X, class Y>
inline bool SemSolver::PointsMap<2, X, Y>::has(const KeyType &x) const
{
    return find(x)!=end();
};

template<class X, class Y>
inline bool SemSolver::PointsMap<2, X, Y>::isEmpty() const
{
    return points.empty();
};

template<class X, class Y>
inline typename SemSolver::PointsMap<2, X, Y>::Iterator
SemSolver::PointsMap<2, X, Y>::end()
```

```
{
    return points.end();
};

template<class X, class Y>
inline typename SemSolver::PointsMap<2, X, Y>::ConstIterator
        SemSolver::PointsMap<2, X, Y>::end() const
{
    return points.end();
};

template<class X, class Y>
inline void SemSolver::PointsMap<2, X, Y>::erase(Iterator position)
{
    points.erase(position);
};

template<class X, class Y>
inline typename SemSolver::PointsMap<2, X, Y>::SizeType
        SemSolver::PointsMap<2, X, Y>::erase(const KeyType &x)
{
    return points.erase(x);
};

template<class X, class Y>
inline void SemSolver::PointsMap<2, X, Y>::erase(Iterator first, Iterator last)
{
    points.erase(first, last);
};

template<class X, class Y>
inline typename SemSolver::PointsMap<2, X, Y>::ConstIterator
        SemSolver::PointsMap<2, X, Y>::find(const KeyType &x) const
{
    ConstIterator it = begin();
    while(it!=end())
    {
        if(less(it->first, x))
            ++it;
        else if(less(x, it->first))
            it=end();
        else
            break;
    }
    return it;
};

template<class X, class Y>
inline typename SemSolver::PointsMap<2, X, Y>::Iterator
        SemSolver::PointsMap<2, X, Y>::find(const KeyType &x)
{
    Iterator it = begin();
    while(it!=end())
    {
        if(less(it->first, x))
            ++it;
        else if(less(x, it->first))
            it=end();
        else
            break;
    }
    return it;
```

```
};

template<class X, class Y>
inline typename SemSolver::PointsMap<2, X, Y>::Iterator
        SemSolver::PointsMap<2, X, Y>::insert(const ValueType &x)
{
    return insert(begin(), x);
};

template<class X, class Y>
inline typename SemSolver::PointsMap<2, X, Y>::Iterator
        SemSolver::PointsMap<2, X, Y>::insert(const KeyType &x,
                                              const MappedType &y)
{
    return insert(ValueType(x,y));
};

template<class X, class Y>
typename SemSolver::PointsMap<2, X, Y>::Iterator
        SemSolver::PointsMap<2, X, Y>::insert(Iterator it, const ValueType &x)
{
    while(it!=begin())
    {
        if(less(x.first, it->first))
            --it;
        else
            break;
    }
    while(it!=end())
    {
        if(less(it->first, x.first))
            ++it;
        else
            break;
    }
    if(it==end() || less(x.first, it->first))
        return points.insert(it, x);
    return end();
};

template<class X, class Y>
template<class InputIterator>
void SemSolver::PointsMap<2, X, Y>::insert(InputIterator first, InputIterator last)
{
    Iterator it = begin();
    while(first!=last)
        it = insert(it, *first++);
    insert(it, last);
};

template<class X, class Y>
inline typename SemSolver::PointsMap<2, X, Y>::SizeType
        SemSolver::PointsMap<2, X, Y>::size() const
{
    return points.size();
};




template<class X, class Y>
```

```
inline typename SemSolver::PointsMap<2, X, Y>::MappedType &
        SemSolver::PointsMap<2, X, Y>::operator[](const KeyType &x)
{
    return insert(ValueType(x, MappedType()))->second;
};

#endif // POINTSMAP_HPP
```

7.65 pointsset.hpp File Reference

```
#include <SemSolver/point.hpp>
```

Classes

- class SemSolver::PointsSet< 2, X >

    Class for handling sets of 2D Points.

Namespaces

- namespace SemSolver

    Project main namespace.

7.66 pointsset.hpp

```
#ifndef POINTSSET_HPP
#define POINTSSET_HPP

namespace SemSolver
{
    template<int d, class X>
    class PointsSet;
};


#include <SemSolver/point.hpp>

namespace SemSolver
{

    template<int d, class X>
    class PointsSet;


    template<class X>
    class PointsSet<2, X>
    {
        typedef QList< Point<2, X> > list;

    protected:
```

```
        typedef typename list::iterator Iterator;

    public:
        typedef Point<2, X> KeyType;
        typedef Point<2, X> MappedType;
        typedef typename list::value_type ValueType;
        typedef typename list::const_iterator ConstIterator;
        typedef typename list::size_type SizeType;

    private:
        list points;
        const double tolerance;
        bool less(const KeyType &x, const KeyType &y);
        inline bool equal(const KeyType &x, const KeyType &y);

    protected:
        inline Iterator begin();
        inline Iterator end();
        inline Iterator find(const KeyType &x);

    public:
        inline PointsSet(const X &tolerance);
        inline ConstIterator begin() const;
        inline void clear();
        inline bool has(const KeyType &x) const;
        inline bool isEmpty() const;
        inline ConstIterator end() const;
        inline void erase(Iterator position);
        inline SizeType erase(const KeyType &x);
        inline void erase(Iterator first, Iterator last);
        inline ConstIterator find(const KeyType &x) const;
        inline Iterator insert(const ValueType &x);
        Iterator insert(Iterator position, const ValueType &x);
        template<class InputIterator>
        void insert(InputIterator first, InputIterator last);
        inline SizeType size() const;
        inline MappedType &operator[](const KeyType &key);
    };
};

template<class X>
bool SemSolver::PointsSet<2, X>::less(const KeyType &x, const KeyType &y)
{
    if(x.x()+tolerance < y.x())
        return true;
    if(x.x()-tolerance > y.x())
        return false;
    if(x.y()+tolerance < y.y())
        return true;
    return false;
};

template<class X>
inline bool SemSolver::PointsSet<2, X>::equal(const KeyType &x, const KeyType &y)

{
    return !less(x,y) && !less(y,x);
};


template<class X>
inline SemSolver::PointsSet<2, X>::PointsSet(const X &tol)
```

```
    : tolerance(tol)
{
};

template<class X>
inline typename SemSolver::PointsSet<2, X>::Iterator
        SemSolver::PointsSet<2, X>::begin()
{
    return points.begin();
};

template<class X>
inline typename SemSolver::PointsSet<2, X>::ConstIterator
        SemSolver::PointsSet<2, X>::begin() const
{
    return points.begin();
};

template<class X>
inline void SemSolver::PointsSet<2, X>::clear()
{
    points.clear();
};

template<class X>
inline bool SemSolver::PointsSet<2, X>::has(const KeyType &x) const
{
    return find(x)!=end();
};

template<class X>
inline bool SemSolver::PointsSet<2, X>::isEmpty() const
{
    return points.empty();
};

template<class X>
inline typename SemSolver::PointsSet<2, X>::Iterator
SemSolver::PointsSet<2, X>::end()
{
    return points.end();
};

template<class X>
inline typename SemSolver::PointsSet<2, X>::ConstIterator
        SemSolver::PointsSet<2, X>::end() const
{
    return points.end();
};

template<class X>
inline void SemSolver::PointsSet<2, X>::erase(Iterator position)
{
    points.erase(position);
};

template<class X>
inline typename SemSolver::PointsSet<2, X>::SizeType
        SemSolver::PointsSet<2, X>::erase(const KeyType &x)
{
    return points.erase(x);
};
```

```
template<class X>
inline void SemSolver::PointsSet<2, X>::erase(Iterator first, Iterator last)
{
    points.erase(first, last);
};

template<class X>
inline typename SemSolver::PointsSet<2, X>::ConstIterator
        SemSolver::PointsSet<2, X>::find(const KeyType &x) const
{
    ConstIterator it = begin();
    while(it!=end())
    {
        if(less(*it, x))
            ++it;
        else if(less(x, *it))
            it=end();
        else
            break;
    }
    return it;
};

template<class X>
inline typename SemSolver::PointsSet<2, X>::Iterator
        SemSolver::PointsSet<2, X>::find(const KeyType &x)
{
    Iterator it = begin();
    while(it!=end())
    {
        if(less(*it, x))
            ++it;
        else if(less(x, *it))
            it=end();
        else
            break;
    }
    return it;
};

template<class X>
inline typename SemSolver::PointsSet<2, X>::Iterator
        SemSolver::PointsSet<2, X>::insert(const ValueType &x)
{
    return insert(begin(), x);
};

template<class X>
typename SemSolver::PointsSet<2, X>::Iterator
        SemSolver::PointsSet<2, X>::insert(Iterator it, const ValueType &x)
{
    while(it!=begin())
    {
        if(less(x, *it))
            --it;
        else
            break;
    }
    while(it!=end())
    {
        if(less(*it, x) )
```

```
                ++it;
            else
                break;
        }
        if(it==end() || less(x, *it))
            return points.insert(it, x);
        return end();
};

template<class X>
template<class InputIterator>
void SemSolver::PointsSet<2, X>::insert(InputIterator first, InputIterator last)
{
    Iterator it = begin();
    while(first!=last)
        it = insert(it, *first++);
    insert(it, last);
};

template<class X>
inline typename SemSolver::PointsSet<2, X>::SizeType
SemSolver::PointsSet<2, X>::size() const
{
    return points.size();
};




template<class X>
inline typename SemSolver::PointsSet<2, X>::MappedType &
        SemSolver::PointsSet<2, X>::operator[](const KeyType &x)
{
    return insert(ValueType(x, MappedType()))->second;
};

#endif // POINTSSET_HPP
```

7.67 polygon.hpp File Reference

```
#include <CGAL/Simple_cartesian.h>
#include <CGAL/Filtered_kernel.h>
#include <CGAL/Polygon_2.h>
#include <SemSolver/point.hpp>
```

Classes

- class SemSolver::Polygon< 2, X >

  Class for handling 2D polygons.

Namespaces

- namespace SemSolver

    Project main namespace.

7.68 polygon.hpp

```
#ifndef POLYGON_HPP
#define POLYGON_HPP

namespace SemSolver
{

    template<int d, class X>
    class Polygon;
};

#include <CGAL/Simple_cartesian.h>
#include <CGAL/Filtered_kernel.h>
#include <CGAL/Polygon_2.h>

#include <SemSolver/point.hpp>

namespace SemSolver
{


    template<class X>
    class Polygon<2, X>
        : public CGAL::Polygon_2< CGAL::Filtered_kernel< CGAL::Simple_cartesian<X
> > >
    {

        typedef CGAL::Point_2< CGAL::Filtered_kernel< CGAL::Simple_cartesian<X> >
 >
                CGAL_point;

        typedef CGAL::Polygon_2< CGAL::Filtered_kernel< CGAL::Simple_cartesian<X>
 > >
                CGAL_polygon;

    public:

        inline Polygon();

        inline Polygon(CGAL_point const *first,
                       CGAL_point const *last);

        inline bool contains(Point<2, X> const &point) const;

    };

};

template<class X>
inline SemSolver::Polygon<2, X>::Polygon()
    : CGAL_polygon()
{};
```

```
template<class X>
inline SemSolver::Polygon<2, X>::Polygon(const CGAL_point *first,
                                        const CGAL_point *last)
                                            : CGAL_polygon(first, last)
{};

template<class X>
inline bool SemSolver::Polygon<2, X>::contains(const Point<2, X> &point) const
{
    return bounded_side(point) != CGAL::ON_UNBOUNDED_SIDE;
};

#endif // POLYGON_HPP
```

7.69 polygonation.hpp File Reference

```
#include <vector>
#include <SemSolver/polygon.hpp>
#include <SemSolver/point.hpp>
```

Classes

- class SemSolver::Polygonation< 2, X >

  Class for handling 2D polygonations.

- class SemSolver::Polygonation< 2, X >::Element

  A Polygonation element.

Namespaces

- namespace SemSolver

  Project main namespace.

7.70 polygonation.hpp

```
#ifndef POLYGONATION_HPP
#define POLYGONATION_HPP

namespace SemSolver
{

    template<int d, class X>
    class Polygonation;
};

```

```
#include <vector>

#include <SemSolver/polygon.hpp>
#include <SemSolver/point.hpp>

namespace SemSolver
{


    template<class X>
    class Polygonation<2,X>
    {

    public:


        class Element
        {

        private:
            Polygon<2, X>    polygon;
            std::vector<int> neighbours_vector;

        public:
            inline Element();

            inline Element(Polygon<2, X> const &geometry,
                           std::vector<int> const &neighbours);

            inline Polygon<2, X> const &geometry() const;


            unsigned vertexPosition(Point<2, X> const &vertex) const;

            inline void clear();

            inline Point<2, X> vertex(int const &index) const;

            inline int size() const;

            inline typename Polygon<2, X>::Vertex_const_iterator verticesBegin() const;

            inline typename Polygon<2, X>::Vertex_const_iterator verticesEnd() const;

            inline std::vector<int>::const_iterator neighboursBegin() const;

            inline std::vector<int>::const_iterator neighboursEnd() const;

            inline void setGeometry(Point<2,X> const *first,
                                    Point<2,X> const *last);

            inline void setNeighbour(const unsigned &index,
                                     int const &id);

            inline int neighbour(const unsigned &index) const;

            inline bool contains(Point<2, X> const &point) const;

        };

```

```
    private:
        std::vector<Element> elements;

    public:
        bool isQuadrangulation() const;


        void refine();

        inline void clear();

        inline const Element &element(const unsigned &index) const;

        inline unsigned size() const;

        inline void addElement(Polygon<2, X> const &polygon,
                               std::vector<int> const &neighbours);

        std::vector<unsigned> elementIndicesAt(Point<2, X> const &point) const;
    };
};

template<class X>
bool SemSolver::Polygonation<2, X>::isQuadrangulation() const
{
    for(typename std::vector<Element>::const_iterator it = elements.begin();
    it != elements.end(); ++it)
    {
        if(it->size()!=4)
            return false;
    }
    return true;
};

template<class X>
void SemSolver::Polygonation<2, X>::refine()
{
    int *subelements_first = new int[size()];
    subelements_first[0]=0;
    for(unsigned i=0; i<size()-1; ++i)
        subelements_first[i+1]=subelements_first[i]+element(i).size();
    std::vector<Element> subelements;
    for(unsigned i=0; i<size(); ++i)
    {
        Element const &old_element = element(i);
        Point<2,X> cent = centroid(old_element.verticesBegin(),
                                   old_element.verticesEnd());
        Point<2,X> vertices[4];
        Polygonation::Element subelement;
        int n = old_element.size();
        for(int j=0; j<n; ++j)
        {
            vertices[0] = old_element.vertex(j);
            vertices[1] = midpoint(old_element.vertex(j),
                                   old_element.vertex((j+1)%n));
            vertices[2] = cent;
            vertices[3] = midpoint(old_element.vertex(j),
                                   old_element.vertex((j+n-1)%n));
            subelement.setGeometry(vertices, vertices+4);
            if(old_element.neighbour(j)<0) // border
                subelement.setNeighbour(0,old_element.neighbour(j));
            else
```

```
            {
                int neigh = old_element.neighbour(j)-1;
                int neigh_first = subelements_first[neigh];
                element(neigh);
                old_element.vertex(j);
                int pos = element(neigh).vertexPosition(old_element.vertex(j));
                subelement.setNeighbour(0,neigh_first+pos+1);
            }
            if(old_element.neighbour((j+1)%n)<0)
                subelement.setNeighbour(1, old_element.neighbour((j+1)%n));
            else
            {
                int neigh = old_element.neighbour((j+1)%n)-1;
                int neigh_first = subelements_first[neigh];
                int pos = element(neigh).vertexPosition(old_element.vertex(j));
                subelement.setNeighbour(1, neigh_first+pos+1);
            }
            subelement.setNeighbour(2,subelements_first[i]+(j+1)%n+1);
            subelement.setNeighbour(3,subelements_first[i]+(j+n-1)%n+1);
            subelements.push_back(subelement);
        }
    }
    elements.clear();
    elements = subelements;
};

template<class X>
inline void SemSolver::Polygonation<2, X>::clear()
{
    elements.clear();
};

template<class X>
inline const typename SemSolver::Polygonation<2, X>::Element
        &SemSolver::Polygonation<2, X>::element(const unsigned &index) const
{
#ifdef SEMDEBUG
    if(index>=elements.size())
        qFatal("SemSolver::Polygonation::element - ERROR : index out of bounds.")
;
#endif //SEMDEBUG
    return elements[index];
};

template<class X>
inline unsigned SemSolver::Polygonation<2, X>::size() const
{
    return elements.size();
};

template<class X>
inline void SemSolver::Polygonation<2, X>::addElement(Polygon<2, X> const &polygo
n,
                                                      std::vector<int> const &nei
ghbours)
{
    elements.push_back(Element(polygon, neighbours));
};

template<class X>
std::vector<unsigned> SemSolver::Polygonation<2, X>::elementIndicesAt(
        SemSolver::Point<2, X> const &point) const
```

```
{
    std::vector<unsigned> indices;
    for(unsigned i=0; i<size(); ++i)
        if(element(i).contains(point))
            indices.push_back(i);
    return indices;
};

template<class X>
inline SemSolver::Polygonation<2, X>::Element::Element()
{};

template<class X>
inline SemSolver::Polygonation<2, X>::Element::Element(Polygon<2, X> const &geometry,
                                                        std::vector<int> const &neighbours)
{
    polygon = geometry;
    neighbours_vector = neighbours;
};

template<class X>
inline SemSolver::Polygon<2, X> const &SemSolver::Polygonation<2, X>::Element::geometry()
        const
{
    return polygon;
}

template<class X>
unsigned SemSolver::Polygonation<2, X>::Element::vertexPosition(Point<2, X> const &vertex)
        const
{
    for(unsigned i=0; i<polygon.size(); ++i)
    {
        if(polygon.vertex(i) == vertex)
        {
            return i;
        }
    }
    return -1;
};

template<class X>
inline void SemSolver::Polygonation<2, X>::Element::clear()
{
    polygon.clear();
    neighbours_vector.clear();
};

template<class X>
inline SemSolver::Point<2, X> SemSolver::Polygonation<2, X>::Element::vertex(
        int const &index) const
{
    return polygon.vertex(index);
};

template<class X>
inline int SemSolver::Polygonation<2, X>::Element::size() const
{
```

```
    return polygon.size();
};

template<class X>
inline typename SemSolver::Polygon<2, X>::Vertex_const_iterator
        SemSolver::Polygonation<2, X>::Element::verticesBegin() const
{
    return polygon.vertices_begin();
};

template<class X>
inline typename SemSolver::Polygon<2, X>::Vertex_const_iterator
        SemSolver::Polygonation<2, X>::Element::verticesEnd() const
{
    return polygon.vertices_end();
};

template<class X>
inline std::vector<int>::const_iterator
        SemSolver::Polygonation<2, X>::Element::neighboursBegin() const
{
    return neighbours_vector.begin();
};

template<class X>
inline std::vector<int>::const_iterator
        SemSolver::Polygonation<2, X>::Element::neighboursEnd() const
{
    return neighbours_vector.end();
};

template<class X>
inline void SemSolver::Polygonation<2, X>::Element::setGeometry(Point<2,X> const *first,
                                                                 Point<2,X> const *last)
{
    polygon = Polygon<2,X>(first, last);
    neighbours_vector.resize(polygon.size(),0);
};

template<class X>
inline void SemSolver::Polygonation<2, X>::Element::setNeighbour(const unsigned &index,
                                                                  int const &id)
{
#ifdef SEMDEBUG
    if(index>=neighbours_vector.size())
        qFatal("SemSolver::Polygonation::Element::setNeighbour - ERROR : index out of bo"\
               "unds.");
#endif //SEMDEBUG
    neighbours_vector[index] = id;
};

template<class X>
inline int SemSolver::Polygonation<2, X>::Element::neighbour(const unsigned &index)
        const
{
#ifdef SEMDEBUG
    if(index>=neighbours_vector.size())
```

```
        qFatal("SemSolver::Polygonation::Element::neighbour - ERROR : index out o
f bound"\
               "s.");
#endif //SEMDEBUG
    return neighbours_vector[index];
};

template<class X>
inline bool SemSolver::Polygonation<2, X>::Element::contains(Point<2, X> const &p
oint)
        const
{
    return polygon.contains(point);
};

#endif // POLYGONATION_HPP
```

7.71 polygonwithholes.hpp File Reference

```
#include <CGAL/Cartesian.h>
#include <CGAL/Filtered_kernel.h>
#include <CGAL/Polygon_with_holes_2.h>
#include <SemSolver/polygon.hpp>
```

Classes

- class SemSolver::PolygonWithHoles< 2, X >

  Class for handling 2D polygons with holes.

Namespaces

- namespace SemSolver

  Project main namespace.

7.72 polygonwithholes.hpp

```
#ifndef POLYGONWITHHOLES_HPP
#define POLYGONWITHHOLES_HPP

namespace SemSolver
{

    template <int d, class X>
            class PolygonWithHoles;
}

#include <CGAL/Cartesian.h>
```

```
#include <CGAL/Filtered_kernel.h>
#include <CGAL/Polygon_with_holes_2.h>

#include <SemSolver/polygon.hpp>

namespace SemSolver
{


    template<class X>
    class PolygonWithHoles<2,X>
        : public CGAL::Polygon_with_holes_2<
        CGAL::Filtered_kernel< CGAL::Simple_cartesian<X> > >
    {
        typedef CGAL::Polygon_with_holes_2<
                CGAL::Filtered_kernel< CGAL::Simple_cartesian<X> > >
                CGAL_Polygon_with_holes;
    public:
        inline CGAL_Polygon_with_holes const &cgal()
        {
            return *this;
        };
    };
};

#endif // POLYGONWITHHOLES_HPP
```

7.73 polynomial.hpp File Reference

`#include <vector>`

`#include <SemSolver/function.hpp>`

`#include <SemSolver/matrix.hpp>`

Classes

- class SemSolver::Polynomial< X >

    Class for handing the mathematical concept of polynomials over X.

Namespaces

- namespace SemSolver

    Project main namespace.

7.74 polynomial.hpp

```
#ifndef POLYNOMIAL_HPP
#define POLYNOMIAL_HPP
```

```
#include <vector>
#include <SemSolver/function.hpp>
#include <SemSolver/matrix.hpp>
namespace SemSolver
{
    template<class X>
    class Polynomial
    {
    private:
        // members
        std::vector<X> _coefficients;
    public:
        Polynomial()
            : _coefficients(1,0.)
        {
        }
        Polynomial(X const &scalar)
            : _coefficients(1,scalar)
        {
        }
        Polynomial(std::vector<X> const &coefficients)
            : _coefficients(coefficients)
        {
        }
        Polynomial(Polynomial const &poly)
            : _coefficients(poly._coefficients)
        {
        }
        ~Polynomial()
        {};
        Polynomial & operator = (Polynomial const &poly)
                                {
            _coefficients = poly._coefficients;
            return *this;
        }
        Polynomial & operator += (Polynomial const &poly)
                                 {
            int deg = poly.degree();
            if(deg>degree())
                _coefficients.resize(deg+1,0.);
            for(int i=0; i<=deg; ++i)
                _coefficients[i] += poly._coefficients[i];
            return *this;
        }
        Polynomial & operator -= (Polynomial const &poly)
                                 {
            int deg = poly.degree();
            if(deg>degree())
                _coefficients.resize(deg+1,0.);
            for(int i=0; i<=deg; ++i)
                _coefficients[i] -= poly._coefficients[i];
```

```
            return *this;
        }

        Polynomial & operator *= (Polynomial const &poly)
                                 {
            int deg = degree();
            int deg_ = poly.degree();
            std::vector<double> product;
            product.resize(deg+deg_+1,0.);
            for(int i=0; i<=deg; ++i)
                for(int j=0; j<=deg_; ++j)
                    product[i+j] += _coefficients[i]*poly._coefficients[j];
            _coefficients = product;
            return *this;
        }

        Polynomial & operator /= (Polynomial const &poly)
                                 {
            int deg = degree();
            int deg_ = poly.degree();
            Polynomial quotient, residual;
            residual = *this;
            {
                for(int i=deg; i>=deg_; --i)
                {
                    if (i>=0 && residual._coefficients[i])
                    {
                        Polynomial temp;
                        temp._coefficients.resize(deg-deg_+1,double(0));
                        temp._coefficients[i-deg_] =
                                residual._coefficients[i]/poly._coefficients[deg_];
                        quotient+=temp;
                        residual-=poly*temp;
                    }
                }
            }
            _coefficients = quotient._coefficients;
            return *this;
        }

        Polynomial & operator %= (const Polynomial &poly)
                                 {
            int deg = degree();
            int deg_ = poly.degree();
            Polynomial quotient, residual;
            residual = *this;
            {
                for(int i=deg; i>=deg_; --i)
                {
                    if (residual._coefficients[i])
                    {
                        Polynomial temp;
                        temp._coefficients.resize(deg-deg_+1,0.);
                        temp._coefficients[i-deg_] =
                                residual._coefficients[i]/poly._coefficients[deg_];
                        quotient+=temp;
                        residual-=poly*temp;
                    }
                }
            }
```

```
        _coefficients = residual._coefficients;
        return *this;
    }

    Polynomial operator + () const
    {
        return *this;
    }

    Polynomial operator - () const
    {
        Polynomial result(*this);
        for(std::vector<double>::iterator it=result._coefficients.begin();
        it!=result._coefficients.end(); ++it)
            *it = -*it;
        return result;
    }

    Polynomial operator + (const Polynomial &poly) const
    {
        Polynomial sum = *this;
        sum += poly;
        return sum;
    }

    Polynomial operator - (const Polynomial &poly) const
    {
        Polynomial difference = *this;
        difference -= poly;
        return difference;
    }

    Polynomial operator * (const Polynomial &poly) const
    {
        Polynomial product = *this;
        product *= poly;
        return product;
    }

    Polynomial operator / (const Polynomial &poly) const
    {
        Polynomial quotient = *this;
        quotient *= poly;
        return quotient;
    }

    Polynomial operator % (const Polynomial &poly) const
    {
        Polynomial residual = *this;
        residual %= poly;
        return residual;
    }

    bool operator == (const Polynomial &poly) const
    {
        if(degree()!=poly.degree())
            return false;
        else
            for(int i=0; i<degree(); ++i)
                if(_coefficients[i]!=poly._coefficients[i])
                    return 0;
        return 1;
```

```
        }

        bool operator != (const Polynomial &poly) const;

        double operator () (const X &variable) const
        {
            double value=0;
            for(int i=degree(); i>0; --i)
                value = (value + _coefficients[i]) * variable;
            return value+_coefficients[0];
        }

        inline double coefficient(const int &order) const
        {
#ifdef SEMDEBUG
            if(order<0 || order>degree())
                qFatal("SemSolver::Polynomial::coefficient - ERROR : order must be a non"\
                       " negative value not greater than degree");
#endif
            return _coefficients[order];
        }

        int degree() const
        {
            int deg = _coefficients.size()-1;
            while(deg>=0 && !_coefficients[deg])
                --deg;
            return deg;
        }

        inline void setCoefficient(const int &order, X const &coefficient)
        {
#ifdef SEMDEBUG
            if(order<0 || (long)order>=(long)_coefficients.size() )
                qFatal("SemSolver::Polynomial::setCoefficient - ERROR : order "\
                       "out of range.");
#endif
            _coefficients[order]=coefficient;
        };

        inline void setDegree(const int &degree)
        {
#ifdef SEMDEBUG
            if(degree<-1)
                qFatal("SemSolver::Polynomial::setDegree - ERROR : degree must be not le"\
                       "ss than -1.");
#endif
            _coefficients.resize(degree+1,0.);
        };

        Polynomial derivative() const
        {
            int deg = degree();
            if(deg < 1)
                return Polynomial(0.);
            std::vector<double> derivative;
            derivative.resize(deg,0.);
            for(int i=0; i<deg; ++i)
                derivative[i] = (i+1)*_coefficients[i+1];
            return Polynomial(derivative);
```

```
        }

        Polynomial ruffini(X const &zero) const
        {
            std::vector<double> result;
            int deg=degree();
            result.resize(deg+1,double(0.));
            for(int i=deg; i>0; --i)
                result[i-1] = _coefficients[i] + zero * result[i];
            return Polynomial(result);
        }

        std::vector<X> zeros() const
        {
            int deg = degree();
            Matrix<X> companion(deg,deg,0.);
            for(int i=0; i<deg-1; ++i)
            {
                companion[i+1][i] = 1;
                companion[i][deg-1] = -_coefficients[i]/_coefficients[deg];
            }
            Vector<X> eigenvalues(companion.realEigenvalues());
            std::vector<double> zeros;
            for(int i=0; i< deg; ++i)
                zeros.push_back(eigenvalues[i]);
            return zeros;
        }
    };
}

#endif // POLYNOMIAL_HPP
```

7.75 polynomialfunction.hpp File Reference

```
#include <SemSolver/polynomial.hpp>
#include <SemSolver/function.hpp>
#include <SemSolver/point.hpp>
```

Classes

- class SemSolver::PolynomialFunction< d, X >

  Class for handling polynomial separable functions.

Namespaces

- namespace SemSolver

  Project main namespace.

7.76 polynomialfunction.hpp

```
#ifndef POLYNOMIALFUNCTION_HPP
#define POLYNOMIALFUNCTION_HPP

#include <SemSolver/polynomial.hpp>
#include <SemSolver/function.hpp>
#include <SemSolver/point.hpp>

namespace SemSolver
{



    template<int d, class X>
    class PolynomialFunction : public Function< Point<d,X>, X >
    {
    private:
        Polynomial<X> _polynomials[d];

    public:
        PolynomialFunction()
        {
            for(int i=0; i<d; ++i)
                _polynomials[i] = 0.;
        };

        PolynomialFunction(Polynomial<X> const *polynomials)
        {
            for(int i=0; i<d; ++i)
                _polynomials[i] = polynomials[i];
        };

        PolynomialFunction(PolynomialFunction<d,X> const &polynomial_function)
        {
            for(int i=0; i<d; ++i)
            {
                _polynomials[i] = polynomial_function.polynomial(i);
            }
        };

        ~PolynomialFunction()
        {};

        PolynomialFunction<d,X> &operator=(PolynomialFunction<d,X> const &polynom
ial_function)
                                          {
            for(int i=0; i<d; ++i)
                _polynomials[i] = polynomial_function.polynomial(i);
            return *this;
        };

        Polynomial<X> const &polynomial(int const &index) const
        {
#ifdef SEMDEBUG
            if(index<0 || index>=d)
                qFatal("SemSolver::PolynomialFunction::polynomial - ERROR : index
 out of"\
                       "range");
#endif
            return _polynomials[index];
        };
```

```
        void setPolynomial(int const &index, Polynomial<X> const &poly)
        {
#ifdef SEMDEBUG
            if(index<0 || index>=d)
                qFatal("SemSolver::PolynomialFunction::setPolynomial ERROR : inde
x out o"\
                       "f range");
#endif
            _polynomials[index] = poly;
        };

        X evaluate(Point<d,X> const &x) const
        {
            double result = 1;
            for(int i=0; i<d; ++i)
                result *= polynomial(i)(x.cartesian(i));
            return result;
        };
    };
};

#endif // POLYNOMIALFUNCTION_HPP
```

7.77 problem.hpp File Reference

```
#include <SemSolver/semgeometry.hpp>
#include <SemSolver/equation.hpp>
#include <SemSolver/boundaryconditions.hpp>
#include <SemSolver/semparameters.hpp>
```

Classes

- class SemSolver::Problem< d, X >

  Class for handling a mathematical problem given by an equation, boundary conditions, geometry and parameters.

Namespaces

- namespace SemSolver

  Project main namespace.

7.78 problem.hpp

```
#ifndef PROBLEM_HPP
#define PROBLEM_HPP
```

```

namespace SemSolver
{
    template<int d, class X>
    class Problem;
}

#include <SemSolver/semgeometry.hpp>
#include <SemSolver/equation.hpp>
#include <SemSolver/boundaryconditions.hpp>
#include <SemSolver/semparameters.hpp>

namespace SemSolver
{
    template<int d, class X>
    class Problem
    {
        const SemGeometry<d, X>         *_geometry;
        const Equation<d, X>            *_equation;
        const BoundaryConditions<d, X> *_boundary_conditions;
        const SemParameters<X>          *_parameters;

    public:
        Problem();
        ~Problem();
        inline void setGeometry(const SemGeometry<d, X> *geometry);
        inline void setEquation(const Equation<d, X> *equation);
        inline void setBoundaryConditions(
                const BoundaryConditions<d, X> *boundary_conditions);
        inline void setParameters(const SemParameters<X> *parameters);

        inline void clearGeometry();
        inline void clearEquation();
        inline void clearBoundaryConditions();
        inline void clearParameters();

        inline const SemGeometry<d, X> *geometry() const;
        inline const Equation<d, X> *equation() const;
        inline const BoundaryConditions<d, X> *boundaryConditions() const;
        inline const SemParameters<X> *parameters() const;

        inline bool isDefined();
    };
};

template<int d, class X>
SemSolver::Problem<d, X>::Problem()
{
    _geometry = 0;
    _equation = 0;
    _boundary_conditions = 0;
    _parameters = 0;
};

template<int d, class X>
SemSolver::Problem<d, X>::~Problem()
{
    delete _geometry;
    delete _equation;
    delete _boundary_conditions;
    delete _parameters;
};
```

```

template<int d, class X>
inline void SemSolver::Problem<d, X>::setGeometry(const SemGeometry<d, X> *geometry)
{
    _geometry = geometry;
};

template<int d, class X>
inline void SemSolver::Problem<d, X>::setEquation(const Equation<d, X> *equation)

{
    _equation = equation;
};

template<int d, class X>
inline void SemSolver::Problem<d, X>::setBoundaryConditions(
        const BoundaryConditions<d, X> *boundary_conditions)
{
    _boundary_conditions = boundary_conditions;
};

template<int d, class X>
inline void SemSolver::Problem<d, X>::setParameters(const SemParameters<X> *parameters)
{
    _parameters = parameters;
};

template<int d, class X>
inline void SemSolver::Problem<d, X>::clearGeometry()
{
    delete _geometry;
    _geometry = 0;
};

template<int d, class X>
inline void SemSolver::Problem<d, X>::clearEquation()
{
    delete _equation;
    _equation = 0;
};

template<int d, class X>
inline void SemSolver::Problem<d, X>::clearBoundaryConditions()
{
    delete _boundary_conditions;
    _boundary_conditions = 0;
};

template<int d, class X>
inline void SemSolver::Problem<d, X>::clearParameters()
{
    delete _parameters;
    _parameters = 0;
};

template<int d, class X>
inline const SemSolver::SemGeometry<d, X> *SemSolver::Problem<d, X>::geometry() const
{
    return _geometry;
```

```
};

template<int d, class X>
inline const SemSolver::Equation<d, X> *SemSolver::Problem<d, X>::equation() const
{
    return _equation;
};

template<int d, class X>
inline const SemSolver::BoundaryConditions<d, X> *
        SemSolver::Problem<d, X>::boundaryConditions() const
{
    return _boundary_conditions;
};

template<int d, class X>
inline const SemSolver::SemParameters<X> *SemSolver::Problem<d, X>::parameters() const
{
    return _parameters;
};

template<int d, class X>
inline bool SemSolver::Problem<d, X>::isDefined()
{
    return ( _geometry && _equation && _boundary_conditions && _parameters );
};

#endif // PROBLEM_HPP
```

7.79 pslg.hpp File Reference

```
#include <QFile>
#include <SemSolver/pslg.hpp>
#include <SemSolver/IO/nextnonemptlinevalues.hpp>
```

Namespaces

- namespace SemSolver

    Project main namespace.

- namespace SemSolver::IO

    Namespace for Input/Output operations on SemSolver Classes.

Functions

- template<class X >
  bool SemSolver::IO::read_PSLG (QFile *file, PSLG< X > &pslg)

7.80 pslg.hpp

```
#ifndef IO_PSLG_HPP
#define IO_PSLG_HPP

#include <QFile>

#include <SemSolver/pslg.hpp>
#include <SemSolver/IO/nextnonemptlinevalues.hpp>

namespace SemSolver
{
    namespace IO
    {
        template<class X>
        bool read_PSLG(QFile *file,
                       PSLG<X> &pslg)
{
    pslg.clear();

#ifdef SEMDEBUG
    bool is_open =
#endif
            file->open(QIODevice::ReadOnly);
#ifdef SEMDEBUG
    if(!is_open)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : cannot open 'poly_file'");
        return false;
    }
#endif
    QTextStream input(file);
    QStringList values;
    values = next_non_empty_line_values(input);
#ifdef SEMDEBUG
    if(values.isEmpty())
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : missing 'number_of_vertices'"
);
        file->close();
        return false;
    }
#endif
    bool ok;
    int number_of_vertices = values[0].toInt(&ok);
#ifdef SEMDEBUG
    if(!ok)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_vertices' must be
an inte"\
                 "ger");
        file->close();
        return false;
    }
    if(number_of_vertices < 0)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_vertices' must be
non-neg"\
                 "ative");
        file->close();
        return false;
    }
```

```
#endif
    pslg.setNumberOfVertices(number_of_vertices);
#ifdef SEMDEBUG
    if(values.size()<2)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : missing 'dimension'");
        file->close();
        return false;
    }
    int dimension = values[1].toInt(&ok);
    if(!ok)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : 'dimension' must be an integer");
        file->close();
        return false;
    }
    if(dimension != 2)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : 'dimension' must be 2");
        file->close();
        return false;
    }
    if(values.size()<3)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : missing 'number_of_attributes'");
        file->close();
        return false;
    }
    int number_of_vertices_attributes = values[2].toInt(&ok);
    if(!ok)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_vertices_attributes' must"\
                 " be an integer");
        file->close();
        return false;
    }
    if(number_of_vertices_attributes < 0)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_vertices_attributes' must"\
                 " be non-negative");
        file->close();
        return false;
    }
    if(number_of_vertices_attributes > 0)
        qWarning("SemSolver::IO::readPSLG - ERROR : ignoring vertices attributes in 'pol"\
                 "y_file'");
#endif
    pslg.setNumberOfVerticesAttributes(0);
#ifdef SEMDEBUG
    if(values.size()<4)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : missing 'number_of_vertices_boundary"\
                 "_markers'");
        file->close();
        return false;
    }
```

```
    int number_of_vertices_boundary_markers = values[3].toInt(&ok);
    if(!ok)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_vertices_boundary_
markers"\
                 "' must be an integer");
        file->close();
        return false;
    }
    if(number_of_vertices_boundary_markers!=0 && number_of_vertices_boundary_mark
ers != 1)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_vertices_boundary_
markers"\
                 "' must be 0 or 1");
        file->close();
        return false;
    }
    if(number_of_vertices_boundary_markers > 0)
        qWarning("SemSolver::IO::readPSLG - ERROR : ignoring vertices boundary ma
rkers i"\
                 "\n 'poly_file'");
#endif
    pslg.setNumberOfVerticesBoundaryMarkers(0);
#ifdef SEMDEBUG
    if(values.size()>4)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : too many inputs on first line
");
        file->close();
        return false;
    }
#endif
    for(int i=0; i<number_of_vertices; ++i)
    {
        values = next_non_empty_line_values(input);
#ifdef SEMDEBUG
        if(values.isEmpty())
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Vertex::vertex_n
umber'");
            file->close();
            return false;
        }
#endif
        int vertex_number = values[0].toInt(&ok);
#ifdef SEMDEBUG
        if(!ok)
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : 'Vertex::vertex_number' m
ust be "\
                     "an integer");
            file->close();
            return false;
        }
        for(int j=0; j<i; ++j)
        {
            if(vertex_number==pslg.vertex(j).number)
            {
                qWarning("SemSolver::IO::readPSLG - ERROR : 'Vertex::vertex_numbe
r' must"\
                         "be unique");
```

```
                file->close();
                return false;
            }
        }
        if(values.size()<2)
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Vertex::x'");
            file->close();
            return false;
        }
#endif
        double x = values[1].toDouble(&ok);
#ifdef SEMDEBUG
        if(!ok)
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Vertex::x' must be a do"\
                     "uble");
            file->close();
            return false;
        }
        if(values.size()<3)
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Vertex::y'");
            file->close();
            return false;
        }
#endif
        double y = values[2].toDouble(&ok);
#ifdef SEMDEBUG
        if(!ok)
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : 'Vertex::y' must be a double");
            file->close();
            return false;
        }
        if(values.size()<3+number_of_vertices_attributes)
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Vertex::attribute'");
            file->close();
            return false;
        }
        if(values.size()<3+number_of_vertices_attributes
           +number_of_vertices_boundary_markers)
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Vertex::boundary_marker"\
                     "'");
            file->close();
            return false;
        }
#endif
        pslg.setVertex(i,vertex_number,x,y);
    }
    values = next_non_empty_line_values(input);
#ifdef SEMDEBUG
    if(values.isEmpty())
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : missing 'number_of_segments'");
```

```
            file->close();
            return false;
        }
#endif
    int number_of_segments = values[0].toInt(&ok);
#ifdef SEMDEBUG
    if(!ok)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_segments' must be an inte"\
                 "ger");
        file->close();
        return false;
    }
    if (number_of_segments < 0)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_segments' must be non-neg"\
                 "ative");
        file->close();
        return false;
    }
#endif
    pslg.setNumberOfSegments(number_of_segments);
#ifdef SEMDEBUG
    if(values.size()<2)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : missing 'number_of_segments_boundary"\
                 "_markers'");
        file->close();
        return false;
    }
    int number_of_segments_boundary_markers = values[1].toInt(&ok);
    if(!ok)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_segments_boundary_markers"\
                 "' must be an integer");
        file->close();
        return false;
    }
    if (number_of_segments_boundary_markers!=0 && number_of_segments_boundary_markers!=1)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_segments_boundary_markers"\
                 "' must be 0 or 1");
        file->close();
        return false;
    }
    if(number_of_segments_boundary_markers > 0)
        qWarning("SemSolver::IO::readPSLG - ERROR : ignoring segments boundary markers i"\
                 "n 'poly_file'");
    if(values.size()>2)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : 'too many inputs on segments header "\
                 "line");
        file->close();
        return false;
```

```
  }
#endif
  for(int i=0; i<number_of_segments; ++i)
  {
    values = next_non_empty_line_values(input);
#ifdef SEMDEBUG
    if(values.isEmpty())
    {
      qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Segment::segment
_number"\
           "'");
      file->close();
      return false;
    }
#endif
    int segment_number = values[0].toInt(&ok);
#ifdef SEMDEBUG
    if(!ok)
    {
      qWarning("SemSolver::IO::readPSLG - ERROR : 'Segment::segment_number'
 must b"\
           "e an integer");
      file->close();
      return false;
    }
    for(int j=0; j<i; ++j)
    {
      if(segment_number==pslg.segment(j).number)
      {
        qWarning("SemSolver::IO::readPSLG - ERROR : 'Segment::segment_num
ber' mu"\
             "st be unique");
        file->close();
        return false;
      }
    }
    if(values.size()<2)
    {
      qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Segment::source'
");
      file->close();
      return false;
    }
#endif
    int source = values[1].toInt(&ok);
#ifdef SEMDEBUG
    if(!ok)
    {
      qWarning("SemSolver::IO::readPSLG - ERROR  : 'Segment::source' must b
e an in"\
           "teger");
      file->close();
      return false;
    }
    if(values.size()<3)
    {
      qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Segment::target'
");
      file->close();
      return false;
    }
#endif
```

```
        int target = values[2].toInt(&ok);
#ifdef SEMDEBUG
        if(!ok)
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : 'Segment::target' must be an int"\
                     "eger");
            file->close();
            return false;
        }
        bool found_source = false;
        bool found_target = false;
        for(int j=0; j<number_of_vertices && !(found_source&&found_target); ++j)
        {
            if(pslg.vertex(j).number==source)
                found_source = true;
            if(pslg.vertex(j).number==target)
                found_target = true;
        }
        if(!found_source)
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : 'Segment::source' must be an exi"\
                     "sting 'vertex_number'");
            file->close();
            return false;
        }
        if(!found_target)
        {
            qWarning("SemSolver::IO::readPSLG - ERROR  : 'Segment::target' must be an ex"\
                     "isting 'vertex_number'");
            file->close();
            return false;
        }
        if(values.size()<3+number_of_segments_boundary_markers)
        {
            qWarning("SemSolver::IO::readPSLG - ERROR  : missing 'Segment::boundary_mark"\
                     "er'");
            file->close();
            return false;
        }
        if(values.size()>3+number_of_segments_boundary_markers)
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : too many inputs on segment line");
            file->close();
            return false;
        }
#endif
        pslg.setSegment(i, segment_number, source, target);
    }
    values = next_non_empty_line_values(input);
#ifdef SEMDEBUG
    if(values.isEmpty())
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : missing 'number_of_holes'");
        file->close();
        return false;
    }
#endif
```

```
    int number_of_holes = values[0].toInt(&ok);
#ifdef SEMDEBUG
    if(!ok)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : 'Segment::number_of_holes' mu
st be a"\
                 "n integer");
        file->close();
        return false;
    }
    if (number_of_holes < 0)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_holes' must be non
-negati"\
                 "ve");
        file->close();
        return false;
    }
    if(values.size()>1)
    {
        qWarning("SemSolver::IO::readPSLG - ERROR  : too many inputs on holes hea
der lin"\
                 "e");
        file->close();
        return false;
    }
#endif
    pslg.setNumberOfHoles(number_of_holes);
    for(int i=0; i<number_of_holes; ++i)
    {
        values = next_non_empty_line_values(input);
#ifdef SEMDEBUG
        if(values.isEmpty())
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Hole::hole_numbe
r'");
            file->close();
            return false;
        }
#endif
        int hole_number = values[0].toInt(&ok);
#ifdef SEMDEBUG
        if(!ok)
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : 'Hole::hole_number' must
be an i"\
                     "nteger");
            file->close();
            return false;
        }
        for(int j=0; j<i; ++j)
        {
            if(hole_number==pslg.hole(j).number)
            {
                qWarning("SemSolver::IO::readPSLG - ERROR : 'Hole::hole_number' m
ust be "\
                         "unique");
                file->close();
                return false;
            }
        }
        if(values.size()<2)
```

```
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Hole::x'");
            file->close();
            return false;
        }
#endif
        double x = values[1].toDouble(&ok);
#ifdef SEMDEBUG
        if(!ok)
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : 'Hole::x' must be a double");
            file->close();
            return false;
        }
        if(values.size()<3)
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Hole::y'");
            file->close();
            return false;
        }
#endif
        double y = values[2].toDouble(&ok);
#ifdef SEMDEBUG
        if(!ok)
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : 'Hole::y' must be a double");
            file->close();
            return false;
        }
        if(values.size()>3)
        {
            qWarning("SemSolver::IO::readPSLG - ERROR : 'too many inputs on hole line");
            file->close();
            return false;
        }
#endif
        pslg.setHole(i, hole_number, x, y);
    }
    values = next_non_empty_line_values(input);
#ifdef SEMDEBUG
    if(!values.isEmpty())
        qWarning("SemSolver::IO::readPSLG - ERROR : ignoring extra information in 'poly_"\
                 "file'");
#endif
    file->close();
    return true;
};
    };
};

#endif // IO_PSLG_HPP
```

7.81 pslg.hpp File Reference

```
#include <cmath>
```

Classes

- class SemSolver::PSLG< X >

  Class for handing Planar Straight Line Graphs.

- struct SemSolver::PSLG< X >::Vertex

  PSLG Vertex struct.

- struct SemSolver::PSLG< X >::Segment

  PSLG Segment struct.

- struct SemSolver::PSLG< X >::Hole

  PSLG Hole struct.

Namespaces

- namespace SemSolver

  Project main namespace.

7.82 pslg.hpp

```
#ifndef PSLG_HPP
#define PSLG_HPP

namespace SemSolver
{
    template<class X>
    class PSLG;
};

#include <cmath>

namespace SemSolver
{


    template<class X>
    class PSLG
    {
    public:


        struct Vertex
        {
            int number;
            X x;
            X y;
            double *attributes;
            int marker;
        };

```

```

        struct Segment
        {
            int number;
            int source;
            int target;
            int marker;
        };


        struct Hole
        {
            int number;
            X x;
            X y;
        };

    private:

        unsigned   vertices_number;
        unsigned   dimension;
        unsigned   vertices_attributes_number;
        unsigned   vertices_boundary_markers_number;
        Vertex *vertices_list;

        unsigned    segments_number;
        unsigned    segments_boundary_markers_number;
        Segment *segments_list;

        unsigned holes_number;
        Hole *holes_list;

    public:

        PSLG();

        ~PSLG();

        void clear();

        inline Vertex const &vertex(const unsigned &index) const;

        inline Segment const &segment(const unsigned &index) const;

        inline Hole const &hole(const unsigned &index) const;

        inline unsigned const &vertices() const;

        inline unsigned const &segments() const;

        inline unsigned const &holes() const;

        void setNumberOfVertices(const unsigned &number);

        void setNumberOfSegments(const unsigned &number);

        void setNumberOfHoles(const unsigned &number);

        void setNumberOfVerticesAttributes(const unsigned &number);

        inline void setNumberOfVerticesBoundaryMarkers(const unsigned &number);

```

```
        void setVertex(unsigned const &index,
                       int const &number,
                       const X &x,
                       const X &y,
                       double *attributes = 0,
                       int const &marker = 0);

        void setSegment(unsigned const &index,
                        int const &number,
                        int const &source,
                        int const &target,
                        int const &marker = 0);

        void setHole(unsigned const &index,
                     int const &number,
                     const X &x,
                     const X &y);
    };
};

template<class X>
SemSolver::PSLG<X>::PSLG()
{
    vertices_number = 0;
    dimension = 2;
    vertices_attributes_number = 0;
    vertices_boundary_markers_number = 0;
    vertices_list = 0;
    segments_number = 0;
    segments_boundary_markers_number = 0;
    segments_list = 0;
    holes_number = 0;
    holes_list = 0;
};

template<class X>
SemSolver::PSLG<X>::~PSLG()
{
    for(unsigned i=0; i<vertices_number; ++i)
    {
        delete [] vertices_list[i].attributes;
        vertices_list[i].attributes = 0;
    }
    //delete [] vertices_list;
    //delete [] segments_list;
    //delete [] holes_list;
};

template<class X>
void SemSolver::PSLG<X>::clear()
{
    for(unsigned i=0; i<vertices_number; ++i)
    {
        delete [] vertices_list[i].attributes;
        vertices_list[i].attributes = 0;
    }
    delete [] vertices_list;
    delete [] segments_list;
    delete [] holes_list;
    vertices_number = 0;
    vertices_attributes_number = 0;
    vertices_boundary_markers_number = 0;
```

```
    vertices_list = 0;
    segments_number = 0;
    segments_boundary_markers_number = 0;
    segments_list = 0;
    holes_number = 0;
    holes_list = 0;
};

template<class X>
inline const typename SemSolver::PSLG<X>::Vertex
        &SemSolver::PSLG<X>::vertex(const unsigned &index) const
{
#ifdef SEMDEBUG
    if(index>=vertices_number)
        qFatal("SemSolver::PSLG::vertex - ERROR : index out of bounds.");
#endif //SEMDEBUG
    return vertices_list[index];
};

template<class X>
inline const typename SemSolver::PSLG<X>::Segment
        &SemSolver::PSLG<X>::segment(const unsigned &index) const
{
#ifdef SEMDEBUG
    if(index>=segments_number)
        qFatal("SemSolver::PSLG::segment - ERROR : index out of bounds.");
#endif //SEMDEBUG
    return segments_list[index];
};

template<class X>
inline const typename SemSolver::PSLG<X>::Hole
        &SemSolver::PSLG<X>::hole(const unsigned &index) const
{
#ifdef SEMDEBUG
    if(index>=holes_number)
        qFatal("SemSolver::PSLG::hole - ERROR : index out of bounds.");
#endif //SEMDEBUG
    return holes_list[index];
};

template<class X>
inline const unsigned &SemSolver::PSLG<X>::vertices() const
{
    return vertices_number;
};

template<class X>
inline const unsigned &SemSolver::PSLG<X>::segments() const
{
    return segments_number;
};

template<class X>
inline const unsigned &SemSolver::PSLG<X>::holes() const
{
    return holes_number;
};

template<class X>
void SemSolver::PSLG<X>::setNumberOfVertices(const unsigned &number)
{
```

```
    for(unsigned i=0; i<vertices_number; ++i)
    {
        delete [] vertices_list[i].attributes;
        vertices_list[i].attributes = 0;
    }
    delete [] vertices_list;
    vertices_number = number;
    if(number)
    {
        vertices_list = new Vertex[number];
        if(vertices_attributes_number)
            for(unsigned i=0; i<number; ++i)
                vertices_list[i].attributes = new double[vertices_attributes_numb
er];
        else
            for(unsigned i=0; i<number; ++i)
                vertices_list[i].attributes = 0;
    }
    else
        vertices_list = 0;
};

template<class X>
void SemSolver::PSLG<X>::setNumberOfSegments(const unsigned &number)
{
    delete [] segments_list;
    segments_number = number;
    if(number)
        segments_list = new Segment[number];
    else
        segments_list = 0;
};

template<class X>
void SemSolver::PSLG<X>::setNumberOfHoles(const unsigned &number)
{
    delete [] holes_list;
    holes_number = number;
    if(number)
        holes_list = new Hole[number];
    else
        holes_list = 0;
};

template<class X>
void SemSolver::PSLG<X>::setNumberOfVerticesAttributes(const unsigned &number)
{
    vertices_attributes_number = number;
    if(number)
    {
        for(unsigned i=0; i<vertices_number; ++i)
        {
            delete [] vertices_list[i].attributes;
            vertices_list[i].attributes = new double[number];
        }
    }
    else
    {
        for(unsigned i=0; i<vertices_number; ++i)
        {
            delete [] vertices_list[i].attributes;
            vertices_list[i].attributes = 0;
```

```
        }
    }
};

template<class X>
void SemSolver::PSLG<X>::setNumberOfVerticesBoundaryMarkers(const unsigned &number)
{
#ifdef SEMDEBUG
    if(number!=0 && number !=1)
        qFatal("SemSolver::PSLG::setNumberOfVerticesBoundaryMarkers - ERROR : number mus"\
               "t be 0 or 1.");
#endif //SEMDEBUG
    vertices_boundary_markers_number = number;
};

template<class X>
void SemSolver::PSLG<X>::setVertex(const unsigned &index,
                                   int const &number,
                                   const X &x,
                                   const X &y,
                                   double *attributes,
                                   int const &marker)
{
#ifdef SEMDEBUG
    if(index>=vertices_number)
        qFatal("SemSolver::PSLG::setVertex - ERROR : index out of bounds.");
#endif //SEMDEBUG
    vertices_list[index].number = number;
    vertices_list[index].x = x;
    vertices_list[index].y = y;
    delete [] vertices_list[index].attributes;
    if(vertices_attributes_number)
    {
        vertices_list[index].attributes = new double[vertices_attributes_number];
        for (unsigned i=0; i<vertices_attributes_number; ++i)
            vertices_list[index].attributes[i] = attributes[i];
    }
    else
        vertices_list[index].attributes = 0;
    vertices_list[index].marker = marker;
};

template<class X>
void SemSolver::PSLG<X>::setSegment(const unsigned &index,
                                    int const &number,
                                    int const &source,
                                    int const &target,
                                    int const &marker)
{
#ifdef SEMDEBUG
    if(index>=segments_number)
        qFatal("SemSolver::PSLG::setSegment- ERROR : index out of bounds.");
#endif //SEMDEBUG
    segments_list[index].number = number;
    segments_list[index].source = source;
    segments_list[index].target = target;
    segments_list[index].marker = marker;
};

```

```
template<class X>
void SemSolver::PSLG<X>::setHole(const unsigned &index,
                                 int const &number,
                                 const X &x,
                                 const X &y)
{
#ifdef SEMDEBUG
    if(index>=holes_number)
        qFatal("SemSolver::PSLG::setHole - ERROR : index out of bounds.");
#endif //SEMDEBUG
    holes_list[index].number = number;
    holes_list[index].x = x;
    holes_list[index].y = y;
};

#endif // PSLG_HPP

```

7.83 qrsolve.hpp File Reference

`#include <jama_qr.h>`

`#include <SemSolver/matrix.hpp>`

`#include <SemSolver/vector.hpp>`

Namespaces

- namespace SemSolver

  Project main namespace.

- namespace SemSolver::Solver

  Solver namespace.

Functions

- template< class X >
  bool SemSolver::Solver::qr_solve (Matrix< X > const &A, Vector< X > const &b, Vector< X > &x)

7.84 qrsolve.hpp

```
#ifndef QRSOLVE_HPP
#define QRSOLVE_HPP

#if defined _WIN32 || defined _WIN64
#       include <SemSolver/math_defines>
#endif

#include <jama_qr.h>
```

```

#include <SemSolver/matrix.hpp>
#include <SemSolver/vector.hpp>

namespace SemSolver
{

    namespace Solver
    {
        template<class X>
        bool qr_solve(Matrix<X> const &A,
                      Vector<X> const &b,
                      Vector<X> &x)
        {
            JAMA::QR<double> qr(A);
#ifdef SEMDEBUG
            if(!qr.isFullRank())
            {
                qWarning("SemSolver::Solver::qr_solve - ERROR : Matrix A is not f
ull ran"\
                         "k.");
                return false;
            }
#endif
            x = qr.solve(b);
            return true;
        };
    };
};

#endif // QRSOLVE_HPP
```

7.85 scriptfunction.hpp File Reference

`#include <cmath>`

`#include <SemSolver/point.hpp>`

`#include <SemSolver/function.hpp>`

`#include <SemSolver/vector.hpp>`

`#include <QString>`

`#include <QStringList>`

`#include <QScriptEngine>`

Classes

- class SemSolver::ScriptFunction< Point< d, X >, Y >

  Class for handling function from Euclidean space X^d to scalar space Y defined in scripts.

- class SemSolver::ScriptFunction< Point< d, X >, Vector< Y > >

  Class for handling function from Euclidean space X^d to Vectorial space Y^n defined inscripts.

- class SemSolver::ScriptFunction< Point< 2, X >, Y >

  Class for handling function from 2D Euclidean space X^2 to scalar space Y defined in scripts.

- class SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > >

  Class for handling function from 2D Euclidean space X^2 to Vectorial space Y^n defined in scripts.

Namespaces

- namespace SemSolver

  Project main namespace.

7.86 scriptfunction.hpp

```
#ifndef SCRIPTFUNCTION_HPP
#define SCRIPTFUNCTION_HPP

namespace SemSolver
{
    template<class X, class Y>
    class ScriptFunction;
};

#include <cmath>

#include <SemSolver/point.hpp>
#include <SemSolver/function.hpp>
#include <SemSolver/vector.hpp>

#include <QString>
#include <QStringList>
#include <QScriptEngine>

namespace SemSolver
{
    template<int d, class X, class Y>
    class ScriptFunction< Point<d, X>, Y >
        : public Function< Point<d, X>, Y >
    {
        QString script;
        QString program;
        QScriptEngine *engine;
    public:

        ScriptFunction(QString const &string);

        ~ScriptFunction();

        Y evaluate(const Point<d, X> &point) const;

        QString mml() const;
```

```
    };

    template<int d, class X, class Y>
    class ScriptFunction< Point<d, X>, Vector<Y> >
        : public Function< Point<d, X>, Vector<Y> >
    {
        QStringList scripts;
        QString program;
        QScriptEngine *engine;
    public:

        ScriptFunction(QStringList const &strings);

        ~ScriptFunction();

        Vector<Y> evaluate(const Point<d, X> &point) const;

        QString mml() const;
    };

    template<class X, class Y>
    class ScriptFunction< Point<2, X>, Y >
        : public Function< Point<2, X>, Y >
    {
        QString script;
        QString program;
        QScriptEngine *engine;
    public:

        ScriptFunction(QString const &string);

        ~ScriptFunction();

        Y evaluate(const Point<2, X> &point) const;

        QString mml() const;
    };

    template<class X, class Y>
    class ScriptFunction< Point<2, X>, Vector<Y> >
        : public Function< Point<2, X>, Vector<Y> >
    {
        QStringList scripts;
        QString program;
        QScriptEngine *engine;
    public:

        ScriptFunction(QStringList const &strings);

        ~ScriptFunction();

        Vector<Y> evaluate(const Point<2, X> &point) const;

        QString mml() const;
    };
};

template<int d, class X, class Y>
SemSolver::ScriptFunction< SemSolver::Point<d, X>, Y >::ScriptFunction(
        const QString &string)
{
    script = string;
```

```
    engine = new QScriptEngine;
    program = "function f(";
    for(int i=0; i<d-1; ++i)
        program += "x" + QString::number(i) + ", ";
    program += "x" + QString::number(d) + ") { return " + script + "; }";
    engine->evaluate("function abs   (x)   { return Math.abs(x);      }"\
                     "function acos  (x)   { return Math.acos(x);     }"\
                     "function asin  (x)   { return Math.asin(x);     }"\
                     "function atan  (x)   { return Math.atan(x);     }"\
                     "function atan2 (x,y) { return Math.atan2(x,y); }"\
                     "function ceil  (x)   { return Math.ceil(x);     }"\
                     "function cos   (x)   { return Math.cos(x);      }"\
                     "function exp   (x)   { return Math.exp(x);      }"\
                     "function floor (x)   { return Math.floor(x);    }"\
                     "function log   (x)   { return Math.log(x);      }"\
                     "function max   (x,y) { return Math.max(x,y);    }"\
                     "function min   (x,y) { return Math.min(x,y);    }"\
                     "function pow   (x,y) { return Math.pow(x,y);    }"\
                     "function random()    { return Math.random();    }"\
                     "function round (x)   { return Math.round(x);    }"\
                     "function sin   (x)   { return Math.sin(x);      }"\
                     "function sqrt  (x)   { return Math.sqrt(x);     }"\
                     "function tan   (x)   { return Math.tan(x);      }"\
                     "var pi = Math.PI;                                 "\
                     "var e  = Math.E ;                                 ");
    engine->evaluate(program);
};

template<int d, class X, class Y>
SemSolver::ScriptFunction< SemSolver::Point<d, X>, Y >::~ScriptFunction()
{
    delete engine;
};

template<int d, class X, class Y>
Y SemSolver::ScriptFunction< SemSolver::Point<d, X>, Y >::evaluate(
        const SemSolver::Point<d, X> &point) const
{
    QString call = "f(";
    for(int i=0; i<d-1; ++i)
        call += QString::number(point.cartesian(i)) + ",";
    call += QString::number(point.cartesian(d-1)) + ")";
    QScriptValue value = engine->evaluate(call);
    return value.toNumber();
};

template<int d, class X, class Y>
QString SemSolver::ScriptFunction< SemSolver::Point<d, X>, Y >::mml() const
{
    return "<mtext> " + script + "</mtext>";
};

template<int d, class X, class Y>
SemSolver::ScriptFunction< SemSolver::Point<d, X>, SemSolver::Vector<Y> >::ScriptFunction(
        const QStringList &strings)
{
    scripts = strings;
    engine = new QScriptEngine;
    for(int j=0; j<scripts.size(); ++j)
    {
        program += "function f" + QString::number(j) +"(";
```

```
        for(int i=0; i<d-1; ++i)
            program += "x" + QString::number(i) + ", ";
        program += "x" + QString::number(d) + ") { return " + scripts[j] + "; }\n";
    }

    engine->evaluate("function abs    (x)   { return Math.abs(x);     }"\
                     "function acos   (x)   { return Math.acos(x);    }"\
                     "function asin   (x)   { return Math.asin(x);    }"\
                     "function atan   (x)   { return Math.atan(x);    }"\
                     "function atan2  (x,y) { return Math.atan2(x,y); }"\
                     "function ceil   (x)   { return Math.ceil(x);    }"\
                     "function cos    (x)   { return Math.cos(x);     }"\
                     "function exp    (x)   { return Math.exp(x);     }"\
                     "function floor  (x)   { return Math.floor(x);   }"\
                     "function log    (x)   { return Math.log(x);     }"\
                     "function max    (x,y) { return Math.max(x,y);   }"\
                     "function min    (x,y) { return Math.min(x,y);   }"\
                     "function pow    (x,y) { return Math.pow(x,y);   }"\
                     "function random()     { return Math.random();   }"\
                     "function round  (x)   { return Math.round(x);   }"\
                     "function sin    (x)   { return Math.sin(x);     }"\
                     "function sqrt   (x)   { return Math.sqrt(x);    }"\
                     "function tan    (x)   { return Math.tan(x);     }"\
                     "var pi = Math.PI;                                "\
                     "var e  = Math.E ;                                ");
    engine->evaluate(program);
};

template<int d, class X, class Y>
SemSolver::ScriptFunction<SemSolver::Point<d,X>, SemSolver::Vector<Y> >::~ScriptFunction()
{
    delete engine;
};

template<int d, class X, class Y>
SemSolver::Vector<Y> SemSolver::ScriptFunction<
        SemSolver::Point<d, X>, SemSolver::Vector<Y> >::evaluate(
                const SemSolver::Point<d, X> &point) const
{
    int l = scripts.size();
    Vector<Y> result(l);
    for(int j=0; j<l; ++j)
    {
        QString call = "f" + QString::number(j) + "(";
        for(int i=0; i<d-1; ++i)
            call += QString::number(point.cartesian(i)) + ",";
        call += QString::number(point.cartesian(d-1)) + ")";
        QScriptValue value = engine->evaluate(call);
        result[j] = value.toNumber();
    };
    return result;
};

template<int d, class X, class Y>
QString SemSolver::ScriptFunction<SemSolver::Point<d, X>, SemSolver::Vector<Y> >::mml(
        ) const
{
    QString mml = "<mfenced open='(' close=')'><mtable>";
    for(int i=0; i<scripts.size(); ++i)
```

```
            mml += "<mtr><mtd><mtext>" + scripts[i] + "</mtext></mtd></mtr>";
        mml += "</mtable></mfenced>";
        return mml;
};

template<class X, class Y>
SemSolver::ScriptFunction< SemSolver::Point<2, X>, Y >::ScriptFunction(
            const QString &string)
{
        script = string;
        engine = new QScriptEngine;
        program = "function f(x,y) { return " + script + "; }";
        engine->evaluate("function abs   (x)   { return Math.abs(x);     }"\
                         "function acos  (x)   { return Math.acos(x);    }"\
                         "function asin  (x)   { return Math.asin(x);    }"\
                         "function atan  (x)   { return Math.atan(x);    }"\
                         "function atan2 (x,y) { return Math.atan2(x,y); }"\
                         "function ceil  (x)   { return Math.ceil(x);    }"\
                         "function cos   (x)   { return Math.cos(x);     }"\
                         "function exp   (x)   { return Math.exp(x);     }"\
                         "function floor (x)   { return Math.floor(x);   }"\
                         "function log   (x)   { return Math.log(x);     }"\
                         "function max   (x,y) { return Math.max(x,y);   }"\
                         "function min   (x,y) { return Math.min(x,y);   }"\
                         "function pow   (x,y) { return Math.pow(x,y);   }"\
                         "function random()    { return Math.random();   }"\
                         "function round (x)   { return Math.round(x);   }"\
                         "function sin   (x)   { return Math.sin(x);     }"\
                         "function sqrt  (x)   { return Math.sqrt(x);    }"\
                         "function tan   (x)   { return Math.tan(x);     }"\
                         "var pi = Math.PI;                               "\
                         "var e  = Math.E ;                               ");
        engine->evaluate(program);
};

template<class X, class Y>
SemSolver::ScriptFunction< SemSolver::Point<2, X>, Y >::~ScriptFunction()
{
        delete engine;
};

template<class X, class Y>
Y SemSolver::ScriptFunction< SemSolver::Point<2, X>, Y >::evaluate(
            const SemSolver::Point<2, X> &point) const
{
        QString call = "f(" + QString::number(point.x()) + "," + QString::number(point.y()) \
                            + ")";
        QScriptValue value = engine->evaluate(call);
        return value.toNumber();
};

template<class X, class Y>
QString SemSolver::ScriptFunction< SemSolver::Point<2, X>, Y >::mml() const
{
        return "<mtext> " + script + "</mtext>";
};

template<class X, class Y>
SemSolver::ScriptFunction< SemSolver::Point<2, X>, SemSolver::Vector<Y> >::ScriptFunction(
            const QStringList &strings)
```

```
{
    scripts = strings;
    engine = new QScriptEngine;
    for(int j=0; j<scripts.size(); ++j)
        program += "function f" + QString::number(j) +"(x,y) { return " + scripts[j] + \
                   "; }\n";

    engine->evaluate("function abs   (x)   { return Math.abs(x);     }"\
                     "function acos  (x)   { return Math.acos(x);    }"\
                     "function asin  (x)   { return Math.asin(x);    }"\
                     "function atan  (x)   { return Math.atan(x);    }"\
                     "function atan2 (x,y) { return Math.atan2(x,y); }"\
                     "function ceil  (x)   { return Math.ceil(x);    }"\
                     "function cos   (x)   { return Math.cos(x);     }"\
                     "function exp   (x)   { return Math.exp(x);     }"\
                     "function floor (x)   { return Math.floor(x);   }"\
                     "function log   (x)   { return Math.log(x);     }"\
                     "function max   (x,y) { return Math.max(x,y);   }"\
                     "function min   (x,y) { return Math.min(x,y);   }"\
                     "function pow   (x,y) { return Math.pow(x,y);   }"\
                     "function random()    { return Math.random();   }"\
                     "function round (x)   { return Math.round(x);   }"\
                     "function sin   (x)   { return Math.sin(x);     }"\
                     "function sqrt  (x)   { return Math.sqrt(x);    }"\
                     "function tan   (x)   { return Math.tan(x);     }"\
                     "var pi = Math.PI;                               "\
                     "var e  = Math.E ;                               ");
    engine->evaluate(program);
};

template<class X, class Y>
SemSolver::ScriptFunction<SemSolver::Point<2,X>, SemSolver::Vector<Y> >::~ScriptFunction()
{
    delete engine;
};

template<class X, class Y>
SemSolver::Vector<Y> SemSolver::ScriptFunction<
        SemSolver::Point<2, X>, SemSolver::Vector<Y> >::evaluate(
                const SemSolver::Point<2, X> &point) const
{
    int l = scripts.size();
    Vector<Y> result(l);
    for(int j=0; j<l; ++j)
    {
        QString call = "f" + QString::number(j) + "(" + QString::number(point.x()) + ","
                           + QString::number(point.y()) + ")";
        QScriptValue value = engine->evaluate(call);
        result[j] = value.toNumber();
    };
    return result;
};

template<class X, class Y>
QString SemSolver::ScriptFunction<SemSolver::Point<2, X>, SemSolver::Vector<Y> >::mml(
        ) const
{
    QString mml = "<mfenced open='(' close=')'><mtable>";
```

```
    for(int i=0; i<scripts.size(); ++i)
        mml += "<mtr><mtd><mtext>" + scripts[i] + "</mtext></mtd></mtr>";
    mml += "</mtable></mfenced>";
    return mml;
};

#endif // SCRIPTFUNCTION_HPP
```

7.87 segment.hpp File Reference

```
#include <CGAL/Cartesian.h>
#include <CGAL/Object.h>
#include <CGAL/Segment_2.h>
#include <CGAL/intersections.h>
#include <SemSolver/point.hpp>
```

Classes

- class SemSolver::Segment< 2, X >
- struct SemSolver::Segment< 2, X >::less

Namespaces

- namespace SemSolver

  Project main namespace.

7.88 segment.hpp

```
#ifndef SEGMENT_HPP
#define SEGMENT_HPP

namespace SemSolver
{
    template<int d, class X>
    class Segment;
};

#include <CGAL/Cartesian.h>
#include <CGAL/Object.h>
#include <CGAL/Segment_2.h>
#include <CGAL/intersections.h>

#include <SemSolver/point.hpp>

namespace SemSolver
{
    template<class X>
```

```
    class Segment<2,X>
        : public CGAL::Segment_2< CGAL::Filtered_kernel< CGAL::Simple_cartesian<X> > > >
    {
        typedef CGAL::Point_2< CGAL::Filtered_kernel< CGAL::Simple_cartesian<X> > >
                CGAL_point;
        typedef CGAL::Segment_2< CGAL::Filtered_kernel< CGAL::Simple_cartesian<X> > >
                CGAL_segment;

    public:

        Segment(CGAL_segment const &s) : CGAL_segment(s) {};

        Segment(Point<2,X> const &source, Point<2,X> const &target)
            : CGAL_segment(source, target) {};

        CGAL_segment const &cgal() const
        {
            return *this;
        };

        bool intersect(Segment<2,X> const &segment) const
        {
            return CGAL::do_intersect(segment, *this);
        };

        bool intersectInteriorly(Segment<2,X> const &segment) const
        {
            if(!intersect(segment))
                return false;
            CGAL_point intersection;
            CGAL::Object result = CGAL::intersection(segment.cgal(), this->cgal());
            if(!CGAL::assign(intersection, result))
                return true;
            if(intersection!=segment.source() && intersection!=segment.target() &&
               intersection!=this->source() && intersection!=this->target())
                return true;
            return false;
        };

        struct less
        {
            bool operator() (Segment const &s0, Segment const &s1) const
            {
                if(s0.source().x()<s1.source().x())
                    return true;
                if(s0.source().x()>s1.source().x())
                    return false;
                if(s0.source().y()<s1.source().y())
                    return true;
                if(s0.source().y()>s1.source().y())
                    return false;
                if(s0.target().x()<s1.target().x())
                    return true;
                if(s0.target().x()>s1.target().x())
                    return false;
                if(s0.target().y()<s1.target().y())
                    return true;
```

```
                    //if(s0.target().y()>s1.target().y())
                    return false;
                }
            };
        };
};

#endif // SEGMENT_HPP
```

7.89 segmentsmap.hpp File Reference

```
#include <map>
#include <SemSolver/segment.hpp>
#include <SemSolver/point.hpp>
```

Classes

- class SemSolver::SegmentsMap< d, X >

Namespaces

- namespace SemSolver

  Project main namespace.

7.90 segmentsmap.hpp

```
#ifndef SEGMENTSVECTOR_HPP
#define SEGMENTSVECTOR_HPP

namespace SemSolver
{
    template<int d, class X>
    class SegmentsMap;
};

#include <map>

#include <SemSolver/segment.hpp>
#include <SemSolver/point.hpp>


namespace SemSolver
{
    template<int d, class X>
    class SegmentsMap
        : std::map< int, Segment<d,X> >
    {
    private:
        typedef std::map< int, Segment<d,X> > Base;
```

```
    protected:
        typedef typename Base::iterator Iterator;

    public:
        typedef typename Base::const_iterator ConstIterator;
        using Base::begin;
        using Base::end;

    private:
        int _current_id;

    public:

        SegmentsMap()
        {
            _current_id = 0;
        }

        bool contains(Segment<d,X> const &segment) const
        {
            for(ConstIterator it=Base::begin(); it!=Base::end(); ++it)
                if(it->second==segment)
                    return true;
            return false;
        }

        bool has(int const &segment_id) const
        {
            for(ConstIterator it=begin(); it!=end(); ++it)
                if(it->first==segment_id)
                    return true;
            return false;
        }

        Segment<d,X> segmentFrom(Point<d,X> const &source) const
        {
            for(ConstIterator it=begin(); it!=end(); ++it)
                if(it->second.source()==source)
                    return it->second;
            return Segment<d,X>(source, source);
        }

        Segment<d,X> segmentTo(Point<d,X> const &target) const
        {
            for(ConstIterator it=begin(); it!=end(); ++it)
                if(it->second.target()==target)
                    return it->second;
            return Segment<d,X>(target, target);
        }

        Segment<d,X> const &segment(int const &id) const
        {
            ConstIterator it = Base::find(id);
#ifdef SEMDEBUG
            if(it==Base::end())
                qFatal("SemSolver::SegmentsMap::modify - ERROR : id out of range");
#endif
            return it->second;
        };

```

```
        int id(Segment<d,X> const &segment) const
        {
            for(ConstIterator it=begin(); it!=end(); ++it)
                if(it->second==segment)
                    return it->first;
            qFatal("SemSolver::SegmentsMap::id - ERROR : segment must exist");
            return 0;
        };

        int add(Segment<d,X> const &segment)
        {
            int id = _new_id();
            insert(id,segment);
            return id;
        }

        void insert(int const &id, Segment<d,X> const &segment)
        {
            Base::insert(typename Base::value_type(id,segment));
        }

        void remove(int const &id)
        {
            Base::erase(id);
        }

        inline void modify(int const &id, Segment<d,X> const &segment)
        {
            Iterator it = Base::find(id);
#ifdef SEMDEBUG
            if(it==Base::end())
                qFatal("SemSolver::SegmentsMap::modify - ERROR : id out of range");
#endif
            it->second = segment;
        }

        int segments() const
        {
            return Base::size();
        };

        virtual bool intersect(Segment<d,X> const &segment) const
        {
            for(ConstIterator it=begin(); it!=end(); ++it)
                if(segment.intersect(it->second))
                    return true;
            return false;
        }

        virtual bool intersectInteriorly(Segment<d,X> const &segment) const
        {
            for(ConstIterator it=begin(); it!=end(); ++it)
                if(segment.intersectInteriorly(it->second))
                    return true;
            return false;
        }

        bool isConsistentWith(Segment<d,X> const &segment) const
        {
            for (ConstIterator it=begin(); it!=end(); ++it)
                if(it->second.source()==segment.source() ||
```

```
                        it->second.target()==segment.target() )
                         return false;
                 return true;
             }

             bool haveOn(Point<d,X> const &point) const
             {
                 for (ConstIterator it=begin(); it!=end(); ++it)
                     if(it->second.has_on(point))
                         return true;
                 return false;
             }


         private:
             int const &_new_id()
             {
                 return ++_current_id;
             };

         };
 };

 #endif // SEGMENTSVECTOR_HPP
```

7.91 semfunction.hpp File Reference

```
#include <SemSolver/function.hpp>
#include <SemSolver/polynomialfunction.hpp>
#include <SemSolver/bileartransformation.hpp>
#include <SemSolver/point.hpp>
#include <SemSolver/semgeometry.hpp>
#include <SemSolver/matrix.hpp>
#include <SemSolver/vector.hpp>
#include <SemSolver/semparameters.hpp>
```

Classes

- class SemSolver::SemFunction< d, X >

  Class for handling spectral elements functions.

- class SemSolver::SemFunction< 2, X >

  Class for handling 2D spectral elements functions.

Namespaces

- namespace SemSolver

Project main namespace.

7.92 semfunction.hpp

```
#ifndef SEMFUNCTION_HPP
#define SEMFUNCTION_HPP

#include <SemSolver/function.hpp>
#include <SemSolver/polynomialfunction.hpp>
#include <SemSolver/bilineartransformation.hpp>
#include <SemSolver/point.hpp>
#include <SemSolver/semgeometry.hpp>
#include <SemSolver/matrix.hpp>
#include <SemSolver/vector.hpp>
#include <SemSolver/semparameters.hpp>

namespace SemSolver
{

    template <int d,class X>
            class SemFunction
                : public Function< Point<d,X>, X >
    {};


    template<class X>
    class SemFunction<2,X>
        : public Function< Point<2,X>, X >
    {
        typedef std::vector< PolynomialFunction<2,X> > PolynomialFunctionVector;
        typedef std::vector< BilinearTransformation<X> > BilinearTransformationsVector;

        Polygonation<2,X> _polygonation;
        PolynomialFunctionVector _polynomials;
        BilinearTransformationsVector _maps;

    public:
        //***** tranformation to be generalized *****//
                SemFunction(
                        SemGeometry<2,X> const &geometry,
                        PolynomialFunctionVector const &polynomials,
                        BilinearTransformationsVector const &maps)
                            : _polygonation(geometry.subDomains()),
                            _polynomials(polynomials),
                            _maps(maps)
        {};

        SemFunction(SemFunction const &sem_function)
            : _polygonation(sem_function._polygonation),
            _polynomials(sem_function._polynomials),
            _maps(sem_function._maps)
        {};

        inline PolynomialFunction<2,X> const &polynomial(const unsigned &index) const
        {
#if SEMDEBUG
```

```
            if(index>=_polygonation.size())
                qFatal("SemSolver::SemFunction::polynomial error : index out of range");
#endif
            return _polynomials.at(index);
        };

        inline void setPolynomialComponent(const unsigned &index,
                                           const unsigned &component,
                                           const Polynomial<X> &poly)
        {
#if SEMDEBUG
            if(index>=_polygonation.size())
                qFatal("SemSolver::SemFunction::polynomial error : index out of range");
#endif
            _polynomials.at(index).setPolynomial(component, poly);
        };

        inline BilinearTransformation<X> const &map(const unsigned &index) const
        {
#if SEMDEBUG
            if(index>=_polygonation.size())
                qFatal("SemSolver::SemFunction::polynomial error : index out of range");
#endif
            return _maps.at(index);
        };

        double evaluate(Point<2,X> const &x) const
        {
            std::vector<unsigned> element_index = _polygonation.elementIndicesAt(x);
            if(element_index.size()==0)
                return 0.;
            Point<2,X> x_hat = map(element_index[0]).evaluateInverse(x);
            return polynomial(element_index[0]).evaluate(x_hat);
        };

        Vector<X> evaluateRestrictionGradient(
                int const &element_index,
                Point<2,X> const &P) const
        {
            Vector<double> gradient(2);
            Point<2,X> const &P_hat = map(element_index).evaluateInverse(P);
            Matrix<X> tIJ_phi = map(element_index).evaluateTransposeInverseJacobian(P_hat);
            X const &x_hat = P_hat.x();
            X const &y_hat = P_hat.y();
            Polynomial<X> const &px = polynomial(element_index).polynomial(0);
            Polynomial<X> const &py = polynomial(element_index).polynomial(1);
            X psi_x = px.derivative()(x_hat) * py(y_hat);
            X psi_y = px(x_hat) * py.derivative()(y_hat);
            gradient[0] = psi_x * tIJ_phi[0][0] + psi_y * tIJ_phi[0][1];
            gradient[1] = psi_x * tIJ_phi[1][0] + psi_y * tIJ_phi[1][1];
            return gradient;
        };
    };
};

#endif // SEMFUNCTION_HPP
```

7.93 semgeometry.hpp File Reference

```
#include <SemSolver/pslg.hpp>
#include <SemSolver/polygonation.hpp>
```

Classes

- class SemSolver::SemGeometry< d, X >

  Class for describing the geometry of a SemProblem.

- class SemSolver::SemGeometry< 2, X >

Namespaces

- namespace SemSolver

  Project main namespace.

7.94 semgeometry.hpp

```
#ifndef SEMGEOMETRY_HPP
#define SEMGEOMETRY_HPP

namespace SemSolver
{



    template<int d, class X>
    class SemGeometry
    {
    };
};

#include <SemSolver/pslg.hpp>
#include <SemSolver/polygonation.hpp>

namespace SemSolver
{

    template<class X>
    class SemGeometry<2,X>
    {

        PSLG<X> pslg;
        Polygonation<2,X> sub_domains;

    public:

        inline PSLG<X> const &domain() const
                         {
    return pslg;
```

```
};

        inline Polygonation<2,X> const &subDomains() const
{
    return sub_domains;
};

        inline void setDomain(PSLG<X> const &domain)
                {
    pslg = domain;
};

        inline void setSubDomains(Polygonation<2,X> const &sub_domains)
                        {
    this->sub_domains = sub_domains;
};

        inline bool contains(const Point<2, X> &point) const
{
    return !sub_domains.elementIndicesAt(point).empty();
};
    };
};

#endif // SEMGEOMETRY_HPP
```

7.95 semparameters.hpp File Reference

Classes

- class SemSolver::SemParameters< X >

Namespaces

- namespace SemSolver

  Project main namespace.

7.96 semparameters.hpp

```
#ifndef SEMPARAMETERS_HPP
#define SEMPARAMETERS_HPP

namespace SemSolver
{
    template<class X>
    class SemParameters;
};

namespace SemSolver
{
    template <class X>
    class SemParameters
```

```
    {
        int _degree;
        X _tolerance;
        X _penality;

    public:
        SemParameters()
        {};

        SemParameters(int const &degree,
                      X const &tolerance,
                      X const &penality)
                          : _degree(degree),
                          _tolerance(tolerance),
                          _penality(penality)
        {};

        inline int const &degree() const
        {
            return _degree;
        };

        inline X const &tolerance() const
        {
            return _tolerance;
        };

        inline X const &penality() const
        {
            return _penality;
        };

        inline void setDegree(const int &d)
        {
            _degree = d;
        };

        inline void setTolerance(const X &t)
        {
            _tolerance = t;
        };

        inline void setPenality(const X &p)
        {
            _penality = p;
        };
    };
};

#endif // SEMPARAMETERS_HPP
```

7.97 semspace.hpp File Reference

```
#include <vector>
#include <cmath>
#include <SemSolver/hilbertspace.hpp>
#include <SemSolver/semfunction.hpp>
```

```
#include <SemSolver/multiindex.hpp>
#include <SemSolver/semgeometry.hpp>
#include <SemSolver/semparameters.hpp>
#include <SemSolver/pointsmap.hpp>
```

Classes

- class SemSolver::SemSpace< d, X >
- class SemSolver::SemSpace< 2, X >
- class SemSolver::SemSpace< 2, X >::Node
- class SemSolver::SemSpace< 2, X >::Element

    Class for handling members of the space as Fourier coefficients.

Namespaces

- namespace SemSolver

    Project main namespace.

7.98 semspace.hpp

```
#ifndef SEMSPACE_HPP
#define SEMSPACE_HPP

namespace SemSolver
{
    template <int d, class X>
            class SemSpace;
}

#include <vector>
#include <cmath>

#include <SemSolver/hilbertspace.hpp>
#include <SemSolver/semfunction.hpp>
#include <SemSolver/multiindex.hpp>
#include <SemSolver/semgeometry.hpp>
#include <SemSolver/semparameters.hpp>
#include <SemSolver/pointsmap.hpp>

namespace SemSolver
{


    template <int d, class X>
            class SemSpace
                : public HilbertSpace< SemFunction<d,X>, X >
    {};

    template<class X>
```

```
class SemSpace<2,X>
    : public HilbertSpace< SemFunction<2,X>, X >
{
public:
    class Node
    {
        typedef std::vector< MultiIndex<2> > Indices2Vector;
        typedef std::vector< MultiIndex<3> > Indices3Vector;

        Point<2,X> _point;
        Indices3Vector _element_indices;
        Indices2Vector _border_indices;

        void addSubDomainIndex(MultiIndex<3> const &index)
        {
            MultiIndex<3>::less compare;
            Indices3Vector::iterator it;
            for(it=_element_indices.begin();
            it!=_element_indices.end();
            ++it)
            {
                if(compare(*it,index))
                    continue;
                else if(compare(index,*it))
                    break;
            }
            _element_indices.insert(it,index);
        };

        void addBorderIndex(MultiIndex<2> const &index)
        {
            MultiIndex<2>::less compare;
            Indices2Vector::iterator it;
            for(it=_border_indices.begin();
            it!=_border_indices.end();
            ++it)
            {
                if(compare(*it,index))
                    continue;
                else if(compare(index,*it))
                    break;
            }
            _border_indices.insert(it,index);
        };

            public:
        Node( Point<2,X> const &point)
            : _point(point) {};

        Point<2,X> const &point() const
        {
            return _point;
        };

        inline int supportSubDomains() const
        {
            return _element_indices.size();
        };

        MultiIndex<3> const &subDomainIndex(int const &index) const
        {
            if(index<0 || supportSubDomains()<=index)
```

```
                    qFatal("SemSolver::SemSpace::Node::subDomainIndex - ERROR : there is"\
                           "no element with index element_index.");
                return _element_indices[index];
            };

            inline int supportBorders() const
            {
                return _border_indices.size();
            };

            MultiIndex<2> const &borderIndex(int const &index) const
            {
                if(index<0 || supportBorders()<=index)
                    qFatal("");
                return _border_indices[index];
            };
            friend class SemSpace;
        };

        class Element
            : public Function<Point<2,X>, X>
        {
            SemSpace const *_space;
            std::vector<X> _coefficients;
            std::vector< SemFunction<2,X> *> _base;

        public:
            Element(SemSpace const *space, Vector<X> const &coefficients)
                : _space(space)
            {
                for (int i=0; i<coefficients.dim(); ++i)
                {
                    _coefficients.push_back(coefficients[i]);
                }
            };
            ~Element() {};

            X evaluate(Point<2,X> const &x) const
            {
                double result=0;
                if(_space->_geometry.contains(x))
                {
                    for(unsigned i=0; i<_space->nodes(); ++i)
                    {
                        result += _coefficients[i]*_space->baseFunction(i)->evaluate(x);
                    }
                }
                return result;
            };
        };

        typedef MultiIndex<2>::less Index2Order;
        typedef MultiIndex<3>::less Index3Order;
        typedef std::vector<Node> NodesVector;
        typedef std::vector< SemFunction<2,X> * > SemFunctionsVector;
        typedef PointsMap<2, X, int> NodesMap;
        typedef typename NodesMap::ConstIterator NodeConstIterator;
        typedef std::map< MultiIndex<3>, int, Index3Order > ElementsMap;
        typedef typename ElementsMap::const_iterator ElementConstIterator;
        typedef std::map< MultiIndex<2>, MultiIndex<3>, Index2Order >\
```

```
                BordersMap;
        typedef std::vector<int> BordersVector;
        typedef std::map< MultiIndex<3>, double, Index3Order > WeightsMap;
        typedef typename Polygonation<2,X>::Element SubDomain;

    protected:

        SemParameters<X> const &_parameters;
        SemGeometry<2,X> const &_geometry;

    private:
        NodesVector _nodes;
        SemFunctionsVector _base;

        NodesMap _point_map;
        ElementsMap _element_map;
        BordersMap _border_map;
        std::map<int, int> _border_ids;
        BordersVector _borders;
        WeightsMap _weights;

        inline int addSubDomainNode(MultiIndex<3> const &index,
                                    Point<2,X> const &point)
        {
            int i;
            NodeConstIterator it = _point_map.find(point);
            if(it==_point_map.end())
            {
                i = _nodes.size();
                _point_map.insert(point, i);
                _nodes.push_back(point);
            }
            else
                i = it->second;
            _nodes[i].addSubDomainIndex(index);

            _element_map[index] = i;
            return i;
        };

        inline int addBorderNode(MultiIndex<2> const &border_index,
                                 MultiIndex<3> const &element_index)
        {
            ElementConstIterator it = _element_map.find(element_index);
#ifdef SEMDEBUG
            if(it==_element_map.end())
                qFatal("SemSolver::SemSpace::addBorderNode - ERROR : there is no
element"\
                       " with index element_index.");
#endif
            int i = it->second;
            _nodes[i].addBorderIndex(border_index);
            _border_map[border_index] = element_index;
            return i;
        };

        inline void addWeight(MultiIndex<3> const &index, double const &weight)
        {
            _weights[index] = weight;
        };

        inline int setBaseRestrictionPolynomialFunciton(int const &index,
```

```
                                                    int const &element_index,

                                                    Polynomial<X> const &px,
                                                    Polynomial<X> const &py)
        {
            _base[index]->setPolynomialComponent(element_index,0,px);
            _base[index]->setPolynomialComponent(element_index,1,py);
            return index;
        };

    public:

        SemSpace(SemGeometry<2,X> const &geometry,
                 SemParameters<X> const &parameters)
                     : _parameters(parameters),
                     _geometry(geometry),
                     _point_map(parameters.tolerance())
        {
            int N = parameters.degree();
            int M = subDomains();

            // compute legendre polynomials

            Polynomial<X> _1; // _legendre_0
            Polynomial<X> _x; // _legendre_1
            Polynomial<X> LN;
            Polynomial<X> DLN;
            {
                _1.setDegree(0);
                _1.setCoefficient(0,1.);
                _x.setDegree(1);
                _x.setCoefficient(0,0.);
                _x.setCoefficient(1,1.);

                Polynomial<X> _legendre_k1 = _1;
                Polynomial<X> _legendre_k2 = _x;
                Polynomial<X> t1, t2;
                for (int k=1; k<N; ++k)
                {
                    t1 = _x * _legendre_k2;
                    t1 *= (2.*k+1.)/(k+1.);
                    t2 = _legendre_k1;
                    t2 *= k/(k+1.);
                    _legendre_k1 = _legendre_k2;
                    _legendre_k2 = t1-t2;
                }
                LN = _legendre_k2;
                DLN = LN.derivative();
            }

            // compute GLL nodes

            std::vector<double> gll_nodes = DLN.zeros();
            gll_nodes.push_back(-1.);
            gll_nodes.push_back(1.);
            std::sort(gll_nodes.begin(), gll_nodes.end());

            // compute GLL polynomials

            std::vector< Polynomial<X> > gll_poly(N+1);
            {
                // i = 0
```

```
                gll_poly[0] = _1-_x; // (1-x^2) / (x-x_i)
                gll_poly[0] *= DLN; // [ (1-x^2) * DLN(x) ] / (x-x_i)
                gll_poly[0] /= -N*(N+1)*LN(-1.); // (-1) / [ N*(N+1) ] * [ (1-x^2 )
                // * DLN(x) ] / [ (x-x_i) * LN(x_i) ]

                // 0 < i < N
                for (int i=1; i<N; ++i)
                {
                    gll_poly[i] = DLN.ruffini(gll_nodes[i]); // DLN(x) / (x-x_i)
                    gll_poly[i] *= _1-_x*_x; // [ (1-x^2) * DLN(x) ] / (x-x_i)
                    gll_poly[i] /= -N*(N+1)*LN(gll_nodes[i]); // (-1) / [ N*(N+1) ]
                    // * [ (1-x^2) * DLN(x) ]
                    // / [ (x-x_i) * LN(x_i) ]
                }

                // i = N
                gll_poly[N] = _1+_x; // (-1) * (1-x^2) / (x-x_i)
                gll_poly[N] *= DLN; // (-1) * [ (1-x^2) * DLN(x) ] / (x-x_i)
                gll_poly[N] /= N*(N+1)*LN(1.); // (-1) / [ N*(N+1) ] * [ (1-x^2)
                // * DLN(x) ] / [ (x-x_i) * LN(x_i) ]
            }

            // compute GLL weights

            std::vector<double> gll_weights(N+1);
            for( int i=0; i<=N; ++i)
            {
                gll_weights[i] =  2./(N*(N+1.));
                gll_weights[i] /= LN(gll_nodes[i]) * LN(gll_nodes[i]);
            }

            // compute maps

            std::vector< BilinearTransformation<X> > maps;
            for(int i=0; i<M; ++i)
            {
                SubDomain const &element = geometry.subDomains().element(i);
                maps.push_back( BilinearTransformation<X>(element.geometry(),
                                                          _parameters.tolerance() ) );
            }

            // get borders ids form PSLG
            for(unsigned i=0; i<geometry.domain().segments(); ++i)
                _border_ids[i] = geometry.domain().segment(i).number;

            // compute nodes

            Point<2,X> x_hat;
            Point<2,X> x;
            double weight;
            MultiIndex<3> element_index;
            MultiIndex<2> border_index;

            for(int i=0; i<M; ++i)
            {
                int left = _geometry.subDomains().element(i).neighbour(0);
                int bottom = _geometry.subDomains().element(i).neighbour(1);
                int right = _geometry.subDomains().element(i).neighbour(2);
                int top = _geometry.subDomains().element(i).neighbour(3);
```

```
                if(left<0) // left is on boundary
                {
                    _borders.push_back(-left-1);
                    left = borders();
                }
                else
                    left=0;
                if(bottom<0) // bottom is on boundary
                {
                    _borders.push_back(-bottom-1);
                    bottom = borders();
                }
                else
                    bottom=0;
                if(right<0) // right is on boundary
                {
                    _borders.push_back(-right-1);
                    right = borders();
                }
                else
                    right=0;
                if(top<0) // top is on boundary
                {
                    _borders.push_back(-top-1);
                    top = borders();
                }
                else
                    top=0;

                // bottom left vertex
                x_hat = Point<2,X>(gll_nodes[0],gll_nodes[0]);
                x = maps[i].evaluate(x_hat);

                element_index.setSubIndex(0,i);
                element_index.setSubIndex(1,0);
                element_index.setSubIndex(2,0);
                weight = gll_weights[0]*gll_weights[0]*\
                         std::abs(maps[i].evaluateJacobianDeterminant(x_hat));
                addWeight(element_index,weight);
                addSubDomainNode(element_index, x);

                if(bottom)
                {
                    border_index.setSubIndex(0,bottom);
                    border_index.setSubIndex(1,0);
                    addBorderNode(border_index,element_index);
                }
                if(left)
                {
                    border_index.setSubIndex(0,left);
                    border_index.setSubIndex(1,0);
                    addBorderNode(border_index,element_index);
                }

                // bottom edge
                for (int j=1; j<N; ++j)
                {
                    x_hat = Point<2,X>(gll_nodes[j],gll_nodes[0]);
                    x = maps[i].evaluate(x_hat);

                    element_index.setSubIndex(0,i);
                    element_index.setSubIndex(1,j);
```

```
                    element_index.setSubIndex(2,0);
                    weight = gll_weights[j]*gll_weights[0]*\
                             std::abs(maps[i].evaluateJacobianDeterminant(x_hat));
                    addWeight(element_index,weight);
                    addSubDomainNode(element_index, x);

                    if(bottom)
                    {
                        border_index.setSubIndex(0,bottom);
                        border_index.setSubIndex(1,j);
                        addBorderNode(border_index,element_index);
                    }
                }

                // bottom right vertex
                x_hat = Point<2,X>(gll_nodes[N],gll_nodes[0]);
                x = maps[i].evaluate(x_hat);

                element_index.setSubIndex(0,i);
                element_index.setSubIndex(1,N);
                element_index.setSubIndex(2,0);
                weight = gll_weights[N]*gll_weights[0]*\
                         std::abs(maps[i].evaluateJacobianDeterminant(x_hat));
                addWeight(element_index,weight);
                addSubDomainNode(element_index, x);

                if(bottom)
                {
                    border_index.setSubIndex(0,bottom);
                    border_index.setSubIndex(1,N);
                    addBorderNode(border_index,element_index);
                }
                if(right)
                {
                    border_index.setSubIndex(0,right);
                    border_index.setSubIndex(1,0);
                    addBorderNode(border_index,element_index);
                }

                for(int k=1; k<N; ++k)
                {
                    // left edge
                    x_hat = Point<2,X>(gll_nodes[0],gll_nodes[k]);
                    x = maps[i].evaluate(x_hat);

                    element_index.setSubIndex(0,i);
                    element_index.setSubIndex(1,0);
                    element_index.setSubIndex(2,k);
                    weight = gll_weights[0]*gll_weights[k]*\
                             std::abs(maps[i].evaluateJacobianDeterminant(x_hat));
                    addWeight(element_index,weight);
                    addSubDomainNode(element_index, x);

                    if(left)
                    {
                        border_index.setSubIndex(0,left);
                        border_index.setSubIndex(1,k);
                        addBorderNode(border_index,element_index);
                    }

```

```
            // interior
            for(int j=1; j<N; ++j)
            {
                x_hat = Point<2,X>(gll_nodes[j],gll_nodes[k]);
                x = maps[i].evaluate(x_hat);

                element_index.setSubIndex(0,i);
                element_index.setSubIndex(1,j);
                element_index.setSubIndex(2,k);
                weight = gll_weights[j]*gll_weights[k]*\
                         std::abs(maps[i].evaluateJacobianDeterminant(x_hat));
                addWeight(element_index,weight);
                addSubDomainNode(element_index, x);
            }

            // right edge
            x_hat = Point<2,X>(gll_nodes[N],gll_nodes[k]);
            x = maps[i].evaluate(x_hat);

            element_index.setSubIndex(0,i);
            element_index.setSubIndex(1,N);
            element_index.setSubIndex(2,k);
            weight = gll_weights[N]*gll_weights[k]*\
                     std::abs(maps[i].evaluateJacobianDeterminant(x_hat));
            addWeight(element_index,weight);
            addSubDomainNode(element_index, x);

            if(right)
            {
                border_index.setSubIndex(0,right);
                border_index.setSubIndex(1,k);
                addBorderNode(border_index,element_index);
            }
        }

        // top left vertex
        x_hat = Point<2,X>(gll_nodes[0],gll_nodes[N]);
        x = maps[i].evaluate(x_hat);

        element_index.setSubIndex(0,i);
        element_index.setSubIndex(1,0);
        element_index.setSubIndex(2,N);
        weight = gll_weights[0]*gll_weights[N]*\
                 std::abs(maps[i].evaluateJacobianDeterminant(x_hat));
        addWeight(element_index,weight);
        addSubDomainNode(element_index, x);

        if(top)
        {
            border_index.setSubIndex(0,top);
            border_index.setSubIndex(1,0);
            addBorderNode(border_index,element_index);
        }
        if(left)
        {
            border_index.setSubIndex(0,left);
            border_index.setSubIndex(1,N);
            addBorderNode(border_index,element_index);
        }

```

```
            // top edge
            for(int j=1; j<N; ++j)
            {
                x_hat = Point<2,X>(gll_nodes[j],gll_nodes[N]);
                x = maps[i].evaluate(x_hat);

                element_index.setSubIndex(0,i);
                element_index.setSubIndex(1,j);
                element_index.setSubIndex(2,N);
                weight = gll_weights[j]*gll_weights[N]*\
                         std::abs(maps[i].evaluateJacobianDeterminant(x_hat))
;
                addWeight(element_index,weight);
                addSubDomainNode(element_index, x);

                if(top)
                {
                    border_index.setSubIndex(0,top);
                    border_index.setSubIndex(1,j);
                    addBorderNode(border_index,element_index);
                }
            }

            // top right vertex
            x_hat = Point<2,X>(gll_nodes[N],gll_nodes[N]);
            x = maps[i].evaluate(x_hat);

            element_index.setSubIndex(0,i);
            element_index.setSubIndex(1,N);
            element_index.setSubIndex(2,N);
            weight = gll_weights[N]*gll_weights[N]*\
                     std::abs(maps[i].evaluateJacobianDeterminant(x_hat));
            addWeight(element_index,weight);
            addSubDomainNode(element_index, x);

            if(top)
            {
                border_index.setSubIndex(0,top);
                border_index.setSubIndex(1,N);
                addBorderNode(border_index,element_index);
            }
            if(right)
            {
                border_index.setSubIndex(0,right);
                border_index.setSubIndex(1,N);
                addBorderNode(border_index,element_index);
            }
        }


        // base functions
        std::vector< PolynomialFunction<2,X> > zero_polys(M);
        for(unsigned i=0; i<_nodes.size(); ++i)
            _base.push_back(new SemFunction<2,X>(_geometry,zero_polys,maps));


        for(int i=0; i<M; ++i)
        {
            for(int j=0; j<=N; ++j)
            {
                for(int k=0; k<=N; ++k)
                {
```

```
                        element_index.setSubIndex(0,i);
                        element_index.setSubIndex(1,j);
                        element_index.setSubIndex(2,k);
                        int index = subDomainIndex(element_index);
                        setBaseRestrictionPolynomialFunciton(index,i,gll_poly[j],
                                                                     gll_poly[k]);
                    }
                }
            }
        }

        unsigned nodes() const
        {
            return _nodes.size();
        };

        inline const Node &node(const unsigned &index) const
        {
            if( index>=nodes() )
                qFatal("SemSolver::SemSpace::node : index out of range");
            return _nodes[index];
        };

        inline int subDomains() const
        {
            return _geometry.subDomains().size();
        };

        inline int const &subDomainIndex(MultiIndex<3> const &index) const
        {
            std::map<MultiIndex<3>, int, MultiIndex<3>::less>::const_iterator it =
                        _element_map.find(index);
#ifdef SEMDEBUG
            if(it==_element_map.end())
                qFatal("SemSolver::SemSpace::subDomainIndex - ERROR : there is no elemen"\
                       "t with multi-index index.");
#endif
            return it->second;
        }

        inline Node const &subDomainNode(MultiIndex<3> const &index) const
        {
            return node(subDomainIndex(index));
        };

        inline double const &subDomainWeight(MultiIndex<3> const &index) const
        {
            std::map<MultiIndex<3>, double, MultiIndex<3>::less>::const_iterator it =
                        _weights.find(index);
            if(it==_weights.end())
            {
                qFatal("SemSolver::SemSpace::subDomainWeight - ERROR : there is no weigh"\
                       "t with multi-index index.");
            }
            return it->second;
        };

```

```
        inline unsigned borders() const
        {
            return _borders.size();
        };

        inline int border(int const &i) const
        {
            return _borders[i];
        };

        inline int borderId(int const &border) const
        {
            std::map<int,int>::const_iterator it = _border_ids.find(border);
            if(it!=_border_ids.end())
                return it->second;
            return -1;
        };

        inline MultiIndex<3> const &borderSubDomainIndex(MultiIndex<2> const &ind
ex) const
        {
            std::map<MultiIndex<2>, MultiIndex<3>, MultiIndex<2>::less>::const_it
erator it
                    = _border_map.find(index);
#ifdef SEMDEBUG
            if(it==_border_map.end())
                qFatal("SemSolver::SemSpace::borderSubDomainIndex - ERROR : there
 is no "\
                       "border with multi-index index.");
#endif
            return it->second;
        };

        inline int const &borderIndex(MultiIndex<2> const &index) const
        {
            return subDomainIndex(borderSubDomainIndex(index));
        };

        inline Node const &borderNode(MultiIndex<2> const &index) const
        {
            return subDomainNode(borderSubDomainIndex(index));
        };

        inline double const &borderWeight(MultiIndex<2> const &index) const
        {
            return subDomainWeight(borderSubDomainIndex(index));
        };

        double scalarProduct(Element first, Element second)
        {
            /*********************************************/
            return 0;
        };

        SemFunction<2,X> const *baseFunction(const unsigned &index) const
        {
#if SEMDEBUG
            if(index>=nodes())
                qFatal("SemSolver::HilbertSpace::baseFunction - ERROR : index out
 of ran"\
                       "ge");
#endif
```

```
            return _base[index];
        };

        inline int const &degree() const
        {
            return _parameters.degree();
        };
    };
}

#endif // SEMSPACE_HPP
```

7.99 sequence.hpp File Reference

`#include <list>`

Namespaces

- namespace SemSolver

  Project main namespace.

Typedefs

- typedef std::list< int > SemSolver::Sequence

7.100 sequence.hpp

```
#ifndef SEQUENCE_HPP
#define SEQUENCE_HPP

#include <list>

namespace SemSolver
{
    typedef std::list<int> Sequence;
};

#endif // SEQUENCE_HPP
```

7.101 sequenceslist.hpp File Reference

`#include <list>`

`#include <SemSolver/sequence.hpp>`

Namespaces

- namespace SemSolver

    Project main namespace.

Typedefs

- typedef std::list< Sequence > SemSolver::Sequences_list

7.102 sequenceslist.hpp

```
#ifndef SEQUENCESLIST_HPP
#define SEQUENCESLIST_HPP

#include <list>

#include <SemSolver/sequence.hpp>

namespace SemSolver
{
    typedef std::list<Sequence> Sequences_list;
};

#endif // SEQUENCESLIST_HPP
```

7.103 subdomains.hpp File Reference

```
#include <QFile>
#include <QTextStream>
#include <SemSolver/polygonation.hpp>
```

Namespaces

- namespace SemSolver

    Project main namespace.

- namespace SemSolver::IO

    Namespace for Input/Output operations on SemSolver Classes.

Functions

- template<class X >
  bool SemSolver::IO::read_subdomains (QFile *file, Polygonation< 2, X > &sub_-domains)

Read subdomains Polygonation from file.

- template<class X >
  bool SemSolver::IO::write_subdomains (Polygonation< 2, X > const &sub_domains, QFile ∗file)

  Write subdomains Polygonation to file.

7.104 subdomains.hpp

```
#ifndef IO_SUBDOMAINS_HPP
#define IO_SUBDOMAINS_HPP

#include <QFile>
#include <QTextStream>
#ifdef SEMDEBUG
#  include <QDebug>
#endif

#include  <SemSolver/polygonation.hpp>

namespace SemSolver
{
    namespace IO
    {
        template<class X>
        bool read_subdomains(QFile *file,
                                     Polygonation<2, X> &sub_domains)
{
    if(!file->open(QIODevice::ReadOnly))
        return false;
    QTextStream input(file);
    sub_domains.clear();
    int size, polygon_size, id, neighbour;
    double x, y;
    Polygon<2,X> polygon;
    std::vector<int> neighbours;
    input >> size;
    for(int i=0; i<size; ++i)
    {
        polygon.clear();
        neighbours.clear();
        input >> id;
#ifdef SEMDEBUG
        if(id!=i+1)
        {
            qWarning() << "corrupted domains file";
            return false;
        }
#endif
        input >> polygon_size;
        for(int j=0; j<polygon_size; ++j)
        {
            input >> x;
            input >> y;
            input >> neighbour;
            polygon.push_back(Point<2,X>(x,y));
            neighbours.push_back(neighbour);
```

```
        }
        sub_domains.addElement(polygon, neighbours);
    }
    file->close();
    return true;
};

        template<class X>
        bool write_subdomains(Polygonation<2, X> const &sub_domains,
                              QFile *file)
{
    if(!file->open(QIODevice::WriteOnly))
        return false;
    QTextStream output(file);
    int n = sub_domains.size();
    output << n << "\n";
    for(int i=0; i<n; ++i)
    {
        int m = sub_domains.element(i).size();
        output << i+1 << "\t" << m << "\n";
        std::vector<int>::const_iterator it = sub_domains.element(i).neighboursBegin();
        for(int j=0; j<m; ++j)
        {
            output << sub_domains.element(i).vertex(j).x() << "\t";
            output << sub_domains.element(i).vertex(j).y() << "\t";
            output << *it++ << "\n";
        }
    }
    file->close();
    return true;
};
    };
};

#endif // IO_SUBDOMAINS_HPP
```

7.105 vector.hpp File Reference

```
#include <tnt_array1d.h>
#include <SemSolver/point.hpp>
```

Classes

- class SemSolver::Vector< X >

Namespaces

- namespace SemSolver

  Project main namespace.

Functions

- template<class X >
  Vector< X > SemSolver::operator+ (Vector< X > const &vec1, Vector< X > const &vec2)

  Vector summation.

- template<class X >
  X SemSolver::scalar (Vector< X > const &vec1, Vector< X > const &vec2)

  Vector multiplication for a scalar.

7.106 vector.hpp

```
#ifndef VECTOR_HPP
#define VECTOR_HPP

namespace SemSolver
{
    template<class X>
    class Vector;
};

#include <tnt_array1d.h>

#include <SemSolver/point.hpp>

namespace SemSolver
{
    template<class X>
    class Vector
        : public TNT::Array1D<X>
    {
        typedef TNT::Array1D<X> TNT_array_1d;
    public:
        Vector() : TNT_array_1d() {};

        Vector(TNT_array_1d const &vector) : TNT_array_1d(vector) {};

        Vector(int dimension) : TNT_array_1d(dimension) {};
        Vector(int dimension, X const &value) : TNT_array_1d(dimension, value) {}
;

        template<int d>
        Vector(Point<d, X> const &point)
            : TNT_array_1d(d)
        {
            for(int i=0; i<d; ++i)
                (*this)[i] = point.cartesian(i);
        }

        inline int rows() const
        {
            return this->dim1();
        };
    };

```

```
    template<class X>
    Vector<X> operator +(Vector<X> const &vec1, Vector<X> const &vec2)
    {
        if(vec1.dim() != vec2.dim())
            qFatal("");
        return vec1+vec2;
    };

    template<class X>
    X scalar(Vector<X> const &vec1, Vector<X> const &vec2)
    {
        if(vec1.dim() != vec2.dim())
            qFatal("SemSolver::Vector::scalar - ERROR : vec1 and vec2 must have t
he same"\
                   " dimension.");
        X product = 0;
        for(int i=0; i<vec1.dim(); ++i)
            product += vec1[i]*vec2[i];
        return product;
    };
};

#endif // VECTOR_HPP
```

7.107 workspace.hpp File Reference

```
#include <QFile>
#include <QString>
#include <QStringList>
#include <SemSolver/semgeometry.hpp>
#include <SemSolver/equation.hpp>
#include <SemSolver/boundaryconditions.hpp>
#include <SemSolver/semparameters.hpp>
#include <SemSolver/IO/geometry.hpp>
#include <SemSolver/IO/equation.hpp>
#include <SemSolver/IO/boundaryconditions.hpp>
#include <SemSolver/IO/parameters.hpp>
```

Namespaces

- namespace SemSolver

  Project main namespace.

- namespace SemSolver::IO

  Namespace for Input/Output operations on SemSolver Classes.

Functions

- bool SemSolver::IO::get_geometries_list_from_workspace (QFile ∗file, QStringList &geometries)

  Get list of geometries in workspace.

- bool SemSolver::IO::get_equations_list_from_workspace (QFile ∗file, QStringList &equations)

  Get list of equations in workspace.

- bool SemSolver::IO::get_boundary_conditions_list_from_workspace (QFile ∗file, QStringList &boundary_conditions)

  Get list of boundary conditions in workspace.

- bool SemSolver::IO::get_parameters_list_from_workspace (QFile ∗file, QStringList &parameters)

  Get list of parameters in workspace.

- template<class X >
  bool SemSolver::IO::get_geometry_from_workspace (QFile ∗file, QString const &name, SemGeometry< 2, X > &geometry)

  Read a geometry from workspace.

- template<class X >
  bool SemSolver::IO::get_equation_from_workspace (QFile ∗file, QString const &name, Equation< 2, X > ∗&equation)

  Read an equation from workspace.

- template<class X >
  bool SemSolver::IO::get_boundary_conditions_from_workspace (QFile ∗file, QString const &name, BoundaryConditions< 2, X > &bc)

  Read boundary conditions from workspace.

- template<class X >
  bool SemSolver::IO::get_parameters_from_workspace (QFile ∗file, QString const &name, SemParameters< X > &bc)

  Read parameters from workspace.

- bool SemSolver::IO::extract_file_from_workspace (QFile ∗workspace, QString const &name, QFile ∗file)

  Extract an entry from workspace.

- bool SemSolver::IO::add_file_to_workspace (QFile ∗workspace, QString const &name, QFile ∗file)

  Add an entry to workspace.

- bool SemSolver::IO::remove_file_from_workspace (QFile ∗workspace, QString const &name)

Remove an entry from workspace.

7.108 workspace.hpp

```
#ifndef WORKSPACE_HPP
#define WORKSPACE_HPP

#include <QFile>
#include <QString>
#include <QStringList>

#include <SemSolver/semgeometry.hpp>
#include <SemSolver/equation.hpp>
#include <SemSolver/boundaryconditions.hpp>
#include <SemSolver/semparameters.hpp>

#include  <SemSolver/IO/geometry.hpp>
#include  <SemSolver/IO/equation.hpp>
#include  <SemSolver/IO/boundaryconditions.hpp>
#include  <SemSolver/IO/parameters.hpp>

namespace SemSolver
{
    namespace IO
    {
        bool get_geometries_list_from_workspace(QFile *file,
                                                QStringList &geometries);
        bool get_equations_list_from_workspace(QFile *file,
                                               QStringList &equations);
        bool get_boundary_conditions_list_from_workspace(QFile *file,
                                                         QStringList &boundary_conditions);
        bool get_parameters_list_from_workspace(QFile *file,
                                                QStringList &parameters);
        template<class X>
        bool get_geometry_from_workspace(QFile *file,
                                         QString const &name,
                                         SemGeometry<2, X> &geometry);
        template<class X>
        bool get_equation_from_workspace(QFile *file,
                                         QString const &name,
                                         Equation<2, X> *&equation);
        template<class X>
        bool get_boundary_conditions_from_workspace(QFile *file,
                                                    QString const &name,
                                                    BoundaryConditions<2, X> &bc);
        template<class X>
        bool get_parameters_from_workspace(QFile *file,
                                           QString const &name,
                                           SemParameters<X> &bc);
        bool extract_file_from_workspace(QFile *workspace,
                                         QString const &name,
                                         QFile *file);
        bool add_file_to_workspace(QFile *workspace,
                                   QString const &name,
                                   QFile *file);
        bool remove_file_from_workspace(QFile *workspace,
                                        QString const &name);
```

```
    };
};

template<class X>
bool SemSolver::IO::get_geometry_from_workspace(QFile *file,
                                                const QString &name,
                                                SemGeometry<2,X> &geometry)
{
    QTemporaryFile *temp_file = new QTemporaryFile();
    if(!extract_file_from_workspace(file, name + ".semgeo", temp_file))
    {
        delete temp_file;
        return false;
    }
    if(!read_geometry(temp_file, geometry))
    {
        delete temp_file;
        return false;
    }
    delete temp_file;
    return true;
};

template<class X>
bool SemSolver::IO::get_equation_from_workspace(QFile *file,
                                                const QString &name,
                                                Equation<2, X> *&equation)
{
    QTemporaryFile *temp_file = new QTemporaryFile();
    if(!extract_file_from_workspace(file, name + ".semeqn", temp_file))
    {
        delete temp_file;
        return false;
    }
    if(!read_equation(temp_file, equation))
    {
        delete temp_file;
        return false;
    }
    delete temp_file;
    return true;
};

template<class X>
bool SemSolver::IO::get_boundary_conditions_from_workspace(QFile *file,
                                                           const QString &name,
                                                           BoundaryConditions<2, X> &bc)
{
    QTemporaryFile *temp_file = new QTemporaryFile();
    if(!extract_file_from_workspace(file, name + ".sembcs", temp_file))
    {
        delete temp_file;
        return false;
    }
    if(!read_boundary_conditions(temp_file, bc))
    {
        delete temp_file;
        return false;
    }
    delete temp_file;
    return true;
```

```
};

template<class X>
bool SemSolver::IO::get_parameters_from_workspace(QFile *file,
                                                 const QString &name,
                                                 SemParameters<X> &parameters)
{
    QTemporaryFile *temp_file = new QTemporaryFile();
    if(!extract_file_from_workspace(file, name + ".semprm", temp_file))
    {
        delete temp_file;
        return false;
    }
    if(!read_parameters(temp_file, parameters))
    {
        delete temp_file;
        return false;
    }
    delete temp_file;
    return true;
};

#endif // WORKSPACE_HPP
```

Index