

SemSolver

Generated by Doxygen 1.7.3

Mon Sep 19 2011 18:17:27

Contents

1	Namespace Index	1
1.1	Namespace List	1
2	Class Index	3
2.1	Class Hierarchy	3
3	Class Index	5
3.1	Class List	5
4	File Index	7
4.1	File List	7
5	Namespace Documentation	9
5.1	SemSolver Namespace Reference	9
5.1.1	Detailed Description	12
5.1.2	Typedef Documentation	13
5.1.2.1	Sequence	13
5.1.2.2	Sequences_list	13
5.1.3	Function Documentation	13
5.1.3.1	operator*	13
5.1.3.2	operator+	13
5.1.3.3	operator+	13
5.1.3.4	scalar	14
5.2	SemSolver::Assembler Namespace Reference	14
5.2.1	Detailed Description	15
5.2.2	Function Documentation	15
5.2.2.1	compute_algebraic_system	15
5.2.2.2	compute_border_matrix	15
5.2.2.3	compute_border_vector	15
5.2.2.4	compute_convection_matrix	16
5.2.2.5	compute_diffusion_matrix	16
5.2.2.6	compute_forcing_vector	16
5.2.2.7	compute_reaction_matrix	16
5.3	SemSolver::IO Namespace Reference	16
5.3.1	Detailed Description	18
5.3.2	Function Documentation	18
5.3.2.1	add_file_to_workspace	18
5.3.2.2	extract_file_from_workspace	19
5.3.2.3	get_boundary_conditions_from_workspace	19

5.3.2.4	get_boundary_conditions_list_from_workspace	19
5.3.2.5	get_equation_from_workspace	19
5.3.2.6	get_equations_list_from_workspace	19
5.3.2.7	get_geometries_list_from_workspace	19
5.3.2.8	get_geometry_from_workspace	19
5.3.2.9	get_parameters_from_workspace	19
5.3.2.10	get_parameters_list_from_workspace	20
5.3.2.11	next_non_empty_line_values	20
5.3.2.12	read_boundary_conditions	20
5.3.2.13	read_equation	20
5.3.2.14	read_geometry	20
5.3.2.15	read_parameters	20
5.3.2.16	read_PSLG	20
5.3.2.17	read_subdomains	20
5.3.2.18	remove_file_from_workspace	21
5.3.2.19	write_geometry	21
5.3.2.20	write_subdomains	21
5.4	SemSolver::PostProcessor Namespace Reference	21
5.4.1	Function Documentation	21
5.4.1.1	build_solution	21
5.4.1.2	compute_plot_data	21
5.4.1.3	compute_solution_hull	22
5.5	SemSolver::PreProcessor Namespace Reference	22
5.5.1	Detailed Description	22
5.5.2	Function Documentation	22
5.5.2.1	compute_polygon_with_holes_from_pslg	22
5.5.2.2	compute_polygonation_from_pslg	23
5.5.2.3	compute_vertices_sequences_from_pslg	23
5.6	SemSolver::Solver Namespace Reference	23
5.6.1	Detailed Description	23
5.6.2	Function Documentation	23
5.6.2.1	cholesky_solve	23
5.6.2.2	lu_solve	24
5.6.2.3	qr_solve	24
6	Class Documentation	25
6.1	SemSolver::IO::Archive Class Reference	25
6.1.1	Detailed Description	26
6.1.2	Constructor & Destructor Documentation	26
6.1.2.1	Archive	26
6.1.3	Member Function Documentation	26
6.1.3.1	addFile	26
6.1.3.2	addFile	26
6.1.3.3	addValue	26
6.1.3.4	closeRead	26
6.1.3.5	closeWrite	27
6.1.3.6	entries	27
6.1.3.7	extractFile	27
6.1.3.8	extractFile	27
6.1.3.9	openRead	27

6.1.3.10	openWrite	27
6.2	SemSolver::BilinearTransformation< X > Class Template Reference	27
6.2.1	Detailed Description	29
6.2.2	Constructor & Destructor Documentation	29
6.2.2.1	BilinearTransformation	29
6.2.2.2	BilinearTransformation	29
6.2.2.3	BilinearTransformation	29
6.2.2.4	~BilinearTransformation	30
6.2.3	Member Function Documentation	30
6.2.3.1	evaluate	30
6.2.3.2	evaluateInverse	30
6.2.3.3	evaluateJacobianDeterminant	31
6.2.3.4	evaluateTransposeInverseJacobian	31
6.2.3.5	omega	31
6.2.3.6	setOmega	31
6.2.3.7	setTolerance	32
6.2.3.8	tolerance	32
6.3	SemSolver::BoundaryConditions< d, X > Class Template Reference	32
6.3.1	Detailed Description	32
6.4	SemSolver::BoundaryConditions< 2, X > Class Template Reference	33
6.4.1	Detailed Description	34
6.4.2	Member Typedef Documentation	34
6.4.2.1	FunctionConstIterator	34
6.4.2.2	FunctionPtr	35
6.4.2.3	FunctionsMap	35
6.4.2.4	TypesMap	35
6.4.3	Member Enumeration Documentation	35
6.4.3.1	Type	35
6.4.4	Constructor & Destructor Documentation	35
6.4.4.1	BoundaryConditions	35
6.4.4.2	BoundaryConditions	35
6.4.4.3	BoundaryConditions	36
6.4.5	Member Function Documentation	36
6.4.5.1	borderType	36
6.4.5.2	clear	36
6.4.5.3	conditions	36
6.4.5.4	dirichletData	37
6.4.5.5	labels	37
6.4.5.6	mmls	37
6.4.5.7	neumannData	38
6.4.5.8	robinCoefficient	38
6.4.5.9	robinData	38
6.4.5.10	setBorder	39
6.5	SemSolver::DiffusionConvectionReactionEquation< d, X > Class Template Reference	39
6.5.1	Detailed Description	40
6.5.2	Member Typedef Documentation	41
6.5.2.1	Type	41
6.5.3	Constructor & Destructor Documentation	41
6.5.3.1	DiffusionConvectionReactionEquation	41

6.5.3.2	~DiffusionConvectionReactionEquation	41
6.5.4	Member Function Documentation	41
6.5.4.1	convection	41
6.5.4.2	diffusion	41
6.5.4.3	forcing	42
6.5.4.4	mml	42
6.5.4.5	reaction	42
6.5.4.6	setConvection	42
6.5.4.7	setDiffusion	43
6.5.4.8	setForcing	43
6.5.4.9	setReaction	43
6.5.4.10	type	43
6.6	SemSolver::DiffusionConvectionReactionEquation< 2, X > Class Template Reference	44
6.6.1	Detailed Description	45
6.6.2	Member Typedef Documentation	45
6.6.2.1	Type	45
6.6.3	Constructor & Destructor Documentation	46
6.6.3.1	DiffusionConvectionReactionEquation	46
6.6.3.2	~DiffusionConvectionReactionEquation	46
6.6.4	Member Function Documentation	46
6.6.4.1	convection	46
6.6.4.2	diffusion	46
6.6.4.3	forcing	46
6.6.4.4	mml	47
6.6.4.5	reaction	47
6.6.4.6	setConvection	47
6.6.4.7	setDiffusion	47
6.6.4.8	setForcing	48
6.6.4.9	setReaction	48
6.6.4.10	type	48
6.7	SemSolver::HilbertSpace< Function, X >::Element Class Reference	48
6.7.1	Detailed Description	49
6.7.2	Constructor & Destructor Documentation	49
6.7.2.1	Element	49
6.7.2.2	~Element	49
6.8	SemSolver::Polygonation< 2, X >::Element Class Reference	49
6.8.1	Detailed Description	51
6.8.2	Constructor & Destructor Documentation	51
6.8.2.1	Element	51
6.8.2.2	Element	51
6.8.3	Member Function Documentation	51
6.8.3.1	clear	51
6.8.3.2	contains	51
6.8.3.3	geometry	52
6.8.3.4	neighbour	52
6.8.3.5	neighboursBegin	52
6.8.3.6	neighboursEnd	52
6.8.3.7	setGeometry	53
6.8.3.8	setNeighbour	53

6.8.3.9	size	53
6.8.3.10	vertex	53
6.8.3.11	vertexPosition	54
6.8.3.12	verticesBegin	54
6.8.3.13	verticesEnd	54
6.9	SemSolver::SemSpace< 2, X >::Element Class Reference	54
6.9.1	Detailed Description	55
6.9.2	Constructor & Destructor Documentation	55
6.9.2.1	Element	55
6.9.2.2	~Element	55
6.9.3	Member Function Documentation	55
6.9.3.1	evaluate	55
6.10	SemSolver::Equation< d, X > Class Template Reference	56
6.10.1	Detailed Description	57
6.10.2	Member Enumeration Documentation	57
6.10.2.1	Type	57
6.10.3	Constructor & Destructor Documentation	57
6.10.3.1	Equation	57
6.10.3.2	~Equation	57
6.10.4	Member Function Documentation	58
6.10.4.1	mml	58
6.10.4.2	type	58
6.11	SemSolver::Function< X, Y > Class Template Reference	58
6.11.1	Detailed Description	59
6.11.2	Constructor & Destructor Documentation	59
6.11.2.1	Function	59
6.11.2.2	~Function	59
6.11.3	Member Function Documentation	59
6.11.3.1	evaluate	59
6.11.3.2	mml	60
6.12	SemSolver::HilbertSpace< Function, X > Class Template Reference	60
6.12.1	Detailed Description	61
6.12.2	Member Function Documentation	61
6.12.2.1	baseFunction	61
6.12.2.2	dimension	62
6.12.2.3	norm	62
6.12.2.4	projection	62
6.12.2.5	scalarProduct	62
6.13	SemSolver::PSLG< X >::Hole Struct Reference	63
6.13.1	Detailed Description	63
6.13.2	Member Data Documentation	63
6.13.2.1	number	63
6.13.2.2	x	63
6.13.2.3	y	63
6.14	SemSolver::Homeomorphism< X, Y > Class Template Reference	63
6.14.1	Detailed Description	64
6.14.2	Constructor & Destructor Documentation	64
6.14.2.1	Homeomorphism	64
6.14.2.2	~Homeomorphism	65
6.14.3	Member Function Documentation	65

6.14.3.1	evaluateInverse	65
6.15	SemSolver::Segment< 2, X >::less Struct Reference	65
6.15.1	Detailed Description	65
6.15.2	Member Function Documentation	65
6.15.2.1	operator()	65
6.16	SemSolver::MultiIndex< N >::less Struct Reference	66
6.16.1	Detailed Description	66
6.16.2	Member Function Documentation	66
6.16.2.1	operator()	66
6.17	SemSolver::Matrix< X > Class Template Reference	66
6.17.1	Detailed Description	67
6.17.2	Constructor & Destructor Documentation	67
6.17.2.1	Matrix	67
6.17.2.2	Matrix	67
6.17.2.3	Matrix	68
6.17.3	Member Function Documentation	68
6.17.3.1	columns	68
6.17.3.2	realEigenvalues	68
6.17.3.3	rows	68
6.17.4	Friends And Related Function Documentation	69
6.17.4.1	operator+	69
6.18	SemSolver::MultiIndex< N > Class Template Reference	69
6.18.1	Detailed Description	69
6.18.2	Constructor & Destructor Documentation	70
6.18.2.1	MultiIndex	70
6.18.2.2	MultiIndex	70
6.18.2.3	MultiIndex	70
6.18.3	Member Function Documentation	70
6.18.3.1	setSubIndex	70
6.18.3.2	subIndex	71
6.19	SemSolver::SemSpace< 2, X >::Node Class Reference	71
6.19.1	Detailed Description	72
6.19.2	Constructor & Destructor Documentation	72
6.19.2.1	Node	72
6.19.3	Member Function Documentation	72
6.19.3.1	borderIndex	72
6.19.3.2	point	72
6.19.3.3	subDomainIndex	72
6.19.3.4	supportBorders	73
6.19.3.5	supportSubDomains	73
6.19.4	Friends And Related Function Documentation	73
6.19.4.1	SemSpace	73
6.20	SemSolver::Point< 2, X > Class Template Reference	73
6.20.1	Detailed Description	73
6.20.2	Constructor & Destructor Documentation	74
6.20.2.1	Point	74
6.20.2.2	Point	74
6.20.2.3	Point	74
6.21	SemSolver::PointsBimap< 2, X > Class Template Reference	74
6.21.1	Detailed Description	76

6.21.2	Member Typedef Documentation	76
6.21.2.1	ConstIterator	76
6.21.2.2	KeyType	76
6.21.2.3	MappedType	76
6.21.2.4	SizeType	76
6.21.2.5	ValueType	77
6.21.3	Constructor & Destructor Documentation	77
6.21.3.1	PointsBimap	77
6.21.4	Member Function Documentation	77
6.21.4.1	eraseId	77
6.21.4.2	erasePoint	77
6.21.4.3	findId	77
6.21.4.4	findId	78
6.21.4.5	findPoint	78
6.21.4.6	findPoint	78
6.21.4.7	hasId	79
6.21.4.8	hasPoint	79
6.21.4.9	hasPointOn	79
6.21.4.10	id	80
6.21.4.11	insert	80
6.21.4.12	insertPoint	80
6.21.4.13	modifyId	80
6.21.4.14	modifyPoint	81
6.21.4.15	point	81
6.22	SemSolver::PointsMap< 2, X, Y > Class Template Reference	81
6.22.1	Detailed Description	83
6.22.2	Member Typedef Documentation	83
6.22.2.1	ConstIterator	83
6.22.2.2	Iterator	83
6.22.2.3	KeyType	83
6.22.2.4	MappedType	84
6.22.2.5	SizeType	84
6.22.2.6	ValueType	84
6.22.3	Constructor & Destructor Documentation	84
6.22.3.1	PointsMap	84
6.22.4	Member Function Documentation	84
6.22.4.1	begin	84
6.22.4.2	begin	84
6.22.4.3	clear	85
6.22.4.4	end	85
6.22.4.5	end	85
6.22.4.6	erase	85
6.22.4.7	erase	85
6.22.4.8	erase	86
6.22.4.9	find	86
6.22.4.10	find	86
6.22.4.11	has	86
6.22.4.12	insert	87
6.22.4.13	insert	87
6.22.4.14	insert	87

6.22.4.15	insert	87
6.22.4.16	isEmpty	88
6.22.4.17	operator[]	88
6.22.4.18	size	88
6.23	SemSolver::PointsSet< 2, X > Class Template Reference	89
6.23.1	Detailed Description	90
6.23.2	Member Typedef Documentation	91
6.23.2.1	ConstIterator	91
6.23.2.2	Iterator	91
6.23.2.3	KeyType	91
6.23.2.4	MappedType	91
6.23.2.5	SizeType	91
6.23.2.6	ValueType	91
6.23.3	Constructor & Destructor Documentation	91
6.23.3.1	PointsSet	91
6.23.4	Member Function Documentation	92
6.23.4.1	begin	92
6.23.4.2	begin	92
6.23.4.3	clear	92
6.23.4.4	end	92
6.23.4.5	end	92
6.23.4.6	erase	92
6.23.4.7	erase	93
6.23.4.8	erase	93
6.23.4.9	find	93
6.23.4.10	find	93
6.23.4.11	has	94
6.23.4.12	insert	94
6.23.4.13	insert	94
6.23.4.14	insert	94
6.23.4.15	isEmpty	95
6.23.4.16	operator[]	95
6.23.4.17	size	95
6.24	SemSolver::Polygon< 2, X > Class Template Reference	96
6.24.1	Detailed Description	96
6.24.2	Constructor & Destructor Documentation	96
6.24.2.1	Polygon	96
6.24.2.2	Polygon	97
6.24.3	Member Function Documentation	97
6.24.3.1	contains	97
6.25	SemSolver::Polygonation< 2, X > Class Template Reference	97
6.25.1	Detailed Description	98
6.25.2	Member Function Documentation	98
6.25.2.1	addElement	98
6.25.2.2	clear	98
6.25.2.3	element	99
6.25.2.4	elementIndicesAt	99
6.25.2.5	isQuadrangulation	99
6.25.2.6	refine	99
6.25.2.7	size	100

6.26	SemSolver::PolygonWithHoles< 2, X > Class Template Reference	100
6.26.1	Detailed Description	100
6.26.2	Member Function Documentation	101
6.26.2.1	cgal	101
6.27	SemSolver::Polynomial< X > Class Template Reference	101
6.27.1	Detailed Description	103
6.27.2	Constructor & Destructor Documentation	103
6.27.2.1	Polynomial	103
6.27.2.2	Polynomial	103
6.27.2.3	Polynomial	103
6.27.2.4	Polynomial	104
6.27.2.5	~Polynomial	104
6.27.3	Member Function Documentation	104
6.27.3.1	coefficient	104
6.27.3.2	degree	104
6.27.3.3	derivative	104
6.27.3.4	operator!=	105
6.27.3.5	operator%	105
6.27.3.6	operator%=	105
6.27.3.7	operator()	105
6.27.3.8	operator*	105
6.27.3.9	operator*==	106
6.27.3.10	operator+	106
6.27.3.11	operator+	106
6.27.3.12	operator+=	106
6.27.3.13	operator-	106
6.27.3.14	operator-	107
6.27.3.15	operator-=	107
6.27.3.16	operator/	107
6.27.3.17	operator/=	107
6.27.3.18	operator=	108
6.27.3.19	operator==	108
6.27.3.20	ruffini	108
6.27.3.21	setCoefficient	108
6.27.3.22	setDegree	108
6.27.3.23	zeros	109
6.28	SemSolver::PolynomialFunction< d, X > Class Template Reference	109
6.28.1	Detailed Description	110
6.28.2	Constructor & Destructor Documentation	110
6.28.2.1	PolynomialFunction	110
6.28.2.2	PolynomialFunction	110
6.28.2.3	PolynomialFunction	110
6.28.2.4	~PolynomialFunction	111
6.28.3	Member Function Documentation	111
6.28.3.1	evaluate	111
6.28.3.2	operator=	111
6.28.3.3	polynomial	111
6.28.3.4	setPolynomial	111
6.29	SemSolver::Problem< d, X > Class Template Reference	112
6.29.1	Detailed Description	113

6.29.2	Constructor & Destructor Documentation	113
6.29.2.1	Problem	113
6.29.2.2	~Problem	113
6.29.3	Member Function Documentation	113
6.29.3.1	boundaryConditions	113
6.29.3.2	clearBoundaryConditions	113
6.29.3.3	clearEquation	114
6.29.3.4	clearGeometry	114
6.29.3.5	clearParameters	114
6.29.3.6	equation	114
6.29.3.7	geometry	114
6.29.3.8	isDefined	114
6.29.3.9	parameters	114
6.29.3.10	setBoundaryConditions	115
6.29.3.11	setEquation	115
6.29.3.12	setGeometry	115
6.29.3.13	setParameters	115
6.30	SemSolver::PSLG< X > Class Template Reference	115
6.30.1	Detailed Description	116
6.30.2	Constructor & Destructor Documentation	117
6.30.2.1	PSLG	117
6.30.2.2	~PSLG	117
6.30.3	Member Function Documentation	117
6.30.3.1	clear	117
6.30.3.2	hole	117
6.30.3.3	holes	117
6.30.3.4	segment	118
6.30.3.5	segments	118
6.30.3.6	setHole	118
6.30.3.7	setNumberOfHoles	118
6.30.3.8	setNumberOfSegments	119
6.30.3.9	setNumberOfVertices	119
6.30.3.10	setNumberOfVerticesAttributes	119
6.30.3.11	setNumberOfVerticesBoundaryMarkers	119
6.30.3.12	setSegment	120
6.30.3.13	setVertex	120
6.30.3.14	vertex	120
6.30.3.15	vertices	121
6.31	SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > > Class Template Reference	121
6.31.1	Detailed Description	122
6.31.2	Constructor & Destructor Documentation	122
6.31.2.1	ScriptFunction	122
6.31.2.2	~ScriptFunction	122
6.31.3	Member Function Documentation	122
6.31.3.1	evaluate	122
6.31.3.2	mml	122
6.32	SemSolver::ScriptFunction< Point< 2, X >, Y > Class Template Reference	123
6.32.1	Detailed Description	123

6.32.2	Constructor & Destructor Documentation	124
6.32.2.1	ScriptFunction	124
6.32.2.2	~ScriptFunction	124
6.32.3	Member Function Documentation	124
6.32.3.1	evaluate	124
6.32.3.2	mml	124
6.33	SemSolver::ScriptFunction< Point< d, X >, Vector< Y > > Class Template Reference	125
6.33.1	Detailed Description	125
6.33.2	Constructor & Destructor Documentation	126
6.33.2.1	ScriptFunction	126
6.33.2.2	~ScriptFunction	126
6.33.3	Member Function Documentation	126
6.33.3.1	evaluate	126
6.33.3.2	mml	126
6.34	SemSolver::ScriptFunction< Point< d, X >, Y > Class Template Reference	127
6.34.1	Detailed Description	127
6.34.2	Constructor & Destructor Documentation	127
6.34.2.1	ScriptFunction	127
6.34.2.2	~ScriptFunction	128
6.34.3	Member Function Documentation	128
6.34.3.1	evaluate	128
6.34.3.2	mml	128
6.35	SemSolver::PSLG< X >::Segment Struct Reference	128
6.35.1	Detailed Description	129
6.35.2	Member Data Documentation	129
6.35.2.1	marker	129
6.35.2.2	number	129
6.35.2.3	source	129
6.35.2.4	target	129
6.36	SemSolver::Segment< 2, X > Class Template Reference	129
6.36.1	Detailed Description	130
6.36.2	Constructor & Destructor Documentation	130
6.36.2.1	Segment	130
6.36.2.2	Segment	130
6.36.3	Member Function Documentation	130
6.36.3.1	cgal	130
6.36.3.2	intersect	130
6.36.3.3	intersectInteriorly	131
6.37	SemSolver::SegmentsMap< d, X > Class Template Reference	131
6.37.1	Detailed Description	132
6.37.2	Member Typedef Documentation	132
6.37.2.1	ConstIterator	132
6.37.2.2	Iterator	132
6.37.3	Constructor & Destructor Documentation	132
6.37.3.1	SegmentsMap	132
6.37.4	Member Function Documentation	133
6.37.4.1	add	133
6.37.4.2	contains	133

6.37.4.3	has	133
6.37.4.4	haveOn	133
6.37.4.5	id	133
6.37.4.6	insert	133
6.37.4.7	intersect	134
6.37.4.8	intersectInteriorly	134
6.37.4.9	isConsistentWith	134
6.37.4.10	modify	134
6.37.4.11	remove	134
6.37.4.12	segment	135
6.37.4.13	segmentFrom	135
6.37.4.14	segments	135
6.37.4.15	segmentTo	135
6.38	SemSolver::SemFunction< d, X > Class Template Reference	135
6.38.1	Detailed Description	136
6.39	SemSolver::SemFunction< 2, X > Class Template Reference	136
6.39.1	Detailed Description	137
6.39.2	Constructor & Destructor Documentation	137
6.39.2.1	SemFunction	137
6.39.2.2	SemFunction	137
6.39.3	Member Function Documentation	137
6.39.3.1	evaluate	137
6.39.3.2	evaluateRestrictionGradient	138
6.39.3.3	map	138
6.39.3.4	polynomial	138
6.39.3.5	setPolynomialComponent	138
6.40	SemSolver::SemGeometry< d, X > Class Template Reference	138
6.40.1	Detailed Description	138
6.41	SemSolver::SemGeometry< 2, X > Class Template Reference	139
6.41.1	Detailed Description	139
6.41.2	Member Function Documentation	140
6.41.2.1	contains	140
6.41.2.2	domain	140
6.41.2.3	setDomain	140
6.41.2.4	setSubDomains	140
6.41.2.5	subDomains	140
6.42	SemSolver::SemParameters< X > Class Template Reference	140
6.42.1	Detailed Description	141
6.42.2	Constructor & Destructor Documentation	141
6.42.2.1	SemParameters	141
6.42.2.2	SemParameters	141
6.42.3	Member Function Documentation	142
6.42.3.1	degree	142
6.42.3.2	penalty	142
6.42.3.3	setDegree	142
6.42.3.4	setPenalty	142
6.42.3.5	setTolerance	142
6.42.3.6	tolerance	142
6.43	SemSolver::SemSpace< d, X > Class Template Reference	143
6.43.1	Detailed Description	143

6.44	SemSolver::SemSpace< 2, X > Class Template Reference	143
6.44.1	Detailed Description	145
6.44.2	Member Typedef Documentation	146
6.44.2.1	BordersMap	146
6.44.2.2	BordersVector	146
6.44.2.3	ElementConstIterator	146
6.44.2.4	ElementsMap	146
6.44.2.5	Index2Order	146
6.44.2.6	Index3Order	146
6.44.2.7	NodeConstIterator	146
6.44.2.8	NodesMap	147
6.44.2.9	NodesVector	147
6.44.2.10	SemFunctionsVector	147
6.44.2.11	SubDomain	147
6.44.2.12	WeightsMap	147
6.44.3	Constructor & Destructor Documentation	147
6.44.3.1	SemSpace	147
6.44.4	Member Function Documentation	147
6.44.4.1	baseFunction	147
6.44.4.2	border	148
6.44.4.3	borderId	148
6.44.4.4	borderIndex	148
6.44.4.5	borderNode	148
6.44.4.6	borders	148
6.44.4.7	borderSubDomainIndex	148
6.44.4.8	borderWeight	148
6.44.4.9	degree	149
6.44.4.10	node	149
6.44.4.11	nodes	149
6.44.4.12	scalarProduct	149
6.44.4.13	subDomainIndex	149
6.44.4.14	subDomainNode	149
6.44.4.15	subDomains	149
6.44.4.16	subDomainWeight	150
6.44.5	Member Data Documentation	150
6.44.5.1	_geometry	150
6.44.5.2	_parameters	150
6.45	SemSolver::Vector< X > Class Template Reference	150
6.45.1	Detailed Description	151
6.45.2	Constructor & Destructor Documentation	151
6.45.2.1	Vector	151
6.45.2.2	Vector	151
6.45.2.3	Vector	151
6.45.2.4	Vector	151
6.45.2.5	Vector	151
6.45.3	Member Function Documentation	151
6.45.3.1	rows	151
6.46	SemSolver::PSLG< X >::Vertex Struct Reference	152
6.46.1	Detailed Description	152
6.46.2	Member Data Documentation	152

6.46.2.1	attributes	152
6.46.2.2	marker	152
6.46.2.3	number	152
6.46.2.4	x	153
6.46.2.5	y	153
7	File Documentation	155
7.1	archive.hpp File Reference	155
7.2	archive.hpp	155
7.3	bilineartransformation.hpp File Reference	157
7.4	bilineartransformation.hpp	157
7.5	boundaryconditions.hpp File Reference	162
7.6	boundaryconditions.hpp	162
7.7	boundaryconditions.hpp File Reference	169
7.8	boundaryconditions.hpp	169
7.9	buildsolution.hpp File Reference	171
7.10	buildsolution.hpp	172
7.11	choleskysolve.hpp File Reference	172
7.12	choleskysolve.hpp	173
7.13	computealgebraicsystem.hpp File Reference	173
7.14	computealgebraicsystem.hpp	174
7.15	computebordermatrix.hpp File Reference	176
7.16	computebordermatrix.hpp	176
7.17	computebordervector.hpp File Reference	177
7.18	computebordervector.hpp	178
7.19	computeconvectionmatrix.hpp File Reference	179
7.20	computeconvectionmatrix.hpp	180
7.21	computediffusionmatrix.hpp File Reference	181
7.22	computediffusionmatrix.hpp	182
7.23	computeforcingvector.hpp File Reference	183
7.24	computeforcingvector.hpp	183
7.25	computeplotdata.hpp File Reference	184
7.26	computeplotdata.hpp	185
7.27	computepolygonationfrompslg.hpp File Reference	186
7.28	computepolygonationfrompslg.hpp	186
7.29	computepolygonwithholesfrompslg.hpp File Reference	188
7.30	computepolygonwithholesfrompslg.hpp	189
7.31	computereactionmatrix.hpp File Reference	193
7.32	computereactionmatrix.hpp	194
7.33	computesolutionhull.hpp File Reference	194
7.34	computesolutionhull.hpp	195
7.35	diffusionconvectionreactionequation.hpp File Reference	196
7.36	diffusionconvectionreactionequation.hpp	196
7.37	equation.hpp File Reference	201
7.38	equation.hpp	201
7.39	equation.hpp File Reference	202
7.40	equation.hpp	202
7.41	function.hpp File Reference	205
7.42	function.hpp	205
7.43	geometry.hpp File Reference	206

7.44	geometry.hpp	207
7.45	hilbertspace.hpp File Reference	208
7.46	hilbertspace.hpp	208
7.47	homeomorphism.hpp File Reference	209
7.48	homeomorphism.hpp	210
7.49	lusolve.hpp File Reference	210
7.50	lusolve.hpp	211
7.51	matrix.hpp File Reference	211
7.52	matrix.hpp	212
7.53	multiindex.hpp File Reference	214
7.54	multiindex.hpp	215
7.55	nextnonemptlinevalues.hpp File Reference	216
7.56	nextnonemptlinevalues.hpp	216
7.57	parameters.hpp File Reference	217
7.58	parameters.hpp	217
7.59	point.hpp File Reference	219
7.60	point.hpp	219
7.61	pointsbimap.hpp File Reference	220
7.62	pointsbimap.hpp	221
7.63	pointsmmap.hpp File Reference	225
7.64	pointsmmap.hpp	225
7.65	pointssset.hpp File Reference	230
7.66	pointssset.hpp	230
7.67	polygon.hpp File Reference	234
7.68	polygon.hpp	235
7.69	polygonation.hpp File Reference	236
7.70	polygonation.hpp	236
7.71	polygonwithholes.hpp File Reference	242
7.72	polygonwithholes.hpp	242
7.73	polynomial.hpp File Reference	243
7.74	polynomial.hpp	243
7.75	polynomialfunction.hpp File Reference	248
7.76	polynomialfunction.hpp	249
7.77	problem.hpp File Reference	250
7.78	problem.hpp	250
7.79	pslg.hpp File Reference	253
7.80	pslg.hpp	254
7.81	pslg.hpp File Reference	262
7.82	pslg.hpp	263
7.83	qrsolve.hpp File Reference	269
7.84	qrsolve.hpp	269
7.85	scriptfunction.hpp File Reference	270
7.86	scriptfunction.hpp	271
7.87	segment.hpp File Reference	277
7.88	segment.hpp	277
7.89	segmentsmap.hpp File Reference	279
7.90	segmentsmap.hpp	279
7.91	semfunction.hpp File Reference	282
7.92	semfunction.hpp	283
7.93	semgeometry.hpp File Reference	285

7.94	semgeometry.hpp	285
7.95	semparameters.hpp File Reference	286
7.96	semparameters.hpp	286
7.97	semspace.hpp File Reference	287
7.98	semspace.hpp	288
7.99	sequence.hpp File Reference	300
7.100	sequence.hpp	300
7.101	sequenceslist.hpp File Reference	300
7.102	sequenceslist.hpp	301
7.103	subdomains.hpp File Reference	301
7.104	subdomains.hpp	302
7.105	vector.hpp File Reference	303
7.106	vector.hpp	304
7.107	workspace.hpp File Reference	305
7.108	workspace.hpp	307

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

SemSolver (Project main namespace)	9
SemSolver::Assembler (Assembler namespace)	14
SemSolver::IO (Namespace for Input/Output operations on SemSolver Classes)	16
SemSolver::PostProcessor	21
SemSolver::PreProcessor (PreProcessor namespace)	22
SemSolver::Solver (Solver namespace)	23

Chapter 2

Class Index

2.1 Class Hierarchy

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

SemSolver::IO::Archive	25
SemSolver::BoundaryConditions< d, X >	32
SemSolver::BoundaryConditions< 2, X >	33
SemSolver::HilbertSpace< Function, X >::Element	48
SemSolver::Polygonation< 2, X >::Element	49
SemSolver::Equation< d, X >	56
SemSolver::DiffusionConvectionReactionEquation< d, X >	39
SemSolver::Equation< 2, X >	56
SemSolver::DiffusionConvectionReactionEquation< 2, X >	44
SemSolver::Function< X, Y >	58
SemSolver::Homeomorphism< Point< 2, X >, Point< 2, X > >	63
SemSolver::BilinearTransformation< X >	27
SemSolver::Homeomorphism< X, Y >	63
SemSolver::Function< Point< 2, X >, Point< 2, X > >	58
SemSolver::Function< Point< 2, X >, Vector< Y > >	58
SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > >	121
SemSolver::Function< Point< 2, X >, X >	58
SemSolver::SemFunction< 2, X >	136
SemSolver::SemSpace< 2, X >::Element	54
SemSolver::Function< Point< 2, X >, Y >	58
SemSolver::ScriptFunction< Point< 2, X >, Y >	123
SemSolver::Function< Point< d, X >, Vector< Y > >	58
SemSolver::ScriptFunction< Point< d, X >, Vector< Y > >	125
SemSolver::Function< Point< d, X >, X >	58
SemSolver::PolynomialFunction< d, X >	109
SemSolver::SemFunction< d, X >	135
SemSolver::Function< Point< d, X >, Y >	58

SemSolver::ScriptFunction< Point< d, X >, Y >	127
SemSolver::HilbertSpace< Function, X >	60
SemSolver::HilbertSpace< SemFunction< 2, X >, X >	60
SemSolver::SemSpace< 2, X >	143
SemSolver::HilbertSpace< SemFunction< d, X >, X >	60
SemSolver::SemSpace< d, X >	143
SemSolver::PSLG< X >::Hole	63
SemSolver::Segment< 2, X >::less	65
SemSolver::MultiIndex< N >::less	66
SemSolver::Matrix< X >	66
SemSolver::MultiIndex< N >	69
SemSolver::SemSpace< 2, X >::Node	71
SemSolver::Point< 2, X >	73
SemSolver::PointsBimap< 2, X >	74
SemSolver::PointsMap< 2, X, Y >	81
SemSolver::PointsSet< 2, X >	89
SemSolver::Polygon< 2, X >	96
SemSolver::Polygonation< 2, X >	97
SemSolver::PolygonWithHoles< 2, X >	100
SemSolver::Polynomial< X >	101
SemSolver::Problem< d, X >	112
SemSolver::PSLG< X >	115
SemSolver::PSLG< X >::Segment	128
SemSolver::Segment< 2, X >	129
SemSolver::SegmentsMap< d, X >	131
SemSolver::SemGeometry< d, X >	138
SemSolver::SemGeometry< 2, X >	139
SemSolver::SemParameters< X >	140
SemSolver::Vector< X >	150
SemSolver::PSLG< X >::Vertex	152

Chapter 3

Class Index

3.1 Class List

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

SemSolver::IO::Archive (Class for handling tar uncompressed archives) . . .	25
SemSolver::BilinearTransformation< X > (Class representing a bilinear transformation of 2D euclidean space)	27
SemSolver::BoundaryConditions< d, X > (Class for handling boundary conditions on a SemGeometry)	32
SemSolver::BoundaryConditions< 2, X > (Class for handling boundary conditions on a 2D SemGeometry)	33
SemSolver::DiffusionConvectionReactionEquation< d, X > (Class for handling Diffusion-Convection-Reaction steady equation)	39
SemSolver::DiffusionConvectionReactionEquation< 2, X > (Class for handling Diffusion-Convection-Reaction steady equation on 2D spaces)	44
SemSolver::HilbertSpace< Function, X >::Element (Class for handling space elements as Fourier coefficients)	48
SemSolver::Polygonation< 2, X >::Element (A Polygonation element) . . .	49
SemSolver::SemSpace< 2, X >::Element (Class for handling members of the space as Fourier coefficients)	54
SemSolver::Equation< d, X > (Virtual class for handling general equation) .	56
SemSolver::Function< X, Y > (Prototype class for mathematical functions : $X \rightarrow Y$)	58
SemSolver::HilbertSpace< Function, X > (Prototype class for handling the concept of Hilbert Space)	60
SemSolver::PSLG< X >::Hole (PSLG Hole struct)	63
SemSolver::Homeomorphism< X, Y > (Prototype class for mathematical homemorphism : $X \rightarrow Y$)	63
SemSolver::Segment< 2, X >::less	65
SemSolver::MultiIndex< N >::less (Partial order)	66
SemSolver::Matrix< X > (Class for handling mathematical matrices)	66
SemSolver::MultiIndex< N > (Class multi-index notation)	69

SemSolver::SemSpace< 2, X >::Node	71
SemSolver::Point< 2, X > (Class for handling 2D euclidean points)	73
SemSolver::PointsBimap< 2, X > (Class for handling bi-directional maps between 2D Points and Integers)	74
SemSolver::PointsMap< 2, X, Y > (Class for handling maps with 2D Point as KeyType)	81
SemSolver::PointsSet< 2, X > (Class for handling sets of 2D Points)	89
SemSolver::Polygon< 2, X > (Class for handling 2D polygons)	96
SemSolver::Polygonation< 2, X > (Class for handling 2D polygonations)	97
SemSolver::PolygonWithHoles< 2, X > (Class for handling 2D polygons with holes)	100
SemSolver::Polynomial< X > (Class for handling the mathematical concept of polynomials over X)	101
SemSolver::PolynomialFunction< d, X > (Class for handling polynomial separable functions)	109
SemSolver::Problem< d, X > (Class for handling a mathematical problem given by an equation, boundary conditions, geometry and parameters)	112
SemSolver::PSLG< X > (Class for handling Planar Straight Line Graphs)	115
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)	121
SemSolver::ScriptFunction< Point< 2, X >, Y > (Class for handling function from 2D Euclidean space X^2 to scalar space Y defined in scripts)	123
SemSolver::ScriptFunction< Point< d, X >, Vector< Y > > (Class for handling function from Euclidean space X^d to Vectorial space Y^n defined inscripts)	125
SemSolver::ScriptFunction< Point< d, X >, Y > (Class for handling function from Euclidean space X^d to scalar space Y defined in scripts)	127
SemSolver::PSLG< X >::Segment (PSLG Segment struct)	128
SemSolver::Segment< 2, X >	129
SemSolver::SegmentsMap< d, X >	131
SemSolver::SemFunction< d, X > (Class for handling spectral elements functions)	135
SemSolver::SemFunction< 2, X > (Class for handling 2D spectral elements functions)	136
SemSolver::SemGeometry< d, X > (Class for describing the geometry of a SemProblem)	138
SemSolver::SemGeometry< 2, X >	139
SemSolver::SemParameters< X >	140
SemSolver::SemSpace< d, X >	143
SemSolver::SemSpace< 2, X >	143
SemSolver::Vector< X >	150
SemSolver::PSLG< X >::Vertex (PSLG Vertex struct)	152

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

archive.hpp	155
bilineartransformation.hpp	157
boundaryconditions.hpp	162
IO/boundaryconditions.hpp	169
buildsolution.hpp	171
choleskysolve.hpp	172
computealgebraicsystem.hpp	173
computebordermatrix.hpp	176
computebordervector.hpp	177
computeconvectionmatrix.hpp	179
computediffusionmatrix.hpp	181
computeforcingvector.hpp	183
computeplotdata.hpp	184
computepolygonationfrompslg.hpp	186
computepolygonwithholesfrompslg.hpp	188
computereactionmatrix.hpp	193
computesolutionhull.hpp	194
diffusionconvectionreactionequation.hpp	196
equation.hpp	201
IO/equation.hpp	202
function.hpp	205
geometry.hpp	206
hilbertspace.hpp	208
homeomorphism.hpp	209
lusolve.hpp	210
matrix.hpp	211
multiindex.hpp	214
nextnonemptilinevalues.hpp	216
parameters.hpp	217

point.hpp	219
pointsbimap.hpp	220
pointsmap.hpp	225
pointsset.hpp	230
polygon.hpp	234
polygonation.hpp	236
polygonwithholes.hpp	242
polynomial.hpp	243
polynomialfunction.hpp	248
problem.hpp	250
IO/pslg.hpp	253
pslg.hpp	262
qrsolve.hpp	269
scriptfunction.hpp	270
segment.hpp	277
segmentsmap.hpp	279
semfunction.hpp	282
semgeometry.hpp	285
semparameters.hpp	286
semspace.hpp	287
sequence.hpp	300
sequenceslist.hpp	300
subdomains.hpp	301
vector.hpp	303
workspace.hpp	305

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 handling 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 handling 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 in scripts.
- 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

<i>d</i>	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

<i>mat1</i>	First matrix
<i>mat2</i>	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

<i>mat1</i>	First matrix
<i>mat2</i>	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 &penalty, 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 &penalty, 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 construction 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 & penalty, 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 penalty 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 & penalty, 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 penalty 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](#) from 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 > ¶meters)
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 ¶meters)
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](#) from 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> &polygonaion)`
- `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 geometry 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

<i>file</i>	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

<i>name</i>	Name of the entry
<i>file</i>	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

<i>name</i>	Name of the entry
<i>value</i>	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

<i>file</i>	Pointer to the file where to extract entry data
<i>name</i>	Name of the entry

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

Get an entry

Parameters

<i>name</i>	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:

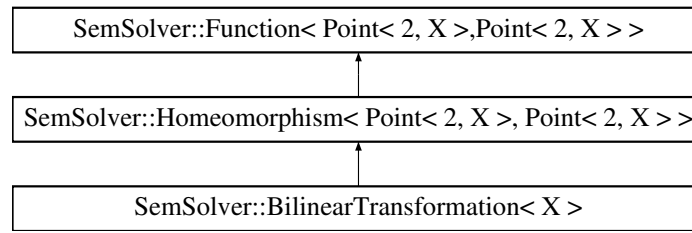
- [archive.hpp](#)

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 transformation.
- Polygon< 2, X > const & [omega](#) () const
Get the reference to Ω .
- X const & [tolerance](#) () const
Get the tolerance used in evaluating the inverse transformation.
- Point< 2, X > [evaluate](#) (Point< 2, X > const &point) const
Evaluate the direct transformation at a point.
- Point< 2, X > [evaluateInverse](#) (Point< 2, X > const &point) const
Evaluate the inverse transformation 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 transformation of 2D euclidean space.

Parameters

<code>X</code>	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 <code>int</code> 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

<i>omega</i>	The polygon to be mapped onto $\hat{\Omega}$. It must be a simple convex counterclockwise-oriented quadrangle
<i>tolerance</i>	The value below which numbers are treated 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 transformation at a point.

Parameters

<i>point</i>	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 transformation at a point.

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

Parameters

<i>point</i>	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

<i>point</i>	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

<i>point</i>	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

<i>omega</i>	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

<i>tolerance</i>	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

<i>d</i>	Dimension of the space on which the geometry lives
<i>X</i>	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

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

<i>conditions</i>	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

<i>index</i>	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

<i>index</i>	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

<i>index</i>	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

<i>index</i>	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

<i>index</i>	of the border
--------------	---------------

Returns

Pointer to a scalar function

Definition at line 261 of file [boundaryconditions.hpp](#).

6.5 SemSolver::DiffusionConvectionReactionEquation< d, X > Class Template Reference 39

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

<i>index</i>	Index of the border
<i>type</i>	Type of the border
<i>f1</i>	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
<i>f2</i>	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:

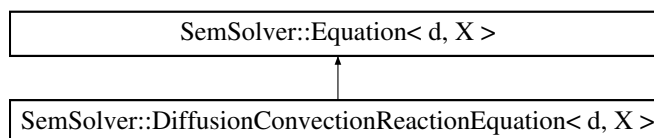
- [boundaryconditions.hpp](#)

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 available equation types.

Public Member Functions

- [DiffusionConvectionReactionEquation](#) ()
Default constructor Construct an empty 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

<i>d</i>	Dimension of the space on which the equation is defined
<i>X</i>	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 <code>int</code> does not fulfil 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 available 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 empty 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

6.5 SemSolver::DiffusionConvectionReactionEquation< d, X > Class Template Reference 43

<i>convection</i>	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

<i>diffusion</i>	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

<i>forcing</i>	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

<i>reaction</i>	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:

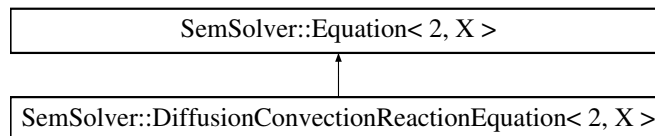
- [diffusionconvectionreactionequation.hpp](#)

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 available equation types.

Public Member Functions

- [DiffusionConvectionReactionEquation](#) ()
Default constructor Construct an empty equation.
- [~DiffusionConvectionReactionEquation](#) ()
Destructor.
- [Type](#) [type](#) () const
Get the equation type.
- QString [mml](#) () const
Get the equation in Mathematical Markup Language format.
- 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 available 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 empty 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.

6.6 SemSolver::DiffusionConvectionReactionEquation< 2, X > Class Template Reference 47

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 Language 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

<i>convection</i>	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

<i>diffusion</i>	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

<i>forcing</i>	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

<i>reaction</i>	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 Element 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 neighbour id.
- std::vector< int >::const_iterator [neighboursEnd](#) () const
Get iterator to last neighbour 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

<i>geometry</i>	The geometry of the element
<i>neighbours</i>	Vector of neighbour element ids in counterclockwise 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

<i>point</i>	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

<i>index</i>	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 neighbour 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 neighbour 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

<i>first</i>	Pointer to the first Point of the sequence
<i>last</i>	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

<i>index</i>	Position of the neighbour
<i>id</i>	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

<i>index</i>	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

<i>vertex</i>	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:

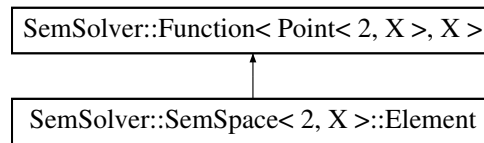
- [polygonation.hpp](#)

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:

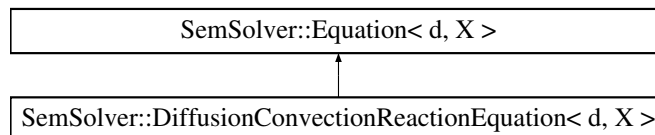
- [semSPACE.hpp](#)

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 available 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

<i>d</i>	Dimension of the space on which the equation is defined
<i>X</i>	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 available 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:

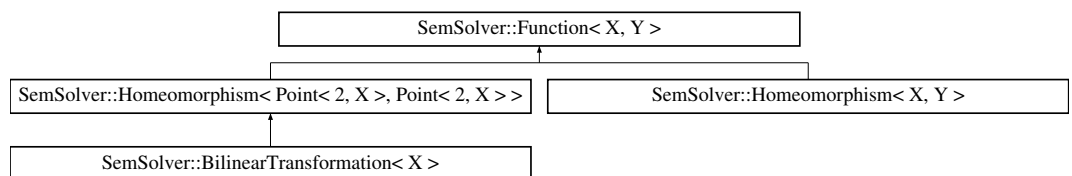
- [equation.hpp](#)

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 : $X \rightarrow Y$.

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

<i>index</i>	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

<i>element</i>	
----------------	--

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:

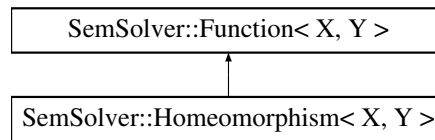
- [pslg.hpp](#)

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

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

```
#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 homomorphism : $X \rightarrow Y$.

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

<i>rows</i>	Number of rows
<i>columns</i>	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

<i>rows</i>	Number of rows
<i>columns</i>	Number of columns
<i>value</i>	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

<i>N</i>	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

<i>sub_indices</i>	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

<i>index</i>	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

<i>index</i>	The component index
<i>sub_index</i>	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

<i>index</i>	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 <code>int</code> does not fulfil 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	Abcissa
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

<code>X</code>	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 <code>int</code> does not fulfil 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

<i>tol</i>	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 >::eraseld (const MappedType & y)` [inline]

Erase a point-id entry if exists.

Parameters

<i>y</i>	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

<i>x</i>	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

<i>id</i>	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

<i>id</i>	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

<i>x</i>	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

<i>x</i>	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

<i>y</i>	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

<i>x</i>	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

<i>segment</i>	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

<code>x</code>	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

<code>x</code>	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

<code>x</code>	The point corresponding to the id to be modified. It must exist
<code>y</code>	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

<code>y</code>	The id corresponding to the point to be modified. It must exist
<code>x</code>	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

<code>y</code>	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 constant.
- 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 fulfil 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

<i>tol</i>	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 constant.

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

<i>position</i>	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

<i>x</i>	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

<i>first</i>	Iterator to the first entry to be removed
<i>last</i>	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

<i>x</i>	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

<i>x</i>	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

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

Returns

the iterator to the point inserted

Definition at line 256 of file [pointsmmap.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

<i>first</i>	iterator to the first pair
<i>last</i>	iterator to the last pair

Definition at line 282 of file [pointsmmap.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

<i>x</i>	the point-mapped_value pair to be inserted
----------	--

Returns

the iterator to the point inserted

Definition at line 233 of file [pointsmmap.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

<i>x</i>	the point value to be inserted
<i>y</i>	the mapped value to be inserted

Returns

the iterator to the point inserted

Definition at line 244 of file [pointsmmap.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 [pointsmmap.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

<i>x</i>	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 [pointsmmap.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 [pointsmmap.hpp](#).

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

- [pointsmmap.hpp](#)

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

Class for handling sets of 2D Points.

```
#include <pointssset.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 constant.
- 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 fulfill 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

<i>tol</i>	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 content.

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

<i>position</i>	Iterator to the element to be removed
-----------------	---------------------------------------

Definition at line 164 of file [pointssset.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

<i>x</i>	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 [pointssset.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

<i>first</i>	Iterator to the first entry to be removed
<i>last</i>	Iterator to the position just beyond the last element to be removed

Definition at line 183 of file [pointssset.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 [pointssset.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

<i>x</i>	The point value to search
----------	---------------------------

Returns

ConstIterator to the entry if found, to end otherwise

Definition at line 193 of file [pointssset.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

<i>x</i>	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

<i>first</i>	iterator to the first pair
<i>last</i>	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 exist otherwise do nothing.

Parameters

<i>x</i>	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 exist otherwise do nothing.

Parameters

<i>x</i>	the point to be inserted
<i>it</i>	iterator to the position guess where to insert new entry

Returns

the iterator to the point inserted

Definition at line 241 of file [pointset.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 [pointset.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

<i>x</i>	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 [pointset.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 [pointset.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 fulfil 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

<i>first</i>	Pointer to the first Point of the sequence
<i>last</i>	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

<i>point</i>	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 quadrangulation.

- 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

<i>polygon</i>	The geometry of the element
<i>neighbours</i>	Vector of neighbour element ids in counterclockwise 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

<i>index</i>	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

<i>point</i>	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 quadrangulation.

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 handling 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 \(std::vector< X > const &coefficients\)](#)
Construct 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 °ree)`
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 handling 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

<i>scalar</i>	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]

Construct a polynomial from its coefficients.

Parameters

<i>coefficients</i>	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

<i>poly</i>	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

<i>order</i>	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

<i>poly</i>	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

<i>poly</i>	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

<i>poly</i>	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

<i>poly</i>	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

<i>poly</i>	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

<i>poly</i>	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

<i>poly</i>	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

<i>poly</i>	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

<i>poly</i>	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

<i>poly</i>	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

<i>poly</i>	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

<i>order</i>	Must be non negative and not greater than degree
<i>coefficient</i>	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

<i>degree</i>	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:

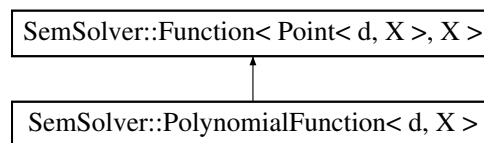
- [polynomial.hpp](#)

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)
Assignment 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

<i>d</i>	Dimension of the space
<i>X</i>	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 fulfill 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]`

Assignment 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

<i>index</i>	The index of the component on which 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 which 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 handling 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 handling 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

<i>index</i>	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

<i>index</i>	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

<i>index</i>	The hole position
<i>number</i>	The hole number
<i>x</i>	Abscissa
<i>y</i>	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

<i>number</i>	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

<i>number</i>	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

<i>number</i>	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

<i>number</i>	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

<i>number</i>	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

<i>index</i>	The segment position
<i>number</i>	The segment number
<i>source</i>	First end-point number
<i>target</i>	Second end-point number
<i>marker</i>	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

<i>index</i>	The vertex position
<i>number</i>	The vertex number
<i>x</i>	Abscissa
<i>y</i>	Ordinate
<i>attributes</i>	Attributes array
<i>marker</i>	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

<i>index</i>	The vertex position
--------------	---------------------

Returns

The vertex

Definition at line 227 of file [pslg.hpp](#).

6.31 SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > > Class Template Reference 121

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:

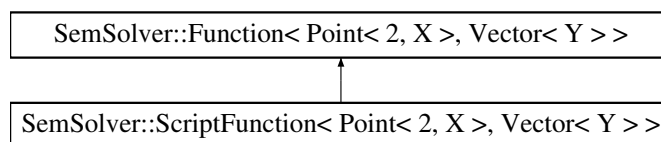
- [pslg.hpp](#)

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 specify 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

<i>strings</i>	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 redefined, 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.

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

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:

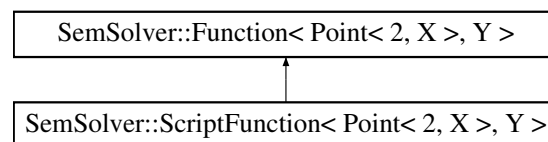
- [scriptfunction.hpp](#)

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 specify 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

<i>string</i>	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 redefined, 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:

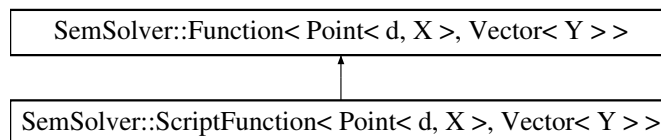
- [scriptfunction.hpp](#)

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 specify 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

<i>strings</i>	Function component definitions in ECMA Script language. Variables are assumed to be x_0, x_1, \dots, x_{d-1} . To the function is associated the ECMA Script "function f0(x_0, x_1, \dots, x_{d-1}) { return string[0] }, function f1(x_0, x_1, \dots, x_{d-1}) { return string[1] }, ..., function fn(x_0, x_1, \dots, x_{d-1}) { return string[n] }". For ease of use, mathematical function of Math object are redefined, 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:

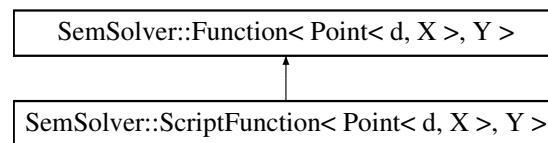
- [scriptfunction.hpp](#)

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 specify 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

<i>string</i>	Function definition in ECMA Script language. Variables are assumed to be x_0, x_1, \dots, x_{d-1} . To the function associated the ECMA Script "function $f(x_0, x_1, \dots, x_{d-1}) \{ \text{return string} \}$ ". For ease of use, mathematical function of Math object are redefined, in this way $\sin(x)$ can be used instead of <code>Math.sin(x)</code>
---------------	---

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 <segmentmap.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

<i>segment</i>	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 intersecting 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 intersecting 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 consistent 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

<i>id</i>	The key to find, it is assumed to exist in map
<i>segment</i>	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:

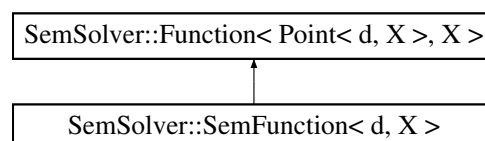
- [segmentsmap.hpp](#)

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:

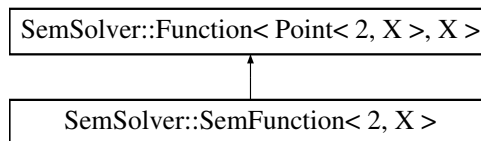
- [semfunction.hpp](#)

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, PolynomialFunction-Vector 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 transformation 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 PolynomialFunction vector and a BilinearTransformation 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 fulfil 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 °ree, X const &tolerance, X const &penalty)
Construct Parameters from degree, tolerance and penalty values.
- int const & [degree](#) () const
Access degree parameter.
- X const & [tolerance](#) () const
Access tolerance parameter.
- X const & [penalty](#) () const
Access penalty parameter.
- void [setDegree](#) (const int &d)
Set degree parameter.
- void [setTolerance](#) (const X &t)
Set tolerance parameter.
- void [setPenalty](#) (const X &p)
Set penalty 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 penalty 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 & penalty)` [inline]

Construct Parameters from degree, tolerance and penalty 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 >::penalty ()
const [inline]`

Access penalty 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 >::setPenalty (const
X & p) [inline]`

Set penalty 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](#).

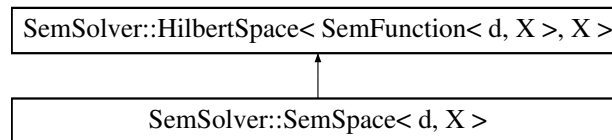
The documentation for this class was generated from the following file:

- [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

<i>d</i>	Dimension of the domain
<i>X</i>	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 fulfil 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](#).

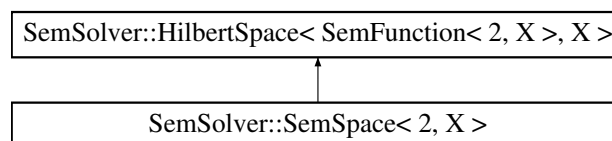
The documentation for this class was generated from the following file:

- [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 ¶meters)
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 corresponding 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 corresponding 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 corresponding 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::ArrayID.
- [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

```
00001 #ifndef IO_ARCHIVE_HPP
```

```
00002 #define IO_ARCHIVE_HPP
00003
00004 #include <QFile>
00005 #include <QString>
00006 #include <QStringList>
00007 #include <QTemporaryFile>
00008 #include <QTextStream>
00009
00010 #include <SemSolver/IO/carchive>
00011
00012 namespace SemSolver
00013 {
00014     namespace IO
00015     {
00016         class Archive
00017         {
00018             enum Status
00019             {
00020                 CLOSED,
00021                 OPENREAD,
00022                 OPENWRITE
00023             };
00024
00025             archive::archive *archive;
00026             QFile *file;
00027             Status status;
00028
00029         public:
00030             Archive(QFile *file);
00031
00032             bool openRead();
00033
00034             bool openWrite();
00035
00036             bool closeRead();
00037
00038             bool closeWrite();
00039
00040             QStringList entries();
00041
00042             template<class T>
00043             bool addValue(T const &value,
00044                           QString const &name)
00045             {
00046                 #if SEMDEBUG
00047                     if(status!=OPENWRITE)
00048                         qFatal("You must openWrite archive before adding entries");
00049                 #endif
00050                 if(!file->exists())
00051                     return false;
00052                 QTemporaryFile *temp_file = new QTemporaryFile();
00053                 if(!temp_file->open(QIODevice::WriteOnly | QIODevice::Text))
00054                     return false;
00055                 QTextStream out(temp_file);
00056                 out << value;
00057                 temp_file->close();
00058                 addFile(temp_file, name);
00059                 delete temp_file;
00060                 return true;
00061             };
00062
00063             bool addFile(QFile *file);
00064
00065             bool addFile(QFile *file);
```

```

00080
00084         bool addFile(QFile *file,
00085                      QString const &name);
00086
00089         bool extractFile(QString const &name);
00090
00094         bool extractFile(QString const &name, QFile *file);
00095     };
00096 };
00097 };
00098
00099 #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 transformation of 2D euclidean space.

Namespaces

- namespace [SemSolver](#)
Project main namespace.

7.4 bilineartransformation.hpp

```

00001 #ifndef BILINEARTRANSFORMATION_HPP
00002 #define BILINEARTRANSFORMATION_HPP
00003
00004 namespace SemSolver
00005 {
00006     template<class X>
00007     class BilinearTransformation;
00008 };
00009
00010 #include <cmath>
00011
00012 #include <SemSolver/homeomorphism.hpp>
00013 #include <SemSolver/matrix.hpp>

```

```

00014 #include <SemSolver/point.hpp>
00015 #include <SemSolver/polygon.hpp>
00016
00017 namespace SemSolver
00018 {
00019     template<class X>
00020     class BilinearTransformation
00021     : public Homeomorphism< Point<2,X>, Point<2,X> >
00022     {
00023     // domain
00024     Polygon<2,X> _omega;
00025
00026     // coefficients
00027     X _Px, _Qx, _Rx, _Sx;
00028     X _Py, _Qy, _Ry, _Sy;
00029     X _PQ, _PR, _PS, _QR, _QS, _RS;
00030
00031     // tolerance for inverse evaluation
00032     X _tolerance;
00033
00034     public:
00035
00036     BilinearTransformation();
00037
00038     BilinearTransformation(Point<2,X> const &A,
00039                             Point<2,X> const &B,
00040                             Point<2,X> const &C,
00041                             Point<2,X> const &D);
00042
00043     BilinearTransformation(Polygon<2,X> const &omega,
00044                             X const &tolerance);
00045
00046     ~BilinearTransformation();
00047
00048     void setOmega(Polygon<2,X> const &omega);
00049
00050     void setTolerance(X const &tolerance);
00051
00052     inline Polygon<2,X> const &omega() const;
00053
00054     inline X const &tolerance() const;
00055
00056     Point<2,X> evaluate(Point<2,X> const &point) const;
00057
00058     Point<2,X> evaluateInverse(Point<2,X> const &point) const;
00059
00060     inline X evaluateJacobianDeterminant(Point<2,X> const &point) const;
00061
00062     Matrix<double>
00063         evaluateTransposeInverseJacobian(Point<2,X> const &point) const;
00064     };
00065 };
00066
00067 template<class X>
00068 SemSolver::BilinearTransformation<X>::BilinearTransformation()
00069 {
00070     _Px = 0., _Py = 0.;
00071     _Qx = -1., _Qy = 0.;
00072     _Rx = 0., _Ry = -1.;
00073     _Sx = 0., _Sy = 0.;
00074     _PQ = 0., _PR = 0.;

```



```

00083     _PS = 0., _QR = 1.;
00084     _QS = 0., _RS = 0.;
00085     _tolerance = 0;
00086 };
00087
00092 template<class X>
00093 SemSolver::BilinearTransformation<X>::BilinearTransformation(Point<2,X> const &A,
00094
00095                                     Point<2,X> const &B,
00096                                     Point<2,X> const &C,
00097                                     Point<2,X> const &D)
00097 {
00098     _Px = (A.x()+B.x()+C.x()+D.x())/4.;
00099     _Py = (A.y()+B.y()+C.y()+D.y())/4.;
00100     _Qx = (A.x()-B.x()-C.x()+D.x())/4.;
00101     _Qy = (A.y()-B.y()-C.y()+D.y())/4.;
00102     _Rx = (A.x()+B.x()-C.x()-D.x())/4.;
00103     _Ry = (A.y()+B.y()-C.y()-D.y())/4.;
00104     _Sx = (A.x()-B.x()+C.x()-D.x())/4.;
00105     _Sy = (A.y()-B.y()+C.y()-D.y())/4.;
00106
00107     _PQ = _Px*_Qy - _Py*_Qx;
00108     _PR = _Px*_Ry - _Py*_Rx;
00109     _PS = _Px*_Sy - _Py*_Sx;
00110     _QR = _Qx*_Ry - _Qy*_Rx;
00111     _QS = _Qx*_Sy - _Qy*_Sx;
00112     _RS = _Rx*_Sy - _Ry*_Sx;
00113
00114     _tolerance = 0.;
00115 };
00116
00125 template<class X>
00126 SemSolver::BilinearTransformation<X>::BilinearTransformation(Polygon<2,X> const &
00127                                     omega,
00128                                     X const &tolerance)
00128 {
00129     setOmega(omega);
00130     setTolerance(tolerance);
00131 };
00132
00134 template<class X>
00135 SemSolver::BilinearTransformation<X>::~BilinearTransformation()
00136 {
00137 }
00138
00142 template<class X>
00143 void SemSolver::BilinearTransformation<X>::setOmega(Polygon<2,X> const &omega)
00144 {
00145     #ifdef SEMDEBUG
00146         if(omega.size()!=4)
00147             qFatal("SemSolver::BilinearTransformation::setOmega - ERROR : omega must
00148                 be a qu\
00149                     \"adrangle.\");
00149         if(!omega.is_simple())
00150             qFatal("SemSolver::BilinearTransformation::setOmega - ERROR : omega must
00151                 be simp\
00152                     \"le.\");
00152         if(!omega.is_convex())
00153             qFatal("SemSolver::BilinearTransformation::setOmega - ERROR : omega must

```

```

        be conv"\
00154         "ex.");
00155     if(!omega.is_counterclockwise_oriented())
00156         qFatal("SemSolver::BilinearTransformation::setOmega - ERROR : omega must
        be coun"\
00157         "terclockwise oriented.");
00158 #endif // SEMDEBUG
00159     _omega = omega;
00160
00161     Point<2,X> const &A = omega.vertex(0);
00162     Point<2,X> const &B = omega.vertex(1);
00163     Point<2,X> const &C = omega.vertex(2);
00164     Point<2,X> const &D = omega.vertex(3);
00165
00166     _Px = (A.x()+B.x()+C.x()+D.x())/4.;
00167     _Py = (A.y()+B.y()+C.y()+D.y())/4.;
00168     _Qx = (A.x()-B.x()-C.x()+D.x())/4.;
00169     _Qy = (A.y()-B.y()-C.y()+D.y())/4.;
00170     _Rx = (A.x()+B.x()-C.x()-D.x())/4.;
00171     _Ry = (A.y()+B.y()-C.y()-D.y())/4.;
00172     _Sx = (A.x()-B.x()+C.x()-D.x())/4.;
00173     _Sy = (A.y()-B.y()+C.y()-D.y())/4.;
00174
00175     _PQ = _Px*_Qy - _Py*_Qx;
00176     _PR = _Px*_Ry - _Py*_Rx;
00177     _PS = _Px*_Sy - _Py*_Sx;
00178     _QR = _Qx*_Ry - _Qy*_Rx;
00179     _QS = _Qx*_Sy - _Qy*_Sx;
00180     _RS = _Rx*_Sy - _Ry*_Sx;
00181 };
00182
00186 template<class X>
00187 void SemSolver::BilinearTransformation<X>::setTolerance(X const &tolerance)
00188 {
00189     #ifdef SEMDEBUG
00190         if(tolerance<0)
00191             qFatal("SemSolver::BilinearTransformation::setTolerance - ERROR : toleran
            ce must"\
00192             "be non negative.");
00193     #endif // SEMDEBUG
00194     _tolerance = tolerance;
00195 };
00196
00199 template<class X>
00200 inline SemSolver::Polygon<2,X> const &
    SemSolver::BilinearTransformation<X>::omega() const
00201 {
00202     return _omega;
00203 }
00204
00206
00207 template<class X>
00208 inline X const &SemSolver::BilinearTransformation<X>::tolerance() const
00209 {
00210     return _tolerance;
00211 };
00212
00215 template<class X>
00216 SemSolver::Point<2,X>
00217 SemSolver::BilinearTransformation<X>::evaluate(Point<2,X> const &point) c
    onst
00218 {

```

```

00219     X x = _Px - _Qx*point.x() - _Rx*point.y() + _Sx*point.x()*point.y();
00220     X y = _Py - _Qy*point.x() - _Ry*point.y() + _Sy*point.x()*point.y();
00221     return Point<2,X>(x,y);
00222 };
00223
00225
00227
00228 template<class X>
00229 SemSolver::Point<2,X>
00230 SemSolver::BilinearTransformation<X>::evaluateInverse(
00231     Point<2,X> const &point) const
00232 {
00233     X x, y;
00234     X XQ = point.x()*_Qy - point.y()*_Qx;
00235     X XR = point.x()*_Ry - point.y()*_Rx;
00236     X XS = point.x()*_Sy - point.y()*_Sx;
00237     if( std::abs(_QS)>_tolerance && std::abs(_RS)>_tolerance )
00238     {
00239         // Omega is not a parallelogram nor a trapezoid
00240         X psi = pow(XS-_PS+_QR, 2.) + 2.*(XR*_QS+XQ*_RS-XS*_QR) - \
00241             4.*_PQ*_RS;
00242         X n0 = XS-_PS;
00243         X n1 = _QR - sqrt(psi);
00244         X d0 = 2.*_QS;
00245         X d1 = 2.*_RS;
00246         x = -(n0-n1)/d0;
00247         y = -(n0+n1)/d1;
00248     }
00249     else if(std::abs(_RS)>_tolerance) // QS trascurabile
00250     {
00251         // Omega is a vertical trapezoid
00252         x = _Qx*(XR-_PR) / (_Sx*(XQ-_PQ)-_Qx*_QR);
00253         y = (XQ-_PQ) / _QR;
00254     }
00255     else if(std::abs(_QS)>_tolerance)
00256     {
00257         // Omega is a horizontal trapezoid
00258         x = (_PR-XR) / _QR;
00259         y = _Rx*(_PQ-XQ) / (_Sx*(_PR-XR)-_Rx*_QR);
00260     }
00261     else
00262     {
00263         // Omega is a parallelogram
00264         x = (_PR-XR) / _QR;
00265         y = (XQ-_PQ) / _QR;
00266     }
00267     return Point<2,X>(x,y);
00268 };
00269
00272 template<class X>
00273 inline X SemSolver::BilinearTransformation<X>::evaluateJacobianDeterminant(
00274     Point<2,X> const &point) const
00275 {
00276     return _QR - point.x()*_QS + point.y()*_RS;
00277 };
00278
00281 template<class X>
00282 SemSolver::Matrix<double>
00283 SemSolver::BilinearTransformation<X>::evaluateTransposeInverseJacobian(
00284     Point<2,X> const &point) const
00285 {
00286     Matrix<double> tiJ(2,2,1/evaluateJacobianDeterminant(point));

```

```

00287     tiJ[0][0] *= point.x()*_Sy - _Ry;
00288     tiJ[0][1] *= _Rx - point.x()*_Sx;
00289     tiJ[1][0] *= _Qy - point.y()*_Sy;
00290     tiJ[1][1] *= point.y()*_Sx - _Qx;
00291     return tiJ;
00292 };
00293
00294 #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

```

00001 #ifndef BOUNDARYCONDITIONS_HPP
00002 #define BOUNDARYCONDITIONS_HPP
00003
00004 #include <QString>
00005 #include <QStringList>
00006
00007 namespace SemSolver
00008 {
00009     template<int d, class X>

```

```

00010     class BoundaryConditions;
00011 };
00012
00013 #include <map>
00014 #include <vector>
00015
00016 #include <SemSolver/function.hpp>
00017 #include <SemSolver/point.hpp>
00018 #include <SemSolver/semgeometry.hpp>
00019
00021 namespace SemSolver
00022 {
00025
00030     template<int d, class X>
00031     class BoundaryConditions
00032     {
00033     };
00034
00036
00041     template<class X>
00042     class BoundaryConditions<2,X>
00043     {
00044     public:
00046         enum Type
00047         {
00048             UNDEFINED,
00051             DIRICHLET,
00054             NEUMANN,
00057             ROBIN
00058         };
00059
00060         typedef Function< Point<2,X>, X > const * FunctionPtr;
00061         typedef std::map<int, FunctionPtr> FunctionsMap;
00062         typedef std::map<int, Type > TypesMap;
00063         typedef typename FunctionsMap::const_iterator FunctionConstIterator;
00064
00065     private:
00066         // types map
00067         TypesMap _types;
00068
00069         // coefficients
00070         FunctionsMap _g;
00071         FunctionsMap _h;
00072         FunctionsMap _gamma;
00073         FunctionsMap _r;
00074
00075     public:
00076
00077         inline BoundaryConditions();
00078
00079         BoundaryConditions(TypesMap const &types,
00080                             FunctionsMap const *g,
00081                             FunctionsMap const *h,
00082                             FunctionsMap const *gamma,
00083                             FunctionsMap const *r);
00084
00085         template<class Y>
00086         BoundaryConditions(BoundaryConditions<2,Y> const &conditions);
00087
00088         inline int conditions() const;
00089
00090         inline Type const &borderType(int const &index) const;

```

```

00091
00092     inline FunctionPtr dirichletData(int const &index) const;
00093
00094     inline FunctionPtr neumannData(int const &index) const;
00095
00096     inline FunctionPtr robinCoefficient(int const &index) const;
00097
00098     inline FunctionPtr robinData(int const &index) const;
00099
00100     void setBorder(int const &index,
00101                   Type const &type,
00102                   FunctionPtr const f1=0,
00103                   FunctionPtr const f2=0);
00104
00105     QStringList labels() const;
00106
00107     QStringList mmls() const;
00108
00109     void clear();
00110 };
00111 };
00112
00114 template<class X>
00115 SemSolver::BoundaryConditions<2, X>::BoundaryConditions()
00116 {
00117 };
00118
00125 template<class X>
00126 SemSolver::BoundaryConditions<2, X>::BoundaryConditions(TypesMap const &types,
00127                                                         FunctionsMap const *g,
00128                                                         FunctionsMap const *h,
00129                                                         FunctionsMap const *gamma
00130                                                         ,
00131                                                         FunctionsMap const *r)
00132 {
00133     _types(types);
00134     for(typename TypesMap::const_iterator it=types.begin(); it!=types.end(); ++it)
00135     {
00136         switch(it->second)
00137         {
00138             case DIRICHLET:
00139                 #ifdef SEMDEBUG
00140                 if(!g)
00141                     qFatal("SemSolver::BoundaryConditions::BoundaryConditions - ERROR
00142 - tryi"\
00143                             "ng to access null pointer g.");
00144                 #endif
00145                 _g[it->first] = g[it->first];
00146                 break;
00147             case NEUMANN:
00148                 #ifdef SEMDEBUG
00149                 if(!h)
00150                     qFatal("SemSolver::BoundaryConditions::BoundaryConditions - ERROR
00151 - tryi"\
00152                             "ng to access null pointer h.");
00153                 #endif
00154                 _h[it->first] = h[it->first];
00155                 break;
00156             case ROBIN:
00157                 #ifdef SEMDEBUG
00158                 if(!gamma)

```

```

00156             qFatal("SemSolver::BoundaryConditions::BoundaryConditions - ERROR
- tryi"\
00157                 "ng to access null pointer gamma.");
00158             if(!r)
00159                 qFatal("SemSolver::BoundaryConditions::BoundaryConditions - ERROR
- tryi"\
00160                     "ng to access null pointer r.");
00161 #endif
00162         _gamma[it->first] = gamma[it->first];
00163         _r[it->first] = r[it->first];
00164         break;
00165     default:
00166         _types[it->first] = UNDEFINED;
00167     }
00168 }
00169 };
00170
00173 template<class X>
00174 template<class Y>
00175 SemSolver::BoundaryConditions<2, X>::BoundaryConditions(
00176     BoundaryConditions<2, Y> const &conditions)
00177 {
00178     _types = conditions._types;
00179     _g = conditions._g;
00180     _h = conditions._h;
00181     _gamma = conditions._gamma;
00182     _r = conditions._r;
00183 };
00184
00187 template<class X>
00188 inline int SemSolver::BoundaryConditions<2, X>::conditions() const
00189 {
00190     return _types.size();
00191 };
00192
00196 template<class X>
00197 inline typename SemSolver::BoundaryConditions<2, X>::Type const &
00198 SemSolver::BoundaryConditions<2, X>::borderType(int const &index) const
00199 {
00200     typename TypesMap::const_iterator it = _types.find(index);
00201 #ifdef SEMDEBUG
00202     if(it==_types.end() )
00203         qFatal("SemSolver::BoundaryConditions::borderType - ERROR : no border at
index.");
00204 #endif
00205     return it->second;
00206 };
00207
00211 template<class X>
00212 inline SemSolver::Function< SemSolver::Point<2, X>, X > const *
00213 SemSolver::BoundaryConditions<2, X>::dirichletData(int const &index) cons
t
00214 {
00215     FunctionConstIterator it = _g.find(index);
00216 #ifdef SEMDEBUG
00217     if(it==_g.end() )
00218         qFatal("SemSolver::BoundaryConditions::dirichletData - ERROR : the border
at ind"
00219             "ex is not Dirichlet.");
00220 #endif
00221     return it->second;
00222 };

```

```

00223
00227 template<class X>
00228 inline SemSolver::Function< SemSolver::Point<2, X>, X > const *
00229     SemSolver::BoundaryConditions<2, X>::neumannData(int const &index) const
00230 {
00231     FunctionConstIterator it = _h.find(index);
00232     #ifdef SEMDEBUG
00233         if(it==_h.end() )
00234             qFatal("SemSolver::BoundaryConditions::neumannData - ERROR : the border a
00235 t index"\
00236                 " is not Neumann.");
00237     #endif
00238     return it->second;
00239 };
00240
00243 template<class X>
00244 inline SemSolver::Function< SemSolver::Point<2, X>, X > const *
00245     SemSolver::BoundaryConditions<2, X>::robinCoefficient(int const &index) c
00246     onst
00247 {
00248     FunctionConstIterator it = _gamma.find(index);
00249     #ifdef SEMDEBUG
00250         if(it==_gamma.end() )
00251             qFatal("SemSolver::BoundaryConditions::robinCoefficient - ERROR : the bor
00252 der at "\
00253             "index is not Robin.");
00254     #endif
00255     return it->second;
00256 };
00257
00259 template<class X>
00260 inline SemSolver::Function< SemSolver::Point<2, X>, X > const *
00261     SemSolver::BoundaryConditions<2, X>::robinData(int const &index) const
00262 {
00263     FunctionConstIterator it = _r.find(index);
00264     #ifdef SEMDEBUG
00265         if(it==_r.end() )
00266             qFatal("SemSolver::BoundaryConditions::robinData - ERROR : the border at
00267 index i"\
00268                 "s not Robin.");
00269     #endif
00270     return it->second;
00271 };
00272
00275
00279 template<class X>
00280 void SemSolver::BoundaryConditions<2, X>::setBorder(int const &index,
00281     Type const &type,
00282     FunctionPtr const f1,
00283     FunctionPtr const f2)
00284 {
00285     _types[index] = type;
00286     switch(type)
00287     {
00288     case DIRICHLET:
00289     #ifdef SEMDEBUG
00290         if(!f1)
00291             qFatal("SemSolver::BoundaryConditions::setBorder - ERROR : trying to
00292 access "\
00293                 "null pointer f1.");
00294     #endif
00295     _g[index] = f1;

```



```

00295         break;
00296     case NEUMANN:
00297 #ifdef SEMDEBUG
00298         if(!f1)
00299             qFatal("SemSolver::BoundaryConditions::setBorder - ERROR : trying to
access \"\
00300                 \"null pointer f1.\");
00301 #endif
00302         _h[index] = f1;
00303         break;
00304     case ROBIN:
00305 #ifdef SEMDEBUG
00306         if(!f1)
00307             qFatal("SemSolver::BoundaryConditions::setBorder - ERROR : trying to
access \"\
00308                 \"null pointer f1.\");
00309         if(!f2)
00310             qFatal("SemSolver::BoundaryConditions::setBorder - ERROR : trying to
access \"\
00311                 \"null pointer f2.\");
00312 #endif
00313         _gamma[index] = f1;
00314         _r[index] = f2;
00315         break;
00316     default:
00317         _types[index] = UNDEFINED;
00318     }
00319 };
00320
00321
00322
00324 template<class X>
00325 QStringList SemSolver::BoundaryConditions<2, X>::labels() const
00326 {
00327     QStringList list;
00328     for(typename TypesMap::const_iterator it=_types.begin(); it!=_types.end(); ++
it)
00329         list << "S" + QString::number(it->first);
00330     return list;
00331 };
00332
00333
00334
00336 template<class X>
00337 QStringList SemSolver::BoundaryConditions<2, X>::mmls() const
00338 {
00339     QStringList list;
00340     for(typename TypesMap::const_iterator it=_types.begin(); it!=_types.end(); ++
it)
00341     {
00342         QString mml;
00343         switch(it->second)
00344         {
00345             case SemSolver::BoundaryConditions<2, X>::DIRICHLET:
00346                 mml = "<mpadded depth='-2'><msub><mi>u</mi><mrow><mo>#124;</mo><msub
><mi>&G</mi>\
00347                     \"amma;</mi><mi>i</mi></msub></mrow></msub><mo>=</mo><msub><mi>g
</mi><m\"
00348                     \"i>i</mi></msub><mtext>, &nbsp; &nbsp; </mtext><msub><mi>g</mi><mi>
>i</mi>>\"
00349                     \"</msub><mfenced open='(' close=')' separators=', '> <mi> x </mi>
> <mi> \"
00350                     \"y </mi> </mfenced> <mo>=</mo>\"
00351                     + dirichletData(it->first)->mml() +

```

```

00352         "</mpadded>";
00353         break;
00354     case SemSolver::BoundaryConditions<2, X>::NEUMANN:
00355         mml = "<mpadded depth='-2'><mo>&nabla;</mo><msub><mi>u</mi><mrow><mo>
00356             &#124;<\"\\
00357             CenterD\"\\
00358             <mi>i</mi></ms>\"\\
00359             i> y </\"\\
00360             \"mi> </mfenced><mo>=</mo>\"
00361             + neumannData(it->first)->mml() +
00362             "</mpadded>";
00363         break;
00364     case SemSolver::BoundaryConditions<2, X>::ROBIN:
00365         mml = "<mpadded depth='-2'><mo>&nabla;</mo><msub><mi>u</mi><mrow><mo>
00366             &#124;<\"\\
00367             CenterD\"\\
00368             a;</mi>>\"\\
00369             i>&Gamm\"\\
00370             i><mi>i\"\\
00371             <mi>i</\"\\
00372             /mi> <m\"\\
00373             \"i> y </mi> </mfenced><mo>=</mo>\"
00374             + robinCoefficient(it->first)->mml() +
00375             \"<mtext>, &nbsp;&nbsp;&nbsp;</mtext><msub><mi>r</mi><mi>i</mi></msub><
00376             mfenced\"\\
00377             \" open='(' close=')' separators=', '> <mi> x </mi> <mi> y </mi>
00378             </mfenc\"\\
00379             \"ed><mo>=</mo>\"
00380             + robinData(it->first)->mml() +
00381             "</mpadded>";
00382         break;
00383     default:
00384         mml = "<mtext>none</mtext>";
00385     }
00386     list << mml;
00387 }
00388 return list;
00389 };
00390 template<class X>
00391 void SemSolver::BoundaryConditions<2, X>::clear()
00392 {
00393     _types.clear();
00394     _g.clear();
00395     _h.clear();
00396     _gamma.clear();
00397     _r.clear();
00398 };
00399 #endif // BOUNDARYCONDITIONS_HPP
00400

```

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

```
00001 #ifndef IO_BOUNDARYCONDITIONS_HPP
00002 #define IO_BOUNDARYCONDITIONS_HPP
00003
00004 #include <QFile>
00005
00006 #include <SemSolver/boundaryconditions.hpp>
00007
00008 namespace SemSolver
00009 {
00010     namespace IO
00011     {
00012         template<class X>
00013         bool read_boundary_conditions(QFile *file,
00014                                     BoundaryConditions<2, X> &boundary_conditions
00015                                     );
00016     };
00017 };
00018 };
00019
00020 #include <SemSolver/scriptfunction.hpp>
00021
00022 template<class X>
00023 bool SemSolver::IO::read_boundary_conditions(QFile *file,
00024                                             BoundaryConditions<2, X> &bc)
00025 {
00026     bc.clear();
```

```

00027 #ifdef SEMDEBUG
00028     if(!file->open(QIODevice::ReadOnly))
00029     {
00030         qWarning("SemSolver::IO::readBoundaryCondition - ERROR : cannot open file
00031         .");
00032         return false;
00033     }
00034     QTextStream input(file);
00035     QStringList values;
00036     values = next_non_empty_line_values(input);
00037     while(!values.isEmpty())
00038     {
00039         #ifdef SEMDEBUG
00040             if(values.size()<2)
00041             {
00042                 qWarning("SemSolver::IO::readBoundaryCondition - ERROR : wrong number
00043                 of inp\"
00044                     \"uts on a line.\");
00045                 file->close();
00046                 return false;
00047             }
00048             #endif
00049             if(values[1]==\"DIRICHLET\")
00050             {
00051                 #ifdef SEMDEBUG
00052                     if(values.size()!=3)
00053                     {
00054                         qWarning("SemSolver::IO::readBoundaryCondition - ERROR : wrong nu
00055                         mber of\"
00056                             \"inputs on Dirichlet line.\");
00057                         file->close();
00058                         return false;
00059                     }
00060                     #endif
00061                     bc.setBorder(values[0].toInt(),
00062                                 BoundaryConditions<2,X>::DIRICHLET,
00063                                 new ScriptFunction< Point<2, X>, X >(values[2]));
00064                     }
00065                     else if(values[1]==\"NEUMANN\")
00066                     {
00067                         #ifdef SEMDEBUG
00068                             if(values.size()!=3)
00069                             {
00070                                 qWarning("SemSolver::IO::readBoundaryCondition - ERROR : wrong nu
00071                                 mber of\"
00072                                     \"inputs on Neumann line.\");
00073                                 file->close();
00074                                 return false;
00075                             }
00076                             #endif
00077                             bc.setBorder(values[0].toInt(),
00078                                         BoundaryConditions<2,X>::NEUMANN,
00079                                         new ScriptFunction< Point<2, X>, X >(values[2]));
00080                             }
00081                             else if(values[1]==\"ROBIN\")
00082                             {
00083                                 #ifdef SEMDEBUG
00084                                     if(values.size()!=4)
00085                                     {
00086                                         qWarning("SemSolver::IO::readBoundaryCondition - ERROR : wrong nu
00087                                         mber of\"

```

```

00084             "inputs on Robin line.");
00085             file->close();
00086             return false;
00087         }
00088     #endif
00089         bc.setBorder(values[0].toInt(),
00090                     BoundaryConditions<2,X>::ROBIN,
00091                     new ScriptFunction< Point<2, X>, X >(values[2]),
00092                     new ScriptFunction< Point<2, X>, X >(values[3]));
00093     }
00094     #ifdef SEMDEBUG
00095     else
00096     {
00097         qWarning("SemSolver::IO::readBoundaryCondition - ERROR : unknown input
00098         t line "\
00099             "in file.");
00100         file->close();
00101         return false;
00102     };
00103     #endif
00104     values = next_non_empty_line_values(input);
00105     file->close();
00106     return true;
00107 };
00108 };
00109
00110 #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

```

00001 #ifndef BUILDSOLUTION_HPP
00002 #define BUILDSOLUTION_HPP
00003
00004 #include <SemSolver/function.hpp>
00005 #include <SemSolver/semSPACE.hpp>
00006 #include <SemSolver/vector.hpp>
00007
00008 namespace SemSolver
00009 {
00010     namespace PostProcessor
00011     {
00012         template<class X>
00013         void build_solution(const SemSpace<2, X> &space,
00014                             const Vector<X> &coefficients,
00015                             Function< Point<2, X>, X > *&solution)
00016         {
00017             typedef typename SemSpace<2, X>::Element Element;
00018             delete solution;
00019             solution = new Element(&space, coefficients);
00020         };
00021     };
00022 };
00023
00024 #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

```

00001 #ifndef CHOLESKYSOLVE_HPP
00002 #define CHOLESKYSOLVE_HPP
00003
00004 #if defined _WIN32 || defined _WIN64
00005 #    include <SemSolver/math_defines>
00006 #endif
00007
00008 #include <jama_cholesky.h>
00009
00010 #include <SemSolver/matrix.hpp>
00011 #include <SemSolver/vector.hpp>
00012
00013 namespace SemSolver
00014 {
00015
00016     namespace Solver
00017     {
00018         template<class X>
00019         bool cholesky_solve(Matrix<X> const &A,
00020                             Vector<X> const &b,
00021                             Vector<X> &x)
00022         {
00023             JAMA::Cholesky<X> cholesky(A);
00024             #ifdef SEMDEBUG
00025             if(!cholesky.is_spd())
00026             {
00027                 qWarning("SemSolver::Solver::cholesky_solve - ERROR : Matrix A is
00028                 not a "\
00029                 "symmetric, positive definite matrix.");
00030                 return false;
00031             }
00032             #endif
00033             x = cholesky.solve(b);
00034             return true;
00035         };
00036     };
00037 };
00038
00039 #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

```
00001 #ifndef COMPUTEALGEBRAICSYSTEM_HPP
00002 #define COMPUTEALGEBRAICSYSTEM_HPP
00003
00004 #include <SemSolver/semSPACE.hpp>
00005 #include <SemSolver/problem.hpp>
00006 #include <SemSolver/diffusionconvectionreactionequation.hpp>
00007 #include <SemSolver/matrix.hpp>
00008 #include <SemSolver/vector.hpp>
00009
00010 #include <SemSolver/Assembler/computediffusionmatrix.hpp>
00011 #include <SemSolver/Assembler/computeconvectionmatrix.hpp>
00012 #include <SemSolver/Assembler/computereactionmatrix.hpp>
00013 #include <SemSolver/Assembler/computebordermatrix.hpp>
00014 #include <SemSolver/Assembler/computeforcingvector.hpp>
00015 #include <SemSolver/Assembler/computebordervector.hpp>
00016
00017 namespace SemSolver
00018 {
00020
00023     namespace Assembler
00024     {
00027
00028         template<class X>
00029         void compute_algebraic_system(const SemSpace<2, X> &space,
00030                                     const Problem<2, X> &problem,
00031                                     Matrix<X> &A,
00032                                     Vector<X> &f)
```



```

00033     {
00034         if( problem.equation()->type()==
Equation<2, X>::DIFFUSION_CONVECTION_REACTION )
00035     {
00036         const DiffusionConvectionReactionEquation<2, X> *equation =
00037             (const DiffusionConvectionReactionEquation<2, X> *)proble
m.equation();
00038         int n = space.nodes();
00039         A = Matrix<X>(n,n,0.);
00040         f = Vector<X>(n,0.);
00041         Matrix<X> Ad, Ac, Ar, Ab;
00042         Vector<X> ff, fb;
00043         if(equation->diffusion())
00044         {
00045             #ifdef SEMDEBUG
00046                 qDebug() << "diffusion matrix";
00047             #endif
00048             Assembler::compute_diffusion_matrix(space, equation->
diffusion(), Ad);
00049             A += Ad;
00050         }
00051         if(equation->convection())
00052         {
00053             #ifdef SEMDEBUG
00054                 qDebug() << "convection matrix";
00055             #endif
00056             Assembler::compute_convection_matrix(space, equation->
convection(), Ac);
00057             A += Ac;
00058         }
00059         if(equation->reaction())
00060         {
00061             #ifdef SEMDEBUG
00062                 qDebug() << "reaction matrix";
00063             #endif
00064             Assembler::compute_reaction_matrix(space, equation->reaction(
), Ar);
00065             A += Ar;
00066         }
00067         #ifdef SEMDEBUG
00068             qDebug() << "border matrix";
00069         #endif
00070         Assembler::compute_border_matrix(space,
problem.boundaryConditions(),
equation->diffusion(),
problem.parameters()->penalty()
,
00071             Ab);
00072         A += Ab;
00073         if(equation->forcing())
00074         {
00075             #ifdef SEMDEBUG
00076                 qDebug() << "forcing vector";
00077             #endif
00078             Assembler::compute_forcing_vector(space, equation->forcing(),
ff);
00079             f += ff;
00080         }
00081         #ifdef SEMDEBUG
00082             qDebug() << "border vector";
00083         #endif
00084         Assembler::compute_border_vector(space,

```

```

00088                                     problem.boundaryConditions() ,
00089                                     equation->diffusion() ,
00090                                     problem.parameters()->penalty()
00091                                     ,
00092                                     f += fb;
00093                                     }
00094                                     };
00095                                     };
00096 };
00097
00098 #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 &penalty, Matrix< X > &matrix)

The computed matrix is stored in the [Matrix](#) referenced by matrix.

7.16 computebordermatrix.hpp

```

00001 #ifndef COMPUTEBORDERMATRIX_HPP
00002 #define COMPUTEBORDERMATRIX_HPP
00003
00004 #include <SemSolver/semSPACE.hpp>
00005 #include <SemSolver/boundaryconditions.hpp>

```

```

00006 #include <SemSolver/matrix.hpp>
00007 #include <SemSolver/multiindex.hpp>
00008
00010 namespace SemSolver
00011 {
00013     namespace Assembler
00014     {
00020
00021         template<class X>
00022         void compute_border_matrix(const SemSpace<2, X> &space,
00023                                   const BoundaryConditions<2, X> *boundary_condi
00024                                   tions,
00025                                   const Function< Point<2, X>, X > *diffusion,
00026                                   double const &penalty,
00027                                   Matrix<X> &matrix )
00028         {
00029             unsigned n = space.nodes();
00030             int N = space.degree();
00031             unsigned Mb = space.borders();
00032
00033             matrix = Matrix<X>(n,n,0.);
00034             for(unsigned i=0; i<Mb; ++i)
00035             {
00036                 for(int j=0; j<=N; ++j)
00037                 {
00038                     MultiIndex<2> mi;
00039                     mi.setSubIndex(0,i+1);
00040                     mi.setSubIndex(1,j);
00041                     int I = space.borderIndex(mi);
00042                     int border = space.borderId(space.border(i));
00043                     if(boundary_conditions->borderType(border) ==
00044                        BoundaryConditions<2,X>::DIRICHLET)
00045                     {
00046                         X const &alpha = space.borderWeight(mi);
00047                         X const &eta = penalty;
00048                         matrix[I][I] += alpha * eta;
00049                     }
00050                     else if(boundary_conditions->borderType(border) ==
00051                        BoundaryConditions<2,X>::ROBIN)
00052                     {
00053                         X const &alpha = space.borderWeight(mi);
00054                         Point<2,X> const &x = space.borderNode(mi).point();
00055                         X mi = diffusion ? diffusion->evaluate(x) : 0;
00056                         X gamma = boundary_conditions->robinCoefficient(
00057                             border)->evaluate(x);
00058                         matrix[I][I] += alpha * mi * gamma;
00059                     }
00060                 }
00061             }
00062         };
00063     };
00064
00065 #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 &penalty, Vector< X> &vector)`

The computed matrix is stored in the [Matrix](#) referenced by matrix.

7.18 computebordervector.hpp

```
00001 #ifndef COMPUTEBORDERVECTOR_HPP
00002 #define COMPUTEBORDERVECTOR_HPP
00003
00004 #include <SemSolver/semSPACE.hpp>
00005 #include <SemSolver/boundaryconditions.hpp>
00006 #include <SemSolver/vector.hpp>
00007
00008 namespace SemSolver
00009 {
00010
00011     namespace Assembler
00012     {
00013
00014         template<class X>
00015         void compute\_border\_vector(const SemSpace<2, X> &space,
00016                                     const BoundaryConditions<2, X> *boundary_condi
00017 tions,
00018                                     const Function< Point<2, X>, X> *diffusion,
00019                                     const double &penalty,
00020                                     Vector<X> &vector)
00021         {
00022             int n = space.nodes();
00023             int N = space.degree();
00024             int Mb = space.borders();
00025             vector = Vector<X>(n, 0.);
00026             for(int i=0; i<Mb; ++i)
00027             {
```

```

00032         for(int j=0; j<=N; ++j)
00033         {
00034             MultiIndex<2> mi;
00035             mi.setSubIndex(0,i+1);
00036             mi.setSubIndex(1,j);
00037             int I = space.borderIndex(mi);
00038             X alpha = space.borderWeight(mi);
00039             int border = space.borderId(space.border(i));
00040             Point<2,X> const &x = space.borderNode(mi).point();
00041             if(boundary_conditions->borderType(border) ==
BoundaryConditions<2,X>::DIRICHLET)
00042             {
00043                 X const &eta = penalty;
00044                 X g = boundary_conditions->dirichletData(border)->evaluat
e(x);
00045                 vector[I] += alpha * eta * g;
00046             }
00047             else if(boundary_conditions->borderType(border)==
BoundaryConditions<2,X>::NEUMANN)
00048             {
00049                 X mi = diffusion ? diffusion->evaluate(x) : 0;
00050                 X h = boundary_conditions->neumannData(border)->evaluate(
x);
00051                 vector[I] += alpha * mi * h;
00052             }
00053             else if(boundary_conditions->borderType(border)==
BoundaryConditions<2,X>::ROBIN)
00054             {
00055                 X mi = diffusion ? diffusion->evaluate(x) : 0;
00056                 X r = boundary_conditions->robinData(border)->evaluate(x)
;
00057                 vector[I] += alpha * mi * r;
00058             }
00059         }
00060     };
00061 };
00062 };
00063 };
00064
00065 #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

```
00001 #ifndef COMPUTECONVECTIONMATRIX_HPP
00002 #define COMPUTECONVECTIONMATRIX_HPP
00003
00004 #include <SemSolver/function.hpp>
00005 #include <SemSolver/point.hpp>
00006 #include <SemSolver/vector.hpp>
00007 #include <SemSolver/matrix.hpp>
00008
00009 namespace SemSolver
00010 {
00011
00012     namespace Assembler
00013     {
00014
00015         template<class X>
00021         void compute_convection_matrix(const SemSpace<2, X> &space,
00022                                     const Function< Point<2, X>,
00023                                     Vector<X> > *convection,
00024                                     Matrix<X> &matrix)
00025         {
00026             typedef typename SemSpace<2,X>::Node Node;
00027
00028             int n = space.nodes();
00029
00030             matrix = Matrix<X>(n,n,0.);
00031
00032             for (int I0=0; I0<n; ++I0)
00033             {
00034                 Node const &node0 = space.node(I0);
00035                 for (int I1=0; I1<n; ++I1)
00036                 {
00037                     Node const &node1 = space.node(I1);
00038                     int l0=0, l1=0;
00039                     while(l0<node0.supportSubDomains() && l1<node1.supportSubDomains())
00040                     {
00041                         if (node0.subDomainIndex(l0).subIndex(0) <
00042                             node1.subDomainIndex(l1).subIndex(0))
00043                             ++l0;
00044                         else if (node0.subDomainIndex(l0).subIndex(0) >
00045                             node1.subDomainIndex(l1).subIndex(0))
00046                             ++l1;
```

```

00047         else
00048         {
00049             MultiIndex<3> mi0 = node0.subDomainIndex(l0);
00050             int i = mi0.subIndex(0);
00051             X alpha = space.subDomainWeight(mi0);
00052             Point<2,X> x0 = node0.point();
00053             Vector<X> beta = convection->evaluate(x0);
00054             Vector<X> const &grad1 =
00055                 space.baseFunction(I1)->\
00056                 evaluateRestrictionGradient(i,x0);
00057             matrix[I0][I1] += alpha * scalar(beta,grad1);
00058             ++l0;
00059             ++l1;
00060         }
00061     }
00062 }
00063 };
00064 };
00065 };
00066 };
00067
00068 #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

```

00001 #ifndef COMPUTEDIFFUSIONMATRIX_HPP
00002 #define COMPUTEDIFFUSIONMATRIX_HPP
00003
00004 #include <SemSolver/function.hpp>
00005 #include <SemSolver/point.hpp>
00006 #include <SemSolver/matrix.hpp>
00007 #include <SemSolver/semSPACE.hpp>
00008
00009 namespace SemSolver
00010 {
00011
00012     namespace Assembler
00013     {
00014
00015         template<class X>
00016         void compute_diffusion_matrix(const SemSpace<2, X> &space,
00017                                     const Function< Point<2, X>, X> *diffusion,
00018                                     Matrix<X> &matrix)
00019         {
00020             typedef typename SemSpace<2, X>::Node Node;
00021
00022             int n = space.nodes();
00023             int N = space.degree();
00024
00025             matrix = Matrix<X>(n,n,0.);
00026
00027             for (int I0=0; I0<n; ++I0)
00028             {
00029                 Node const &node0 = space.node(I0);
00030                 for (int I1=0; I1<n; ++I1)
00031                 {
00032                     Node const &node1 = space.node(I1);
00033                     int l0=0, l1=0;
00034                     while(l0<node0.supportSubDomains() && l1<node1.supportSubDomains())
00035                     {
00036                         if (node0.subDomainIndex(l0).subIndex(0) < node1.subDomainIndex(l1).subIndex(0))
00037                             ++l0;
00038                         else if (node0.subDomainIndex(l0).subIndex(0) > node1.subDomainIndex(l1).subIndex(0))
00039                             ++l1;
00040                         else
00041                         {
00042                             int i = node0.subDomainIndex(l0).subIndex(0);
00043                             for (int j2=0; j2<=N; ++j2)
00044                             {
00045                                 for (int k2=0; k2<=N; ++k2)
00046                                 {
00047                                     MultiIndex<3> mi2;
00048                                     mi2.setSubIndex(0,i);
00049                                     mi2.setSubIndex(1,j2);
00050                                     mi2.setSubIndex(2,k2);
00051                                     X alpha = space.subDomainWeight(mi2);
00052                                     Point<2,X> x2 = space.subDomainNode(mi2).point();
00053
00054                                     X mi = diffusion->evaluate(x2);
00055                                     Vector<X> const &grad0 = space.baseFunction(I0)->evaluateRestrictionGradient(i,x2);

```



```

00060                                     Vector<X> const &grad1 = space.baseFunction(I
00061 1)->evaluateRestrictionGradient(i,x2);
00061                                     matrix[I0][I1] += alpha * mi * scalar(grad1,g
00062 rad0);
00062                                     }
00063                                     }
00064                                     ++i0;
00065                                     ++i1;
00066                                     }
00067                                     }
00068                                     }
00069                                     };
00070                                     };
00071                                     };
00072 };
00073
00074 #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 > &vec-
tor)
The computed matrix is stored in the [Matrix](#) referenced by matrix.

7.24 computeforcingvector.hpp

```

00001 #ifndef COMPUTEFORCINGVECTOR_HPP
00002 #define COMPUTEFORCINGVECTOR_HPP
00003
00004 #include <SemSolver/problem.hpp>
00005 #include <SemSolver/vector.hpp>

```

```

00006
00007 namespace SemSolver
00008 {
00010
00013     namespace Assembler
00014     {
00017
00018         template<class X>
00019         void compute_forcing_vector(const SemSpace<2, X> &space,
00020                                     const Function< Point<2, X>, X > *forcing,
00021                                     Vector<X> &vector)
00022         {
00023             typedef typename SemSpace<2,X>::Node Node;
00024
00025             int n = space.nodes();
00026             vector = Vector<X>(n,0.);
00027             for(int I=0; I<n; ++I)
00028             {
00029                 Node const &node = space.node(I);
00030                 for(int l=0; l<node.supportSubDomains(); ++l)
00031                 {
00032                     MultiIndex<3> Il = node.subDomainIndex(l);
00033                     X const &alpha = space.subDomainWeight(Il);
00034                     Point<2,X> const &x = space.subDomainNode(Il).point();
00035                     X f = forcing->evaluate(x);
00036                     vector[I] += alpha * f;
00037                 }
00038             }
00039         };
00040     };
00041 };
00042
00043 #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

```

00001 #ifndef COMPUTEPLOTDATA_HPP
00002 #define COMPUTEPLOTDATA_HPP
00003
00004 #include <SemSolver/semSPACE.hpp>
00005 #include <SemSolver/vector.hpp>
00006
00007 #include <qwt3d_types.h>
00008
00009 namespace SemSolver
00010 {
00011     namespace PostProcessor
00012     {
00013         template<class X>
00014         void compute_plot_data(const SemSpace<2, X> &space,
00015                               const Vector<X> &u,
00016                               Qwt3D::TripleField &data,
00017                               Qwt3D::CellField &poly)
00018         {
00019             for(int i=0; i<u.rows(); ++i)
00020                 data.push_back(Qwt3D::Triple(space.node(i).point().x(), space.
00021 node(i).point().y(), u[i]));
00022             for(int i=0; i<space.subDomains(); ++i)
00023             {
00024                 for(int j=0; j<space.degree(); ++j)
00025                 {
00026                     for(int k=0; k<space.degree(); ++k)
00027                     {
00028                         Qwt3D::Cell cell;
00029                         MultiIndex<3> mi;
00030                         mi.setSubIndex(0,i);
00031                         mi.setSubIndex(1,j);
00032                         mi.setSubIndex(2,k);
00033                         cell.push_back(space.subDomainIndex(mi));
00034                         mi.setSubIndex(0,i);
00035                         mi.setSubIndex(1,j+1);
00036                         mi.setSubIndex(2,k);
00037                         cell.push_back(space.subDomainIndex(mi));
00038                         mi.setSubIndex(0,i);
00039                         mi.setSubIndex(1,j+1);
00040                         mi.setSubIndex(2,k+1);
00041                         cell.push_back(space.subDomainIndex(mi));
00042                         mi.setSubIndex(0,i);
00043                         mi.setSubIndex(1,j);
00044                         mi.setSubIndex(2,k+1);
00045                         cell.push_back(space.subDomainIndex(mi));
00046                         poly.push_back(cell);
00047                     }
00048                 }
00049             }
00050         }
00051     }
00052 };
00053

```

```
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__
```

```

00005 #  ifdef __GLIBC__
00006 #      if ((__GLIBC__ > 2) || ((__GLIBC__ == 2) && (__GLIBC_MINOR__ >= 1)))
00007 #          include <stdint.h>
00008 #      endif
00009 #  endif
00010 #endif
00011
00012 #if defined _WIN32 || defined _WIN64
00013 #  undef max
00014 #  include <limits>
00015 #endif
00016
00017 #include <boost/shared_ptr.hpp>
00018
00019 #include <CGAL/Cartesian.h>
00020 #include <CGAL/Filtered_kernel.h>
00021 #include <CGAL/Straight_skeleton_2.h>
00022 #include <CGAL/create_straight_skeleton_from_polygon_with_holes_2.h>
00023 #include <CGAL/squared_distance_2.h>
00024 #include <SemSolver/pointsset.hpp>
00025 #include <SemSolver/semparameters.hpp>
00026 #include <SemSolver/segment.hpp>
00027 #include <SemSolver/polygonation.hpp>
00028 #include <SemSolver/polygonwithholes.hpp>
00029 #include <SemSolver/pslg.hpp>
00030 #include <SemSolver/PreProcessor/computepolygonwithholesfrompslg.hpp>
00031
00032 namespace SemSolver
00033 {
00034
00035     namespace PreProcessor
00036     {
00037         template<class X>
00038         bool compute_polygonation_from_pslg(const PSLG<X> &pslg,
00039                                             Polygonation<2,X> &polygonation)
00040         {
00041             typedef CGAL::Straight_skeleton_2< CGAL::Filtered_kernel<
00042                 CGAL::Simple_cartesian<X> > > Skeleton;
00043             typedef typename boost::shared_ptr< Skeleton > SkeletonPtr;
00044
00045             typedef typename Skeleton::Vertex_iterator VertexIterator;
00046             typedef typename Skeleton::Face_iterator FaceIterator;
00047             typedef typename Skeleton::Halfedge_handle HalfedgeHandle;
00048             typedef typename Skeleton::Halfedge_iterator HalfedgeIterator;
00049             typedef typename Segment<2,X>::less Segment_less;
00050             typedef std::map<Segment<2,X>, int, Segment_less> SegmentIdsMap;
00051             typedef typename SegmentIdsMap::value_type Pair;
00052             PolygonWithHoles<2,X> domain;
00053             if(!compute_polygon_with_holes_from_pslg(pslg, domain))
00054             {
00055                 return false;
00056             }
00057             SkeletonPtr skeleton = CGAL::create_interior_straight_skeleton_2(doma
00058 in.cgal());
00059
00060             polygonation.clear();
00061
00062             SegmentIdsMap face_id;
00063             int index = 0;
00064             for (FaceIterator it = skeleton->faces_begin();
00065                  it!=skeleton->faces_end();++it)
00066                 face_id.insert(Pair(

```

```

00070         Segment<2, X>(it->halfedge()->vertex()->point(),
00071                       it->halfedge()->opposite()->vertex()->point
00072         ()),
00073         ++index));
00074     SegmentIdsMap boundary_id;
00075     index=0;
00076     for (HalfedgeIterator it = skeleton->halfedges_begin();
00077          it!= skeleton->halfedges_end(); ++it)
00078         if(it->is_border())
00079             boundary_id.insert(Pair(
00080                 Segment<2, X>(it->vertex()->point(),
00081                               it->opposite()->vertex()->point()), ++i
00082             ndex));
00083     for (FaceIterator it = skeleton->faces_begin();
00084          it != skeleton->faces_end(); ++it)
00085     {
00086         Polygon<2, X> polygon;
00087         std::vector<int> neighbours;
00088         HalfedgeHandle halfedge = it->halfedge();
00089         do
00090         {
00091             polygon.push_back(halfedge->vertex()->point());
00092             if(halfedge->is_bisector())
00093                 neighbours.push_back(face_id[
00094                     Segment<2, X>(halfedge->opposite()->face()->halfe
00095                     dge()->vertex()->point(),
00096                               halfedge->opposite()->face()->halfe
00097                     dge()->opposite()->vertex()->point())]);
00098             else
00099                 neighbours.push_back(-boundary_id[
00100                     Segment<2, X>(halfedge->opposite()->vertex()->poi
00101                     nt(),
00102                               halfedge->opposite()->opposite()->v
00103                     ertex()->point())]);
00104             halfedge = halfedge->next();
00105         }
00106         while(halfedge->vertex()->point()!=it->halfedge()->vertex()->poi
00107         nt());
00108         polygonation.addElement(polygon, neighbours);
00109     }
00110     return true;
00111 }
00112 }
00113 #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

```

00001 #ifndef COMPUTEPOLYGONWITHHOLESFROMPSLG_HPP
00002 #define COMPUTEPOLYGONWITHHOLESFROMPSLG_HPP
00003
00004 #include <SemSolver/polygonwithholes.hpp>
00005 #include <SemSolver/pslg.hpp>
00006 #include <SemSolver/sequenceslist.hpp>
00007
00008 namespace SemSolver
00009 {
00010
00011     namespace PreProcessor
00012     {
00013         template<class X>
00014         bool compute_vertices_sequences_from_pslg(PSLG<X> const &pslg,
00015             Sequences_list &vertices_sequences)
00016         {
00017             for(unsigned i=0; i<pslg.segments(); ++i)
00018             {
00019                 bool segment_added = false;
00020                 for(Sequences_list::iterator it = vertices_sequences.begin();
00021                     it!=vertices_sequences.end() && !segment_added;
00022                     ++it)
00023                 {
00024                     if(pslg.segment(i).source==it->back())
00025                     {
00026                         it->push_back(pslg.segment(i).target);
00027                         segment_added = true;
00028                     }
00029                     else if(pslg.segment(i).target==it->back())
00030                     {
00031                         it->push_back(pslg.segment(i).source);
00032                         segment_added = true;
00033                     }
00034                 }
00035             }
00036         }
00037     }
00038 }

```

```

00036         }
00037     }
00038     if(!segment_added)
00039     {
00040         Sequence sequence;
00041         sequence.push_back(pslg.segment(i).source);
00042         sequence.push_back(pslg.segment(i).target);
00043         vertices_sequences.push_back(sequence);
00044     }
00045 }
00046 Sequences_list::iterator begin = vertices_sequences.begin();
00047 Sequences_list::iterator end = vertices_sequences.end();
00048 Sequences_list::iterator before_end = --vertices_sequences.end();
00049 for(Sequences_list::iterator it_0 = begin; it_0!=before_end && it_0 !
=end;
00050     ++it_0)
00051 {
00052     bool is_closed = (it_0->front()==it_0->back());
00053     if(!is_closed)
00054     {
00055         Sequences_list::iterator after_it_0 = it_0; ++after_it_0;
00056         for(Sequences_list::iterator it_1 = after_it_0; it_1!=end &&
!is_closed; ++it_1)
00057         {
00058             if(it_0->back()==it_1->front())
00059             {
00060                 for(Sequence::const_iterator it_2=it_1->begin();
++it_2!=it_1->end(); )
00061                     it_0->push_back(*it_2);
00062                 vertices_sequences.erase(it_1);
00063                 end = vertices_sequences.end();
00064                 is_closed = (it_0->front()==it_0->back());
00065                 it_1 = it_0;
00066             }
00067             else if(it_0->back()==it_1->back())
00068             {
00069                 for(Sequence::const_reverse_iterator it_2=it_1->rbegin()
n();
00070                     ++it_2!=it_1->rend(); )
00071                     it_0->push_back(*it_2);
00072                 vertices_sequences.erase(it_1);
00073                 end = vertices_sequences.end();
00074                 is_closed = (it_0->front()==it_0->back());
00075                 it_1 = it_0;
00076             }
00077             else if(it_0->front()==it_1->back())
00078             {
00079                 for(Sequence::const_reverse_iterator it_2=it_1->rbegin()
n();
00080                     ++it_2!=it_1->rend(); )
00081                     it_0->push_front(*it_2);
00082                 vertices_sequences.erase(it_1);
00083                 end = vertices_sequences.end();
00084                 is_closed = (it_0->front()==it_0->back());
00085                 it_1 = it_0;
00086             }
00087             else if(it_0->front()==it_1->front())
00088             {
00089                 for(Sequence::const_iterator it_2=it_1->begin();
++it_2!=it_1->end(); )
00090                     it_0->push_front(*it_2);
00091                 vertices_sequences.erase(it_1);
00092             }
00093         }
00094     }

```



```

00095             end = vertices_sequences.end();
00096             is_closed = (it_0->front()==it_0->back());
00097             it_1 = it_0;
00098         }
00099     }
00100 }
00101     before_end = --vertices_sequences.end();
00102 }
00103     return true;
00104 }
00105
00106 template<class X>
00107 bool compute_polygon_with_holes_from_pslg(PSLG<X> const &pslg,
00108                                           PolygonWithHoles<2,X> &polygon)
00109
00110 {
00111     typedef std::list< Polygon<2,X> > Polygons_list;
00112     typedef typename Polygons_list::iterator Polygon_iterator;
00113     typedef typename Polygons_list::const_iterator Polygon_const_iterator
00114 ;
00115     typedef std::map<int,int> Vertices_map;
00116     typedef typename Polygon<2,X>::Vertex_const_iterator Vertex_const_ite
00117 rator;
00118     polygon.clear();
00119     Sequences_list vertices_sequences;
00120     compute_vertices_sequences_from_pslg(pslg,vertices_sequences);
00121     if(vertices_sequences.size()<1)
00122         return false;
00123     bool *is_used_vertex = new bool[pslg.vertices()];
00124     Vertices_map vertices_map;
00125     for(unsigned i=0; i<pslg.vertices(); ++i)
00126     {
00127         is_used_vertex[i] = false;
00128         vertices_map.insert(Vertices_map::value_type(pslg.vertex(i).numbe
00129 r,i));
00130     }
00131     Polygons_list polygons;
00132     for(Sequences_list::const_iterator it_0=vertices_sequences.begin();
00133         it_0!=vertices_sequences.end(); ++it_0)
00134     {
00135         #ifdef SEMDEBUG
00136         if(it_0->front() != it_0->back())
00137         {
00138             qWarning("PSLG::to_polygon : segments must define closed poly
00139 gons");
00140             return false;
00141         }
00142         #endif
00143         Polygon<2,X> subpolygon;
00144         for(Sequence::const_iterator it_1=it_0->begin(); it_1!=--it_0->en
00145 d(); ++it_1)
00146         {
00147             #ifdef SEMDEBUG
00148             if(is_used_vertex[vertices_map[*it_1]])
00149             {
00150                 qWarning("PSLG::to_polygon : no more than two segments mu
00151 st pass through each vertex");
00152                 return false;
00153             }
00154             else

```

```

00151         {
00152     #endif
00153         double const &x = pslg.vertex(vertices_map[*it_1]).x;
00154         double const &y = pslg.vertex(vertices_map[*it_1]).y;
00155         subpolygon.push_back(Point<2,X>(x,y));
00156         is_used_vertex[vertices_map[*it_1]] = true;
00157     #ifdef SEMDEBUG
00158         }
00159     #endif
00160     }
00161     #ifdef SEMDEBUG
00162     if(subpolygon.is_empty())
00163     {
00164         qWarning("PSLG::to_polygon : segments must define non empty p
00165         olygons");
00166         return false;
00167     }
00168     if(!subpolygon.is_simple())
00169     {
00170         qWarning("PSLG::to_polygon : segments must define simple poly
00171         gons");
00172         return false;
00173     }
00174     polygons.push_back(subpolygon);
00175     #ifdef SEMDEBUG
00176     for(unsigned i=0; i<pslg.vertices(); ++i)
00177     {
00178         if(!is_used_vertex[i])
00179         {
00180             qWarning("PSLG::to_polygon : there are unused vertex");
00181             return false;
00182         }
00183     }
00184     #endif
00185     Polygon_iterator outer;
00186     bool found = false;
00187     for(Polygon_iterator it_0 = polygons.begin(); it_0!=polygons.end() &&
00188         !found;
00189         ++it_0)
00190     {
00191         bool is_outer = true;
00192         for(Polygon_const_iterator it_1 = polygons.begin(); it_1!=polygon
00193             s.end()
00194             &&is_outer; ++it_1)
00195             if(it_0!=it_1)
00196                 for (Vertex_const_iterator it_2 = it_1->vertices_begin();
00197                     it_2 != it_1->vertices_end() &&is_outer; ++it_2)
00198                     if(!it_0->has_on_bounded_side(*it_2))
00199                         is_outer = false;
00200         if(is_outer)
00201         {
00202             outer = it_0;
00203             found = true;
00204         }
00205     }
00206     if(outer->is_clockwise_oriented())
00207         outer->reverse_orientation();
00208     polygon.outer_boundary() = *outer;
00209     polygons.erase(outer);

```

```

00208 #ifdef SEMDEBUG
00209         if(polygons.size()<pslg.holes())
00210         {
00211             qWarning("PSLG::to_polygon : there are too many holes");
00212             return false;
00213         }
00214         if(polygons.size()>pslg.holes())
00215         {
00216             qWarning("PSLG::to_polygon : there are not enough holes");
00217             return false;
00218         }
00219 #endif
00220         for(unsigned i=0; i<pslg.holes(); ++i)
00221         {
00222             Polygon_iterator it=polygons.begin();
00223             Point<2,X> hole(pslg.hole(i).x,pslg.hole(i).y);
00224 #ifdef SEMDEBUG
00225             while(!it->has_on_bounded_side(hole))
00226             {
00227                 ++it;
00228                 if(it==polygons.end())
00229                 {
00230                     qWarning("PSLG::to_polygon : there is no hole inside inne
00231 r polygon");
00232                     return false;
00233                 }
00234 #endif
00235                 if(it->is_counterclockwise_oriented())
00236                     it->reverse_orientation();
00237                 polygon.add_hole(*it);
00238                 polygons.erase(it);
00239             }
00240             return true;
00241         }
00242     }
00243 }
00244
00245 #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

```

00001 #ifndef COMPUTEREACTIONMATRIX_HPP
00002 #define COMPUTEREACTIONMATRIX_HPP
00003
00004 #include <SemSolver/semSPACE.hpp>
00005 #include <SemSolver/matrix.hpp>
00006
00007 namespace SemSolver
00008 {
00009
00010     namespace Assembler
00011     {
00012
00013         template<class X>
00014         void compute_reaction_matrix(const SemSpace<2, X> &space,
00015                                     Function< Point<2, X>, X> const *reaction,
00016                                     Matrix<double> &matrix)
00017         {
00018             typedef typename SemSpace<2,X>::Node Node;
00019
00020             int n = space.nodes();
00021             matrix = Matrix<X>(n,n,0.);
00022             for(int I0=0; I0<n; ++I0)
00023             {
00024                 Node const &node0 = space.node(I0);
00025                 Point<2,X> x = node0.point();
00026                 for(int l=0; l<node0.supportSubDomains(); ++l)
00027                 {
00028                     MultiIndex<3> mil = node0.subDomainIndex(l);
00029                     int I1 = space.subDomainIndex(mil);
00030                     X alpha = space.subDomainWeight(mil);
00031                     X gamma = reaction->evaluate(x);
00032                     matrix[I0][I1] += alpha * gamma;
00033                 }
00034             }
00035         };
00036     };
00037 };
00038
00039 #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

```
00001 #ifndef COMPUTESOLUTIONHULL_HPP
00002 #define COMPUTESOLUTIONHULL_HPP
00003
00004 #include <SemSolver/vector.hpp>
00005 #include <SemSolver/semSPACE.hpp>
00006
00007 namespace SemSolver
00008 {
00009     namespace PostProcessor
00010     {
00011         template<class X>
00012         void compute_solution_hull(const SemSpace<2, X> &space,
00013                                   const Vector<X> &coefficients,
00014                                   double &xmin,
00015                                   double &ymin,
00016                                   double &zmin,
00017                                   double &xmax,
00018                                   double &ymax,
00019                                   double &zmax)
00020         {
00021             if(!space.nodes())
00022                 return;
00023             if(!coefficients.rows())
00024                 return;
00025             xmin = xmax = space.node(0).point().x();
00026             ymin = ymax = space.node(0).point().y();
00027             zmin = zmax = coefficients[0];
00028             for(unsigned i=1; i<space.nodes(); ++i)
00029             {
00030                 if(space.node(i).point().x() < xmin)
00031                     xmin = space.node(i).point().x();
00032                 if(space.node(i).point().x() > xmax)
00033                     xmax = space.node(i).point().x();
00034                 if(space.node(i).point().y() < ymin)
00035                     ymin = space.node(i).point().y();
00036                 if(space.node(i).point().y() > ymax)
```

```

00037             ymax = space.node(i).point().y();
00038         }
00039         for(int i=1; i<coefficients.rows(); ++i)
00040         {
00041             /***** Not precise *****/
00042             if(coefficients[i] < zmin)
00043                 zmin = coefficients[i];
00044             if(coefficients[i] > zmax)
00045                 zmax = coefficients[i];
00046         }
00047     };
00048 };
00049 };
00050
00051 #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

```

00001 #ifndef DIFFUSIONCONVECTIONREACTIONEQUATION_HPP
00002 #define DIFFUSIONCONVECTIONREACTIONEQUATION_HPP
00003
00004 #include <QDebug>
00005
00006 namespace SemSolver

```

```

00007 {
00008     template<int d, class X>
00009     class DiffusionConvectionReactionEquation;
00010 };
00011
00012 #include <SemSolver/equation.hpp>
00013 #include <SemSolver/function.hpp>
00014 #include <SemSolver/point.hpp>
00015 #include <SemSolver/vector.hpp>
00016
00017 namespace SemSolver
00018 {
00019     template<int d, class X>
00020     class DiffusionConvectionReactionEquation
00021     : public Equation<d, X>
00022     {
00023     // coefficients
00024     Function< Point<d, X>, X >          *_diffusion;
00025     Function< Point<d, X>, Vector<X> > *_convection;
00026     Function< Point<d, X>, X >          *_reaction;
00027     Function< Point<d, X>, X >          *_forcing;
00028
00029     public:
00030
00031         typedef typename Equation<d, X>::Type Type;
00032
00033         DiffusionConvectionReactionEquation()
00034         {
00035             _diffusion = 0;
00036             _convection = 0;
00037             _reaction = 0;
00038             _forcing = 0;
00039         };
00040
00041         ~DiffusionConvectionReactionEquation()
00042         {
00043             delete _diffusion;
00044             delete _convection;
00045             delete _reaction;
00046             delete _forcing;
00047         };
00048
00049         inline Type type() const
00050         {
00051             return Equation<d, X>::DIFFUSION_CONVECTION_REACTION;
00052         };
00053
00054         QString mml() const;
00055
00056         inline void setDiffusion(Function< Point<d, X>, X > *diffusion)
00057         {
00058             delete _diffusion;
00059             _diffusion = diffusion;
00060         };
00061
00062         inline void setConvection(Function< Point<d, X>, Vector<X> > *convection)
00063         {
00064             delete _convection;
00065             _convection = convection;
00066         };
00067
00068     };
00069
00070     {
00071         delete _convection;
00072         _convection = convection;
00073     };
00074
00075     };

```

```

00084
00087     inline void setReaction(Function< Point<d, X>, X > *reaction)
00088     {
00089         delete _reaction;
00090         _reaction = reaction;
00091     };
00092
00095     inline void setForcing(Function< Point<d, X>, X > *forcing)
00096     {
00097         delete _forcing;
00098         _forcing = forcing;
00099     };
00100
00103     inline Function< Point<d, X>, X > *diffusion() const
00104     {
00105         return _diffusion;
00106     };
00107
00110     inline Function< Point<d, X>, Vector<X> > *convection() const
00111     {
00112         return _convection;
00113     };
00114
00117     inline Function< Point<d, X>, X > *reaction() const
00118     {
00119         return _reaction;
00120     }
00121
00124     inline Function< Point<d, X>, X > *forcing() const
00125     {
00126         return _forcing;
00127     };
00128 };
00129
00137 template<class X>
00138 class DiffusionConvectionReactionEquation<2, X>
00139     : public Equation<2, X>
00140 {
00141     // coefficients
00142     Function< Point<2, X>, X >          *_diffusion;
00143     Function< Point<2, X>, Vector<X> > *_convection;
00144     Function< Point<2, X>, X >          *_reaction;
00145     Function< Point<2, X>, X >          *_forcing;
00146
00147 public:
00148
00149     typedef typename Equation<2, X>::Type Type;
00150
00153     DiffusionConvectionReactionEquation()
00154     {
00155         _diffusion = 0;
00156         _convection = 0;
00157         _reaction = 0;
00158         _forcing = 0;
00159     };
00160
00162     ~DiffusionConvectionReactionEquation()
00163     {
00164         delete _diffusion;
00165         delete _convection;
00166         delete _reaction;
00167         delete _forcing;

```



```

00168         };
00169
00172         inline Type type() const
00173         {
00174             return Equation<2, X>::DIFFUSION_CONVECTION_REACTION;
00175         };
00176
00179         QString mml() const
00180         {
00181             QString mml = "<table><mtr><mt><td>";
00182             if(_diffusion && _convection && _reaction)
00183                 mml += "<mo>-</mo><mo>&Del;&CenterDot;</mo><mfenced open='(' clos
e=')'><mrow><mi>\
00184                     ">&mu;</mi><mo>&Del;</mo> <mi>u</mi></mrow></mfenced><mo>+
</mo><mi>mathva\"
00185                     "riant='bold'>b</mi><mo>&CenterDot;</mo><mo>&Del;</mo> <mi
>u</mi><mo>+</m\"
00186                     "o><mi>&sigma;</mi><mi>u</mi>";
00187             else if(_diffusion && _convection)
00188                 mml += "<mo>-</mo><mo>&Del;&CenterDot;</mo><mfenced open='(' clos
e=')'><mrow><mi>\
00189                     ">&mu;</mi><mo>&Del;</mo> <mi>u</mi></mrow></mfenced><mo>+
</mo><mi>mathva\"
00190                     "riant='bold'>b</mi><mo>&CenterDot;</mo><mo>&Del;</mo> <mi
>u</mi>";
00191             else if(_diffusion && _reaction)
00192                 mml += "<mo>-</mo><mo>&Del;&CenterDot;</mo><mfenced open='(' clos
e=')'><mrow><mi>\
00193                     ">&mu;</mi><mo>&Del;</mo> <mi>u</mi></mrow></mfenced><mo>+
</mo><mi>&sigma\"
00194                     "</mi><mi>u</mi>";
00195             else if(_convection && _reaction)
00196                 mml += "<mi>mathvariant='bold'>b</mi><mo>&CenterDot;</mo><mo>&Del
</mo> <mi>u</m\"
00197                     "i><mo>+</mo><mi>&sigma;</mi><mi>u</mi>";
00198             else if(_diffusion)
00199                 mml += "<mo>-</mo><mo>&Del;&CenterDot;</mo><mfenced open='(' clos
e=')'><mrow><mi>\
00200                     ">&mu;</mi><mo>&Del;</mo> <mi>u</mi></mrow></mfenced>";
00201             else if(_convection)
00202                 mml += "<mi>mathvariant='bold'>b</mi><mo>&CenterDot;</mo><mo>&Del
</mo> <mi>u</m\"
00203                     "i>";
00204             else if(_reaction)
00205                 mml += "<mi>&sigma;</mi><mi>u</mi>";
00206             else
00207                 mml += "<mn>0</mn>";
00208             if(_forcing)
00209                 mml += "<mo>=</mo><mi>f</mi></mt></mtr>";
00210             else
00211                 mml += "<mo>=</mo><mn>0</mn></mt></mtr>";
00212             if(_diffusion || _convection || _reaction || _forcing)
00213             {
00214                 mml += "<mtr><mt><table><mtr>";
00215                 if(_diffusion)
00216                     mml += "<td><mi>&mu;</mi> <mfenced open='(' close=')'>separa
tors=', '> <mi> \"
00217                     "x </mi> <mi> y </mi> </mfenced> <mo>=</mo>\"
00218                     + _diffusion->mml()
00219                     + "</td>";
00220                 if(_convection)
00221                     mml += "<td><mi>mathvariant='bold'>b</mi> <mfenced open='('

```

```

        close=')' separ"\
00222                                     "ators=', '> <mi> x </mi> <mi> y </mi> </mfenced> <mo>=
</mo>"
00223                                     + _convection->mml()
00224                                     + "</mtd>";
00225             if(_reaction)
00226                 mml += "<mtd><mi>&sigma;</mi> <mfenced open='(' close=')' sep
arators=', '> <m"\"
00227                                     "i> x </mi> <mi> y </mi> </mfenced> <mo>= </mo>"
00228                                     + _reaction->mml()
00229                                     + "</mtd>";
00230             if(_forcing)
00231                 mml += "<mtd><mi>f</mi> <mfenced open='(' close=')' separator
s=', '> <mi> x <"\"
00232                                     "/mi> <mi> y </mi> </mfenced> <mo>= </mo>"
00233                                     + _forcing->mml()
00234                                     + "</mtd>";
00235                 mml += "<mtr></mtable></mtd></mtr>";
00236             }
00237             mml += "</mtable>";
00238             return mml;
00239         };
00240
00243         inline void setDiffusion(Function< Point<2, X>, X > *diffusion)
00244         {
00245             delete _diffusion;
00246             _diffusion = diffusion;
00247         };
00248
00251         inline void setConvection(Function< Point<2, X>, Vector<X> > *convection)
00252         {
00253             delete _convection;
00254             _convection = convection;
00255         };
00256
00259         inline void setReaction(Function< Point<2, X>, X > *reaction)
00260         {
00261             delete _reaction;
00262             _reaction = reaction;
00263         };
00264
00267         inline void setForcing(Function< Point<2, X>, X > *forcing)
00268         {
00269             delete _forcing;
00270             _forcing = forcing;
00271         };
00272
00275         inline Function< Point<2, X>, X > *diffusion() const
00276         {
00277             return _diffusion;
00278         };
00279
00282         inline Function< Point<2, X>, Vector<X> > *convection() const
00283         {
00284             return _convection;
00285         };
00286
00289         inline Function< Point<2, X>, X > *reaction() const
00290         {
00291             return _reaction;
00292         }

```

```

00293
00296         inline Function< Point<2, X>, X > *forcing() const
00297         {
00298             return _forcing;
00299         };
00300
00301     };
00302 };
00303
00304 #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

```

00001 #ifndef EQUATION_HPP
00002 #define EQUATION_HPP
00003
00004 #include <QString>
00005
00006 namespace SemSolver
00007 {
00008     template<int d, class X>
00009     class Equation;
00010 }
00011
00012
00014 namespace SemSolver
00015 {
00018
00023     template<int d, class X>
00024     class Equation
00025     {
00026     public:
00028         enum Type
00029         {
00030             NONE,

```

```

00032         DIFFUSION_CONVECTION_REACTION
00033     };
00034
00036     Equation() {};
00037
00039     virtual ~Equation() {};
00040
00042     virtual Type type() const { return NONE; };
00043
00046     virtual QString mml() const { return ""; };
00047 };
00048 };
00049
00050 #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

```

00001 #ifndef IO_EQUATION_HPP
00002 #define IO_EQUATION_HPP
00003
00004 #include <QFile>
00005
00006 #include <SemSolver/equation.hpp>

```

```

00007
00008 namespace SemSolver
00009 {
00011     namespace IO
00012     {
00014         template<class X>
00015         bool read_equation(QFile *file,
00016                             Equation<2, X> *equation);
00017     };
00018 };
00019
00020 #include <SemSolver/diffusionconvectionreactionequation.hpp>
00021 #include <SemSolver/scriptfunction.hpp>
00022
00023 template<class X>
00024 bool SemSolver::IO::read_equation(QFile *file,
00025                                     Equation<2, X> *equation)
00026 {
00027     #ifdef SEMDEBUG
00028         if(!file->open(QIODevice::ReadOnly))
00029         {
00030             qWarning("SemSolver::IO::readEquation - ERROR : cannot open file.");
00031             return false;
00032         }
00033     #else
00034         file->open(QIODevice::ReadOnly);
00035     #endif
00036     QTextStream input(file);
00037     QStringList values;
00038     values = next_non_empty_line_values(input);
00039     #ifdef SEMDEBUG
00040         if(values.isEmpty())
00041         {
00042             qWarning("SemSolver::IO::readEquation - ERROR : missing type value in fil
00043 e.");
00044             file->close();
00045             return false;
00046         }
00047         if(values.size()>1)
00048         {
00049             qWarning("SemSolver::IO::readEquation - ERROR : too many inputs in first
00050 line of\"
00051 \" file.");
00052             file->close();
00053             return false;
00054         }
00055     #endif
00056     if(values[0]=="DIFFUSION_CONVECTION_REACTION")
00057     {
00058         DiffusionConvectionReactionEquation<2, X> *new_equation =
00059             new DiffusionConvectionReactionEquation<2, X>();
00060         values = next_non_empty_line_values(input);
00061         while(!values.isEmpty())
00062         {
00063             if(values[0]=="DIFFUSION")
00064             {
00065                 #ifdef SEMDEBUG
00066                     if(values.size()!=2)
00067                     {
00068                         qWarning("SemSolver::IO::readEquation - ERROR : wrong number
00069 of inpu\"

```

```

00068             "ts on diffusion line.");
00069             file->close();
00070             return false;
00071         }
00072     #endif
00073     new_equation->setDiffusion(
00074         new ScriptFunction< Point<2, X>, X >(values[1]));
00075     }
00076     else if(values[0]=="CONVECTION")
00077     {
00078     #ifdef SEMDEBUG
00079         if(values.size()!=3)
00080         {
00081             qWarning("SemSolver::IO::readEquation - ERROR : wrong number
of inpu"
00082                 "ts on convection line.");
00083             file->close();
00084             return false;
00085         }
00086     #endif
00087     new_equation->setConvection(
00088         new ScriptFunction< Point<2, X>, Vector<X> >(values.mid(1
, 2)));
00089     }
00090     else if(values[0]=="REACTION")
00091     {
00092     #ifdef SEMDEBUG
00093         if(values.size()!=2)
00094         {
00095             qWarning("SemSolver::IO::readEquation - ERROR : wrong number
of inpu"
00096                 "ts on reaction line.");
00097             file->close();
00098             return false;
00099         }
00100     #endif
00101     new_equation->setReaction(new ScriptFunction< Point<2, X>, X >(va
lues[1]));
00102     }
00103     else if(values[0]=="FORCING")
00104     {
00105     #ifdef SEMDEBUG
00106         if(values.size()!=2)
00107         {
00108             qWarning("SemSolver::IO::readEquation - ERROR : wrong number
of inpu"
00109                 "ts on forcing line.");
00110             file->close();
00111             return false;
00112         }
00113     #endif
00114     new_equation->setForcing(new ScriptFunction< Point<2, X>, X >(val
ues[1]));
00115     }
00116     #ifdef SEMDEBUG
00117     else
00118     {
00119         qWarning("SemSolver::IO::readEquation - ERROR : unknown input lin
e in fi"
00120             "le.");
00121         file->close();
00122         return false;

```

```

00123         };
00124 #endif
00125         values = next_non_empty_line_values(input);
00126     }
00127     delete equation;
00128     equation = new_equation;
00129 }
00130 #ifdef SEMDEBUG
00131     else
00132     {
00133         qWarning("SemSolver::IO::readEquation - ERROR : unknown type value in fil
e.");
00134         file->close();
00135         return false;
00136     }
00137 #endif
00138     file->close();
00139
00140     return true;
00141 };
00142
00143 #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

```

00001 #ifndef FUNCTION_HPP
00002 #define FUNCTION_HPP
00003
00004 namespace SemSolver
00005 {
00006     template<class X, class Y>
00007     class Function;
00008 };
00009
00010
00012 namespace SemSolver
00013 {
00014
00018     template<class X, class Y>

```

```

00019     class Function
00020     {
00021     public:
00022         Function() {};
00023         virtual ~Function() {};
00024         virtual Y evaluate(X const &) const { return Y(); };
00025         virtual QString mml() const { return ""; };
00026     };
00027 #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

```

00001 #ifndef IO_GEOMETRY_HPP
00002 #define IO_GEOMETRY_HPP
00003
00004 #include <QFile>
00005
00006 #include <SemSolver/semgeometry.hpp>
00007
00008 #include <SemSolver/IO/archive.hpp>
00009 #include <SemSolver/IO/pslg.hpp>
00010 #include <SemSolver/IO/subdomains.hpp>
00011
00012 namespace SemSolver
00013 {
00014     namespace IO
00015     {
00016         template<class X>
00017         bool read_geometry(QFile *file,
00018             SemGeometry<2, X> &geometry)
00019         {
00020             Archive archive(file);
00021             if(!archive.openRead())
00022                 return false;
00023             QTemporaryFile *poly_file, *domains_file;
00024             poly_file = new QTemporaryFile();
00025             domains_file = new QTemporaryFile();
00026             if(!archive.extractFile("pslg.poly", poly_file))
00027             {
00028                 delete poly_file;
00029                 delete domains_file;
00030                 return false;
00031             }
00032             if(!archive.extractFile("domains.semsub", domains_file))
00033             {
00034                 delete poly_file;
00035                 delete domains_file;
00036                 return false;
00037             }
00038             if(!archive.closeRead())
00039             {
00040                 delete poly_file;
00041                 delete domains_file;
00042                 return false;
00043             }
00044             PSLG<double> pslg;
00045             if(!SemSolver::IO::read_PSLG(poly_file, pslg))
00046             {
00047                 delete poly_file;
00048                 delete domains_file;
00049                 return false;
00050             }
00051             Polygonation<2, double> sub_domains;
00052             if(!SemSolver::IO::read_subdomains(domains_file, sub_domains))
00053             {
00054                 delete poly_file;
00055                 delete domains_file;
00056                 return false;
00057             }
00058             geometry.setDomain(pslg);
00059             geometry.setSubDomains(sub_domains);
00060             delete poly_file;

```

```

00063     delete domains_file;
00064     return true;
00065 };
00066
00068     bool write_geometry(QFile *pslg_file,
00069                        QFile *domains_file,
00070                        QFile *file);
00071 };
00072 };
00073
00074 #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

```

00001 #ifndef HILBERTSPACE_HPP
00002 #define HILBERTSPACE_HPP
00003
00004 #include <cmath>
00005
00006 #include <vector>
00007
00008 namespace SemSolver
00009 {
00010     template<class Function, class X>
00011     class HilbertSpace
00012     {
00013     public:
00014         std::vector<Function> _base;
00015     };
00016 }

```

```

00021     class Element
00022     : std::vector<X>
00023     {
00024         HilbertSpace<Function, X> const *_space;
00025
00026     public:
00028         Element(const HilbertSpace<Function, X> *space) : _space(space) { };
00029
00031         virtual ~Element() {};
00032     };
00033
00037     virtual Function const *baseFunction(int const &index) const
00038     {
00039 #if SEMDEBUG
00040         if(index<0 || index>=dimension())
00041             qFatal("SemSolver::HilbertSpace::baseFunction - ERROR : index out
of ran"\
00042                 "ge.");
00043 #endif
00044         return &_amp;base[index];
00045     };
00046
00049     virtual X scalarProduct(Element const &,
00050                             Element const &) const
00051     {
00052         return X();
00053     };
00054
00058     virtual inline X norm(Element element)
00059     {
00060         return std::sqrt(scalarProduct(element, element));
00061     };
00062
00065     virtual inline int dimension() const
00066     {
00067         return _base.size();
00068     };
00069
00072     virtual Element projection(Function const *) const
00073     {
00074         return Element(this);
00075     };
00076 };
00077 };
00078
00079 #endif // HILBERTSPACE_HPP

```

7.47 homeomorphism.hpp File Reference

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

Classes

- class [SemSolver::Homeomorphism< X, Y >](#)

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

Namespaces

- namespace [SemSolver](#)

Project main namespace.

7.48 homeomorphism.hpp

```

00001 #ifndef HOMEOMORPHISM_HPP
00002 #define HOMEOMORPHISM_HPP
00003
00004 #include <SemSolver/function.hpp>
00005
00006 namespace SemSolver
00007 {
00008     template<class X, class Y>
00009     class Homeomorphism
00010     : public Function<X,Y>
00011     {
00012     public:
00013         Homeomorphism() {};
00014
00015         virtual ~Homeomorphism() {};
00016
00017         virtual X evaluateInverse(Y const &)
00018         {
00019             return X();
00020         };
00021     };
00022 };
00023
00024 #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

```

00001 #ifndef LUSOLVER_HPP
00002 #define LUSOLVER_HPP
00003
00004 #if defined _WIN32 || defined _WIN64
00005 #    include <SemSolver/math_defines>
00006 #endif
00007
00008 #include <jama_lu.h>
00009
00010 #include <SemSolver/matrix.hpp>
00011 #include <SemSolver/vector.hpp>
00012
00013 namespace SemSolver
00014 {
00015
00016     namespace Solver
00017     {
00018         template<class X>
00023         bool lu_solve(Matrix<X> const &A,
00024                      Vector<X> const &b,
00025                      Vector<X> &x)
00026         {
00027             JAMA::LU<X> lu(A);
00028             #ifdef SEMDEBUG
00029                 if(!lu.isNonsingular())
00030                 {
00031                     qWarning("SemSolver::Solver::lu_solve - ERROR : Matrix A is singu
00032 lar.");
00033                     return false;
00034                 }
00035             #endif
00036             x = lu.solve(b);
00037             return true;
00038         };
00039     };
00040 };
00041
00042 #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

```

00001 #ifndef MATRIX_HPP
00002 #define MATRIX_HPP
00003
00004 namespace SemSolver
00005 {
00006     template<class X>
00007     class Matrix;
00008 };
00009
00010 #include <tnt_array2d.h>
00011 #include <jama_eig.h>
00012
00013 #include <SemSolver/vector.hpp>
00014
00016 namespace SemSolver
00017 {
00019     template<class X>
00020     class Matrix
00021     : public TNT::Array2D<X>
00022     {
00023         typedef TNT::Array2D<X> Base;
00024
00025     public:
00026         Matrix();

```

```

00027
00028         Matrix(int rows, int columns);
00029
00030         Matrix(int rows, int columns, X const &value);
00031
00032         inline int rows() const;
00033
00034         inline int columns() const;
00035
00036         template<class Y>
00037         friend Matrix<Y> operator+(Matrix<Y> const &, Matrix<Y> const &);
00038
00039         Vector<X> realEigenvalues() const;
00040     };
00041
00042     template<class X>
00043     Matrix<X> operator *(Matrix<X> const &mat1, Matrix<X> const &mat2);
00044
00045     template<class X>
00046     Matrix<X> operator +(Matrix<X> const &mat1, Matrix<X> const &mat2);
00047 };
00048
00049
00051 template<class X>
00052 SemSolver::Matrix<X>::Matrix()
00053     : Base()
00054 {
00055 };
00056
00060 template<class X>
00061 SemSolver::Matrix<X>::Matrix(int rows, int columns)
00062     : Base(rows, columns)
00063 {
00064 };
00065
00070 template<class X>
00071 SemSolver::Matrix<X>::Matrix(int rows, int columns, X const &value)
00072     : Base(rows, columns, value)
00073 {
00074 };
00075
00078 template<class X>
00079 inline int SemSolver::Matrix<X>::rows() const
00080 {
00081     return Base::dim1();
00082 };
00083
00086 template<class X>
00087 inline int SemSolver::Matrix<X>::columns() const
00088 {
00089     return Base::dim2();
00090 };
00091
00094 template<class X>
00095 SemSolver::Vector<X> SemSolver::Matrix<X>::realEigenvalues() const
00096 {
00097     Vector<X> eigenvalues;
00098     JAMA::Eigenvalue<double> eig(*this);
00099     eig.getRealEigenvalues(eigenvalues);
00100     return eigenvalues;
00101 };
00102

```

```

00107 template<class X>
00108 SemSolver::Matrix<X> SemSolver::operator *(Matrix<X> const &mat1,
00109                                           Matrix<X> const &mat2)
00110 {
00111     int r1 = mat1.dim1();
00112     int c1 = mat1.dim2();
00113     int r2 = mat2.dim1();
00114     int c2 = mat2.dim2();
00115     #ifdef SEMDEBUG
00116         if(c1 != r2)
00117             qFatal("SemSolver::operator * - ERROR : the number of columns of mat1 mus
t match" \
00118                   " the number of rows of mat2.");
00119     #endif
00120     Matrix<X> product(r1, c2, 0);
00121     for (int i=0; i<r1; i++)
00122         for (int j=0; j<c2; j++)
00123             for (int k=0; k<c1; k++)
00124                 product += mat1[i][k] * mat2[k][j];
00125     return product;
00126 };
00127
00132 template<class X>
00133 SemSolver::Matrix<X> SemSolver::operator +(Matrix<X> const &mat1,
00134                                           Matrix<X> const &mat2)
00135 {
00136     #ifdef SEMDEBUG
00137         int r1 = mat1.dim1();
00138         int c1 = mat1.dim2();
00139         int r2 = mat2.dim1();
00140         int c2 = mat2.dim2();
00141         if(c1 != c2)
00142             qFatal("SemSolver::operator * - ERROR : the number of columns of mat1 mus
t match" \
00143                   "the number of columns of mat2.");
00144         if(r1 != r2)
00145             qFatal("SemSolver::Matrix::operator * - ERROR : the number of rows of mat
1 must " \
00146                   "match the number of the rows of mat2.");
00147     #endif
00148     return mat1+mat2;
00149 };
00150
00151 #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

```

00001 #ifndef MULTIINDEX_HPP
00002 #define MULTIINDEX_HPP
00003
00005 namespace SemSolver
00006 {
00009     template<int N>
00010     class MultiIndex
00011     {
00012     public:
00013         int _indices[N];
00014
00016         MultiIndex()
00017         {
00018             for(int i=0; i<N; ++i)
00019                 _indices[i] = 0;
00020         };
00021
00024         MultiIndex(int const sub_indices[N])
00025         {
00026             for(int i=0; i<N; ++i)
00027                 _indices[i] = sub_indices[i];
00028         };
00029
00032         MultiIndex(MultiIndex<N> const &index)
00033         {
00034             for(int i=0; i<N; ++i)
00035                 _indices[i] = index.subIndex(i);
00036         };
00037
00041         int const &subIndex(int const &index) const
00042         {
00043             if(index<0 || N<=index)
00044                 qFatal("SemSolver::MultiIndex::subIndex - ERROR : index out of range.");
00045             return _indices[index];
00046         };
00047
00051         void setSubIndex(int const &index, int const &sub_index)
00052         {
00053             if(index<0 || N<=index)
00054                 qFatal("SemSolver::MultiIndex::subIndex - ERROR : index out of range.");
00055             _indices[index] = sub_index;
00056         };
00057
00059         struct less
00060         {
00061             bool operator() (MultiIndex<N> const &mi0,
00062                             MultiIndex<N> const &mi1) const
00063             {
00064                 for(int i=0; i<N; ++i)

```

```

00065         {
00066             if(mi0.subIndex(i)<mi1.subIndex(i))
00067                 return true;
00068             if(mi0.subIndex(i)>mi1.subIndex(i))
00069                 return false;
00070         }
00071         return false;
00072     };
00073 };
00074 };
00075 };
00076
00077 #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

```

00001 #ifndef NEXTNONEMPTLINEVALUES_HPP
00002 #define NEXTNONEMPTLINEVALUES_HPP
00003
00004 #include <QStringList>
00005 #include <QTextStream>
00006
00007 namespace SemSolver
00008 {
00010     namespace IO
00011     {
00013         QStringList next_non_empty_line_values(QTextStream &text_stream);
00014     };

```

```

00015 };
00016
00017 #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 > ¶meters)
Read [SemParameters](#) from file.

7.58 parameters.hpp

```

00001 #ifndef IO_PARAMETERS_HPP
00002 #define IO_PARAMETERS_HPP
00003
00004 #include <QFile>
00005 #include <QTextStream>
00006
00007 #include <SemSolver/semparameters.hpp>
00008
00009 namespace SemSolver
00010 {
00011     namespace IO
00012     {
00013         template<class X>
00014         bool read_parameters(QFile *file,
00015                             SemParameters<X> &parameters);
00016     };
00017 };
00018
00019 };
00020
00021

```

```

00022 template<class X>
00023 bool SemSolver::IO::read_parameters(QFile *file,
00024                                     SemParameters<X> &parameters)
00025 {
00026     bool degree = false, tolerance = false, penalty = false;
00027 #ifdef SEMDEBUG
00028     if(!file->open(QIODevice::ReadOnly))
00029     {
00030         qWarning("SemSolver::IO::readParameters - ERROR : cannot open file.");
00031         return false;
00032     }
00033 #else
00034     file->open(QIODevice::ReadOnly);
00035 #endif
00036     QTextStream input(file);
00037     QStringList values;
00038     values = next_non_empty_line_values(input);
00039     while(!values.isEmpty())
00040     {
00041         if(values[0]=="DEGREE")
00042         {
00043 #ifdef SEMDEBUG
00044             if(values.size()!=2)
00045             {
00046                 qWarning("SemSolver::IO::readParameters - ERROR : wrong number of
00047 inpu\"
00048                             \"ts on degree line.\");
00049                 file->close();
00050                 return false;
00051             }
00052 #endif
00053             parameters.setDegree(values[1].toInt(&degree));
00054         }
00055         else if(values[0]=="TOLERANCE")
00056         {
00057 #ifdef SEMDEBUG
00058             if(values.size()!=2)
00059             {
00060                 qWarning("SemSolver::IO::readParameters - ERROR : wrong number of
00061 inpu\"
00062                             \"ts on tolerance line.\");
00063                 file->close();
00064                 return false;
00065             }
00066 #endif
00067             parameters.setTolerance(values[1].toDouble(&tolerance));
00068         }
00069         else if(values[0]=="PENALITY")
00070         {
00071 #ifdef SEMDEBUG
00072             if(values.size()!=2)
00073             {
00074                 qWarning("SemSolver::IO::readParameters - ERROR : wrong number of
00075 inpu\"
00076                             \"ts on tolerance line.\");
00077                 file->close();
00078                 return false;
00079             }
00080 #endif
00081             parameters.setPenalty(values[1].toDouble(&penalty));
00082         }
00083     }
00084 #ifdef SEMDEBUG

```

```

00081         else
00082         {
00083             qWarning("SemSolver::IO::readParameters - ERROR : unknown input line
in fi"\
00084                     "le.");
00085             file->close();
00086             return false;
00087         };
00088 #endif
00089         values = next_non_empty_line_values(input);
00090     }
00091     file->close();
00092     return (degree && tolerance && penalty);
00093 };
00094
00095
00096 #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

```

00001 #ifndef POINT_HPP
00002 #define POINT_HPP
00003
00005 namespace SemSolver
00006 {
00009
00015     template <int d, class X>
00016     class Point;

```

```

00017 };
00018
00019 #include <cmath>
00020
00021 #include <CGAL/Cartesian.h>
00022 #include <CGAL/Filtered_kernel.h>
00023 #include <CGAL/Point_2.h>
00024 #include <CGAL/centroid.h>
00025
00027 namespace SemSolver
00028 {
00029
00031
00037     template<class X>
00038     class Point<2,X>
00039     : public CGAL::Point_2< CGAL::Filtered_kernel< CGAL::Simple_cartesian<X>
00040     > >
00041     {
00043         typedef CGAL::Point_2< CGAL::Filtered_kernel< CGAL::Simple_cartesian<X> >
00044         >
00045         CGAL_Point;
00046     public:
00047
00049         inline Point ();
00050
00052         inline Point (CGAL_Point const &cgal_point);
00053
00057         inline Point (X const &x,
00058                     X const &y);
00059     };
00060
00061 };
00062
00063 template<class X>
00064 inline SemSolver::Point<2, X>::Point ()
00065     : CGAL_Point ()
00066 {};
00067
00068 template<class X>
00069 inline SemSolver::Point<2, X>::Point (const CGAL_Point &cgal_point)
00070     : CGAL_Point (cgal_point)
00071 {};
00072
00073 template<class X>
00074 inline SemSolver::Point<2, X>::Point (const X &x,
00075                                     const X &y)
00076     : CGAL_Point (x, y)
00077 {};
00078
00079 #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

```

00001 #ifndef POINTSBIMAP_HPP
00002 #define POINTSBIMAP_HPP
00003
00004 namespace SemSolver
00005 {
00006     template<int d, class X>
00007     class PointsBimap;
00008 };
00009
00010 #include <SemSolver/point.hpp>
00011 #include <SemSolver/segment.hpp>
00012 #include <SemSolver/pointsmap.hpp>
00013
00015 namespace SemSolver
00016 {
00019
00025     template<int d, class X>
00026     class PointsBimap;
00027
00029
00035     template<class X>
00036     class PointsBimap<2, X>
00037     : private PointsMap<2, X, int>
00038     {
00039         typedef PointsMap<2, X, int> Base;
00040         typedef typename Base::Iterator Iterator;
00041
00042     public:
00043         typedef typename Base::KeyType KeyType;
00044         typedef typename Base::MappedType MappedType;
00045         typedef typename Base::ValueType ValueType;
00046         typedef typename Base::ConstIterator ConstIterator;
00047         typedef typename Base::SizeType SizeType;
00048
00049     private:
00050         int last_id;
00051
00052         inline int new_id();
00053
00054     public:
00055
00056         inline PointsBimap(const X &tolerance);

```

```

00057
00058     Iterator insert(const ValueType &x);
00059
00060     inline MappedType insertPoint(const KeyType &x);
00061
00062     inline const KeyType &point(const MappedType &y) const;
00063
00064     inline const MappedType &id(const KeyType &x) const;
00065
00066     inline void modifyPoint(const MappedType &y, const KeyType &x);
00067
00068     inline void modifyId(const KeyType &x, const MappedType &y);
00069
00070     inline void erasePoint(const KeyType &x);
00071
00072     inline void eraseId(const MappedType &y);
00073
00074     ConstIterator findPoint(const KeyType &x) const;
00075
00076     Iterator findPoint(const KeyType &x);
00077
00078     ConstIterator findId(const MappedType &y) const;
00079
00080     Iterator findId(const MappedType &y);
00081
00082     inline bool hasPoint(const KeyType &x) const;
00083
00084     bool hasId(const MappedType &y) const;
00085
00086     bool hasPointOn(const Segment<2, X> &s) const;
00087
00088     using Base::begin;
00089     using Base::clear;
00090     using Base::has;
00091     using Base::isEmpty;
00092     using Base::end;
00093     using Base::erase;
00094     using Base::size;
00095 };
00096 };
00097
00098 template<class X>
00099 inline int SemSolver::PointsBimap<2, X>::new_id()
00100 {
00101     return ++last_id;
00102 };
00103
00104
00105
00107 template<class X>
00108 inline SemSolver::PointsBimap<2, X>::PointsBimap(const X &tol)
00109     : Base(tol), last_id(0)
00110 {
00111 };
00112
00114 template<class X>
00115 typename SemSolver::PointsBimap<2, X>::Iterator
00116 SemSolver::PointsBimap<2, X>::insert(const ValueType &x)
00117 {
00118     if(!hasId(x.second))
00119         return Base::insert(x);
00120     return Base::end();
00121 };

```



```

00122
00126 template<class X>
00127 inline typename SemSolver::PointsBimap<2, X>::MappedType
00128     SemSolver::PointsBimap<2, X>::insertPoint(const KeyType &x)
00129 {
00130     return Base::insert(ValueType(x, new_id()))->second;
00131 };
00132
00136 template<class X>
00137 const typename SemSolver::PointsBimap<2, X>::KeyType &
00138     SemSolver::PointsBimap<2, X>::point(const MappedType &y) const
00139 {
00140     ConstIterator it = begin();
00141     while(it!=end())
00142     {
00143         if(it->second == y)
00144             break;
00145         else
00146             ++it;
00147     }
00148     #ifdef SEMDEBUG
00149         if(it==end())
00150             qFatal("SemSolver::PointsBimap::point: no mapped value y.");
00151     #endif
00152     return it->first;
00153 };
00154
00158 template<class X>
00159 const typename SemSolver::PointsBimap<2, X>::MappedType &
00160     SemSolver::PointsBimap<2, X>::id(const KeyType &x) const
00161 {
00162     ConstIterator it = find(x);
00163     #ifdef SEMDEBUG
00164         if(it==end())
00165             qFatal("SemSolver::PointsBimap::point: no key value x");
00166     #endif
00167     return it->second;
00168 };
00169
00173 template<class X>
00174 inline void SemSolver::PointsBimap<2, X>::modifyPoint(const MappedType &y,
00175                                                         const KeyType &x)
00176 {
00177     #ifdef SEMDEBUG
00178         if(findPoint(x)!=end())
00179             qFatal("SemSolver:PointsBimap<2, X>::modifyPoint - ERROR : key value x mu
00180 st be u"\
00181                 "nique.");
00182     #endif
00183     eraseId(y);
00184     insert(ValueType(x, y));
00185 };
00186
00189 template<class X>
00190 inline void SemSolver::PointsBimap<2, X>::modifyId(const KeyType &x,
00191                                                     const MappedType &y)
00192 {
00193     #ifdef SEMDEBUG
00194         if(findId(y)!=end())
00195             qFatal("SemSolver:PointsBimap<2, X>::modifyId - ERROR : mapped value y mu
00196 st be u"\
00197                 "nique.");

```

```

00197 #endif
00198     erasePoint(x);
00199     insert(ValueType(x,y));
00200 };
00201
00204 template<class X>
00205 inline void SemSolver::PointsBimap<2, X>::erasePoint(const KeyType &x)
00206 {
00207     Iterator it = findPoint(x);
00208     if(it!=end())
00209         erase(it);
00210 };
00211
00214 template<class X>
00215 inline void SemSolver::PointsBimap<2, X>::eraseId(const MappedType &y)
00216 {
00217     Iterator it = findId(y);
00218     if(it!=end())
00219         erase(it);
00220 };
00221
00225 template<class X>
00226 inline typename SemSolver::PointsBimap<2, X>::ConstIterator
00227     SemSolver::PointsBimap<2, X>::findPoint(const KeyType &x) const
00228 {
00229     return Base::find(x);
00230 };
00231
00235 template<class X>
00236 inline typename SemSolver::PointsBimap<2, X>::Iterator
00237     SemSolver::PointsBimap<2, X>::findPoint(const KeyType &x)
00238 {
00239     return Base::find(x);
00240 };
00241
00245 template<class X>
00246 typename SemSolver::PointsBimap<2, X>::ConstIterator
00247     SemSolver::PointsBimap<2, X>::findId(const MappedType &id) const
00248 {
00249     ConstIterator it = begin();
00250     while(it != end())
00251     {
00252         if(it->second == id)
00253             break;
00254         else
00255             ++it;
00256     }
00257     return it;
00258 };
00259
00263 template<class X>
00264 typename SemSolver::PointsBimap<2, X>::Iterator
00265     SemSolver::PointsBimap<2, X>::findId(const MappedType &id)
00266 {
00267     Iterator it = begin();
00268     while(it != end())
00269     {
00270         if(it->second == id)
00271             break;
00272         else
00273             ++it;
00274     }

```

```

00275     return it;
00276 };
00277
00281 template<class X>
00282 inline bool SemSolver::PointsBimap<2, X>::hasPoint(const KeyType &x) const
00283 {
00284     return findPoint(x) != end();
00285 };
00286
00290 template<class X>
00291 bool SemSolver::PointsBimap<2, X>::hasId(const MappedType &y) const
00292 {
00293     return findId(y) != end();
00294 };
00295
00299 template<class X>
00300 inline bool SemSolver::PointsBimap<2, X>::hasPointOn(const Segment<2, X> &segment
00301 ) const
00302 {
00303     for (ConstIterator it=begin(); it!=end(); ++it)
00304         if(it->first!=segment.source() && it->first!=segment.target() && segment.
00305             has_on(it->first))
00306             return true;
00307     return false;
00308 };
00309
00308 #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

```

00001 #ifndef POINTSMAP_HPP
00002 #define POINTSMAP_HPP
00003
00004 namespace SemSolver
00005 {

```

```

00006     template<int d, class X, class Y>
00007     class PointsMap;
00008 };
00009
00010
00011 #include <SemSolver/point.hpp>
00012
00014 namespace SemSolver
00015 {
00018
00024
00025     template<int d, class X, class Y>
00026     class PointsMap;
00027
00029
00035
00036     template<class X, class Y>
00037     class PointsMap<2, X, Y>
00038     {
00039         typedef QList< QPair< Point<2, X>, Y> > list;
00040
00041     protected:
00042         typedef typename list::iterator Iterator;
00043
00044     public:
00045         typedef Point<2, X> KeyType;
00046         typedef Y MappedType;
00047         typedef typename list::value_type ValueType;
00048         typedef typename list::const_iterator ConstIterator;
00049         typedef typename list::size_type SizeType;
00050
00051     private:
00052         list points;
00053         const double tolerance;
00054         bool less(const KeyType &x, const KeyType &y) const;
00055         inline bool equal(const KeyType &x, const KeyType &y) const;
00056
00057     public:
00058         inline Iterator begin();
00059         inline Iterator end();
00060         inline Iterator find(const KeyType &x);
00061
00062         inline PointsMap(const X &tolerance);
00063         inline ConstIterator begin() const;
00064         inline void clear();
00065         inline bool has(const KeyType &x) const;
00066         inline bool isEmpty() const;
00067         inline ConstIterator end() const;
00068         inline void erase(Iterator position);
00069         inline SizeType erase(const KeyType &x);
00070         inline void erase(Iterator first, Iterator last);
00071         inline ConstIterator find(const KeyType &x) const;
00072         inline Iterator insert(const ValueType &x);
00073         inline Iterator insert(const KeyType &x, const MappedType &y);
00074         Iterator insert(Iterator position, const ValueType &x);
00075         template<class InputIterator>
00076         void insert(InputIterator first, InputIterator last);
00077         inline SizeType size() const;
00078         inline MappedType &operator[] (const KeyType &key);
00079     };
00080 };
00081

```

```

00082 template<class X, class Y>
00083 bool SemSolver::PointsMap<2, X, Y>::less(const KeyType &x, const KeyType &y) const
00084 {
00085     if(x.x()+tolerance < y.x())
00086         return true;
00087     if(x.x()-tolerance > y.x())
00088         return false;
00089     if(x.y()+tolerance < y.y())
00090         return true;
00091     return false;
00092 };
00093
00094 template<class X, class Y>
00095 inline bool SemSolver::PointsMap<2, X, Y>::equal(const KeyType &x, const KeyType
&y) const
00096 {
00097     return !less(x,y) && !less(y,x);
00098 };
00099
00101
00103 template<class X, class Y>
00104 inline SemSolver::PointsMap<2, X, Y>::PointsMap(const X &tol)
00105     : tolerance(tol)
00106 {
00107 };
00108
00109 template<class X, class Y>
00110 inline typename SemSolver::PointsMap<2, X, Y>::Iterator
00111     SemSolver::PointsMap<2, X, Y>::begin()
00112 {
00113     return points.begin();
00114 };
00115
00118 template<class X, class Y>
00119 inline typename SemSolver::PointsMap<2, X, Y>::ConstIterator
00120     SemSolver::PointsMap<2, X, Y>::begin() const
00121 {
00122     return points.begin();
00123 };
00124
00126 template<class X, class Y>
00127 inline void SemSolver::PointsMap<2, X, Y>::clear()
00128 {
00129     points.clear();
00130 };
00131
00135 template<class X, class Y>
00136 inline bool SemSolver::PointsMap<2, X, Y>::has(const KeyType &x) const
00137 {
00138     return find(x)!=end();
00139 };
00140
00143 template<class X, class Y>
00144 inline bool SemSolver::PointsMap<2, X, Y>::isEmpty() const
00145 {
00146     return points.empty();
00147 };
00148
00149 template<class X, class Y>
00150 inline typename SemSolver::PointsMap<2, X, Y>::Iterator
    SemSolver::PointsMap<2, X, Y>::end()

```

```

00151 {
00152     return points.end();
00153 };
00154
00157 template<class X, class Y>
00158 inline typename SemSolver::PointsMap<2, X, Y>::ConstIterator
00159     SemSolver::PointsMap<2, X, Y>::end() const
00160 {
00161     return points.end();
00162 };
00163
00166 template<class X, class Y>
00167 inline void SemSolver::PointsMap<2, X, Y>::erase(Iterator position)
00168 {
00169     points.erase(position);
00170 };
00171
00175 template<class X, class Y>
00176 inline typename SemSolver::PointsMap<2, X, Y>::SizeType
00177     SemSolver::PointsMap<2, X, Y>::erase(const KeyType &x)
00178 {
00179     return points.erase(x);
00180 };
00181
00185 template<class X, class Y>
00186 inline void SemSolver::PointsMap<2, X, Y>::erase(Iterator first, Iterator last)
00187 {
00188     points.erase(first, last);
00189 };
00190
00194 template<class X, class Y>
00195 inline typename SemSolver::PointsMap<2, X, Y>::ConstIterator
00196     SemSolver::PointsMap<2, X, Y>::find(const KeyType &x) const
00197 {
00198     ConstIterator it = begin();
00199     while(it!=end())
00200     {
00201         if(less(it->first, x))
00202             ++it;
00203         else if(less(x, it->first))
00204             it=end();
00205         else
00206             break;
00207     }
00208     return it;
00209 };
00210
00211 template<class X, class Y>
00212 inline typename SemSolver::PointsMap<2, X, Y>::Iterator
00213     SemSolver::PointsMap<2, X, Y>::find(const KeyType &x)
00214 {
00215     Iterator it = begin();
00216     while(it!=end())
00217     {
00218         if(less(it->first, x))
00219             ++it;
00220         else if(less(x, it->first))
00221             it=end();
00222         else
00223             break;
00224     }
00225     return it;

```

```

00226 };
00227
00231 template<class X, class Y>
00232 inline typename SemSolver::PointsMap<2, X, Y>::Iterator
00233     SemSolver::PointsMap<2, X, Y>::insert(const ValueType &x)
00234 {
00235     return insert(begin(), x);
00236 };
00237
00242 template<class X, class Y>
00243 inline typename SemSolver::PointsMap<2, X, Y>::Iterator
00244     SemSolver::PointsMap<2, X, Y>::insert(const KeyType &x,
00245                                           const MappedType &y)
00246 {
00247     return insert(ValueType(x,y));
00248 };
00249
00254 template<class X, class Y>
00255 typename SemSolver::PointsMap<2, X, Y>::Iterator
00256     SemSolver::PointsMap<2, X, Y>::insert(Iterator it, const ValueType &x)
00257 {
00258     while(it!=begin())
00259     {
00260         if(less(x.first, it->first))
00261             --it;
00262         else
00263             break;
00264     }
00265     while(it!=end())
00266     {
00267         if(less(it->first, x.first))
00268             ++it;
00269         else
00270             break;
00271     }
00272     if(it==end() || less(x.first, it->first))
00273         return points.insert(it, x);
00274     return end();
00275 };
00276
00280 template<class X, class Y>
00281 template<class InputIterator>
00282 void SemSolver::PointsMap<2, X, Y>::insert(InputIterator first, InputIterator las
00283 t)
00284 {
00285     Iterator it = begin();
00286     while(first!=last)
00287         it = insert(it, *first++);
00288     insert(it, last);
00289 };
00292 template<class X, class Y>
00293 inline typename SemSolver::PointsMap<2, X, Y>::SizeType
00294     SemSolver::PointsMap<2, X, Y>::size() const
00295 {
00296     return points.size();
00297 };
00298
00300
00305
00306
00307 template<class X, class Y>

```

```

00308 inline typename SemSolver::PointsMap<2, X, Y>::MappedType &
00309     SemSolver::PointsMap<2, X, Y>::operator[] (const KeyType &x)
00310 {
00311     return insert(ValueType(x, MappedType()))->second;
00312 };
00313
00314 #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

```

00001 #ifndef POINTSSET_HPP
00002 #define POINTSSET_HPP
00003
00004 namespace SemSolver
00005 {
00006     template<int d, class X>
00007     class PointsSet;
00008 };
00009
00010
00011 #include <SemSolver/point.hpp>
00012
00014 namespace SemSolver
00015 {
00018
00024     template<int d, class X>
00025     class PointsSet;
00026
00028
00034     template<class X>
00035     class PointsSet<2, X>
00036     {
00037         typedef QList< Point<2, X> > list;
00038
00039     protected:

```



```

00040         typedef typename list::iterator Iterator;
00041
00042     public:
00043         typedef Point<2, X> KeyType;
00044         typedef Point<2, X> MappedType;
00045         typedef typename list::value_type ValueType;
00046         typedef typename list::const_iterator ConstIterator;
00047         typedef typename list::size_type SizeType;
00048
00049     private:
00050         list points;
00051         const double tolerance;
00052         bool less(const KeyType &x, const KeyType &y);
00053         inline bool equal(const KeyType &x, const KeyType &y);
00054
00055     protected:
00056         inline Iterator begin();
00057         inline Iterator end();
00058         inline Iterator find(const KeyType &x);
00059
00060     public:
00061         inline PointsSet(const X &tolerance);
00062         inline ConstIterator begin() const;
00063         inline void clear();
00064         inline bool has(const KeyType &x) const;
00065         inline bool isEmpty() const;
00066         inline ConstIterator end() const;
00067         inline void erase(Iterator position);
00068         inline SizeType erase(const KeyType &x);
00069         inline void erase(Iterator first, Iterator last);
00070         inline ConstIterator find(const KeyType &x) const;
00071         inline Iterator insert(const ValueType &x);
00072         Iterator insert(Iterator position, const ValueType &x);
00073         template<class InputIterator>
00074         void insert(InputIterator first, InputIterator last);
00075         inline SizeType size() const;
00076         inline MappedType &operator[](const KeyType &key);
00077     };
00078 };
00079
00080 template<class X>
00081 bool SemSolver::PointsSet<2, X>::less(const KeyType &x, const KeyType &y)
00082 {
00083     if(x.x()+tolerance < y.x())
00084         return true;
00085     if(x.x()-tolerance > y.x())
00086         return false;
00087     if(x.y()+tolerance < y.y())
00088         return true;
00089     return false;
00090 };
00091
00092 template<class X>
00093 inline bool SemSolver::PointsSet<2, X>::equal(const KeyType &x, const KeyType &y)
00094 {
00095     return !less(x,y) && !less(y,x);
00096 };
00097
00098
00099
00100 template<class X>
00101 inline SemSolver::PointsSet<2, X>::PointsSet(const X &tol)

```

```

00102         : tolerance(tol)
00103 {
00104 };
00105
00106 template<class X>
00107 inline typename SemSolver::PointsSet<2, X>::Iterator
00108     SemSolver::PointsSet<2, X>::begin()
00109 {
00110     return points.begin();
00111 };
00112
00115 template<class X>
00116 inline typename SemSolver::PointsSet<2, X>::ConstIterator
00117     SemSolver::PointsSet<2, X>::begin() const
00118 {
00119     return points.begin();
00120 };
00121
00123 template<class X>
00124 inline void SemSolver::PointsSet<2, X>::clear()
00125 {
00126     points.clear();
00127 };
00128
00132 template<class X>
00133 inline bool SemSolver::PointsSet<2, X>::has(const KeyType &x) const
00134 {
00135     return find(x) != end();
00136 };
00137
00140 template<class X>
00141 inline bool SemSolver::PointsSet<2, X>::isEmpty() const
00142 {
00143     return points.empty();
00144 };
00145
00146 template<class X>
00147 inline typename SemSolver::PointsSet<2, X>::Iterator
00148     SemSolver::PointsSet<2, X>::end()
00149 {
00150     return points.end();
00151 };
00152
00154 template<class X>
00155 inline typename SemSolver::PointsSet<2, X>::ConstIterator
00156     SemSolver::PointsSet<2, X>::end() const
00157 {
00158     return points.end();
00159 };
00160
00163 template<class X>
00164 inline void SemSolver::PointsSet<2, X>::erase(Iterator position)
00165 {
00166     points.erase(position);
00167 };
00168
00172 template<class X>
00173 inline typename SemSolver::PointsSet<2, X>::SizeType
00174     SemSolver::PointsSet<2, X>::erase(const KeyType &x)
00175 {
00176     return points.erase(x);
00177 };

```

```

00178
00182 template<class X>
00183 inline void SemSolver::PointsSet<2, X>::erase(Iterator first, Iterator last)
00184 {
00185     points.erase(first, last);
00186 };
00187
00191 template<class X>
00192 inline typename SemSolver::PointsSet<2, X>::ConstIterator
00193     SemSolver::PointsSet<2, X>::find(const KeyType &x) const
00194 {
00195     ConstIterator it = begin();
00196     while(it!=end())
00197     {
00198         if(less(*it, x))
00199             ++it;
00200         else if(less(x, *it))
00201             it=end();
00202         else
00203             break;
00204     }
00205     return it;
00206 };
00207
00208 template<class X>
00209 inline typename SemSolver::PointsSet<2, X>::Iterator
00210     SemSolver::PointsSet<2, X>::find(const KeyType &x)
00211 {
00212     Iterator it = begin();
00213     while(it!=end())
00214     {
00215         if(less(*it, x))
00216             ++it;
00217         else if(less(x, *it))
00218             it=end();
00219         else
00220             break;
00221     }
00222     return it;
00223 };
00224
00228 template<class X>
00229 inline typename SemSolver::PointsSet<2, X>::Iterator
00230     SemSolver::PointsSet<2, X>::insert(const ValueType &x)
00231 {
00232     return insert(begin(), x);
00233 };
00234
00239 template<class X>
00240 typename SemSolver::PointsSet<2, X>::Iterator
00241     SemSolver::PointsSet<2, X>::insert(Iterator it, const ValueType &x)
00242 {
00243     while(it!=begin())
00244     {
00245         if(less(x, *it))
00246             --it;
00247         else
00248             break;
00249     }
00250     while(it!=end())
00251     {
00252         if(less(*it, x) )

```

```

00253         ++it;
00254     else
00255         break;
00256     }
00257     if(it==end() || less(x, *it))
00258         return points.insert(it, x);
00259     return end();
00260 };
00261
00265 template<class X>
00266 template<class InputIterator>
00267 void SemSolver::PointsSet<2, X>::insert(InputIterator first, InputIterator last)
00268 {
00269     Iterator it = begin();
00270     while(first!=last)
00271         it = insert(it, *first++);
00272     insert(it, last);
00273 };
00274
00277 template<class X>
00278 inline typename SemSolver::PointsSet<2, X>::SizeType
    SemSolver::PointsSet<2, X>::size() const
00279 {
00280     return points.size();
00281 };
00282
00284
00289
00290
00291 template<class X>
00292 inline typename SemSolver::PointsSet<2, X>::MappedType &
00293     SemSolver::PointsSet<2, X>::operator[] (const KeyType &x)
00294 {
00295     return insert(ValueType(x, MappedType()))->second;
00296 };
00297
00298 #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

```

00001 #ifndef POLYGON_HPP
00002 #define POLYGON_HPP
00003
00005 namespace SemSolver
00006 {
00009
00015     template<int d, class X>
00016     class Polygon;
00017 };
00018
00019 #include <CGAL/Simple_cartesian.h>
00020 #include <CGAL/Filtered_kernel.h>
00021 #include <CGAL/Polygon_2.h>
00022
00023 #include <SemSolver/point.hpp>
00024
00026 namespace SemSolver
00027 {
00028
00030
00036     template<class X>
00037     class Polygon<2, X>
00038     : public CGAL::Polygon_2< CGAL::Filtered_kernel< CGAL::Simple_cartesian<X
00039 > > >
00039     {
00040
00042         typedef CGAL::Point_2< CGAL::Filtered_kernel< CGAL::Simple_cartesian<X> >
00043 >
00043         CGAL_point;
00044
00046         typedef CGAL::Polygon_2< CGAL::Filtered_kernel< CGAL::Simple_cartesian<X>
00047 > >
00047         CGAL_polygon;
00048
00049     public:
00050
00052         inline Polygon();
00053
00057         inline Polygon(CGAL_point const *first,
00058             CGAL_point const *last);
00059
00063         inline bool contains(Point<2, X> const &point) const;
00064
00065     };
00066 };
00067 };
00068
00069 template<class X>
00070 inline SemSolver::Polygon<2, X>::Polygon()
00071 : CGAL_polygon()
00072 {};

```

```

00073
00074 template<class X>
00075 inline SemSolver::Polygon<2, X>::Polygon(const CGAL_point *first,
00076                                           const CGAL_point *last)
00077                                           : CGAL_polygon(first, last)
00078 {};
00079
00080 template<class X>
00081 inline bool SemSolver::Polygon<2, X>::contains(const Point<2, X> &point) const
00082 {
00083     return bounded_side(point) != CGAL::ON_UNBOUNDED_SIDE;
00084 };
00085
00086 #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

```

00001 #ifndef POLYGONATION_HPP
00002 #define POLYGONATION_HPP
00003
00005 namespace SemSolver
00006 {
00009
00015     template<int d, class X>
00016     class Polygonation;
00017 };
00018

```

```

00019 #include <vector>
00020
00021 #include <SemSolver/polygon.hpp>
00022 #include <SemSolver/point.hpp>
00023
00025 namespace SemSolver
00026 {
00027
00029
00035     template<class X>
00036     class Polygonation<2,X>
00037     {
00038
00039     public:
00040
00042
00044         class Element
00045         {
00046
00047         private:
00048             Polygon<2, X>    polygon;
00049             std::vector<int> neighbours_vector;
00050
00051         public:
00053             inline Element();
00054
00058             inline Element(Polygon<2, X> const &geometry,
00059                             std::vector<int> const &neighbours);
00060
00063             inline Polygon<2, X> const &geometry() const;
00064
00067
00069             unsigned vertexPosition(Point<2, X> const &vertex) const;
00070
00072             inline void clear();
00073
00077             inline Point<2, X> vertex(int const &index) const;
00078
00081             inline int size() const;
00082
00085             inline typename Polygon<2, X>::Vertex_const_iterator verticesBegin()
00086             const;
00087
00089             inline typename Polygon<2, X>::Vertex_const_iterator verticesEnd() co
00090             nst;
00091
00093             inline std::vector<int>::const_iterator neighboursBegin() const;
00094
00097             inline std::vector<int>::const_iterator neighboursEnd() const;
00098
00102             inline void setGeometry(Point<2,X> const *first,
00103                                     Point<2,X> const *last);
00104
00108             inline void setNeighbour(const unsigned &index,
00109                                     int const &id);
00110
00114             inline int neighbour(const unsigned &index) const;
00115
00119             inline bool contains(Point<2, X> const &point) const;
00120
00121         };
00122

```

```

00123     private:
00124         std::vector<Element> elements;
00125
00126     public:
00129         bool isQuadrangulation() const;
00130
00132
00134         void refine();
00135
00137         inline void clear();
00138
00142         inline const Element &element(const unsigned &index) const;
00143
00146         inline unsigned size() const;
00147
00151         inline void addElement(Polygon<2, X> const &polygon,
00152                                std::vector<int> const &neighbours);
00153
00157         std::vector<unsigned> elementIndicesAt(Point<2, X> const &point) const;
00158     };
00159 };
00160
00161 template<class X>
00162 bool SemSolver::Polygonation<2, X>::isQuadrangulation() const
00163 {
00164     for(typename std::vector<Element>::const_iterator it = elements.begin();
00165         it != elements.end(); ++it)
00166     {
00167         if(it->size() != 4)
00168             return false;
00169     }
00170     return true;
00171 };
00172
00173 template<class X>
00174 void SemSolver::Polygonation<2, X>::refine()
00175 {
00176     int *subelements_first = new int[size()];
00177     subelements_first[0]=0;
00178     for(unsigned i=0; i<size()-1; ++i)
00179         subelements_first[i+1]=subelements_first[i]+element(i).size();
00180     std::vector<Element> subelements;
00181     for(unsigned i=0; i<size(); ++i)
00182     {
00183         Element const &old_element = element(i);
00184         Point<2,X> cent = centroid(old_element.verticesBegin(),
00185                                   old_element.verticesEnd());
00186         Point<2,X> vertices[4];
00187         Polygonation::Element subelement;
00188         int n = old_element.size();
00189         for(int j=0; j<n; ++j)
00190         {
00191             vertices[0] = old_element.vertex(j);
00192             vertices[1] = midpoint(old_element.vertex(j),
00193                                   old_element.vertex((j+1)%n));
00194             vertices[2] = cent;
00195             vertices[3] = midpoint(old_element.vertex(j),
00196                                   old_element.vertex((j+n-1)%n));
00197             subelement.setGeometry(vertices, vertices+4);
00198             if(old_element.neighbour(j)<0) // border
00199                 subelement.setNeighbour(0,old_element.neighbour(j));
00200             else

```



```

00201         {
00202             int neigh = old_element.neighbour(j)-1;
00203             int neigh_first = subelements_first[neigh];
00204             element(neigh);
00205             old_element.vertex(j);
00206             int pos = element(neigh).vertexPosition(old_element.vertex(j));
00207             subelement.setNeighbour(0,neigh_first+pos+1);
00208         }
00209         if(old_element.neighbour((j+1)%n)<0)
00210             subelement.setNeighbour(1, old_element.neighbour((j+1)%n));
00211         else
00212         {
00213             int neigh = old_element.neighbour((j+1)%n)-1;
00214             int neigh_first = subelements_first[neigh];
00215             int pos = element(neigh).vertexPosition(old_element.vertex(j));
00216             subelement.setNeighbour(1, neigh_first+pos+1);
00217         }
00218         subelement.setNeighbour(2,subelements_first[i]+(j+1)%n+1);
00219         subelement.setNeighbour(3,subelements_first[i]+(j+n-1)%n+1);
00220         subelements.push_back(subelement);
00221     }
00222 }
00223 elements.clear();
00224 elements = subelements;
00225 };
00226
00227 template<class X>
00228 inline void SemSolver::Polygonation<2, X>::clear()
00229 {
00230     elements.clear();
00231 };
00232
00233 template<class X>
00234 inline const typename SemSolver::Polygonation<2, X>::Element
00235     &SemSolver::Polygonation<2, X>::element(const unsigned &index) const
00236 {
00237     #ifdef SEMDEBUG
00238         if(index>=elements.size())
00239             qFatal("SemSolver::Polygonation::element - ERROR : index out of bounds.");
00240     #endif //SEMDEBUG
00241     return elements[index];
00242 };
00243
00244 template<class X>
00245 inline unsigned SemSolver::Polygonation<2, X>::size() const
00246 {
00247     return elements.size();
00248 };
00249
00250 template<class X>
00251 inline void SemSolver::Polygonation<2, X>::addElement(Polygon<2, X> const &polygo
n,
00252                                                         std::vector<int> const &nei
ghbours)
00253 {
00254     elements.push_back(Element(polygon, neighbours));
00255 };
00256
00257 template<class X>
00258 std::vector<unsigned> SemSolver::Polygonation<2, X>::elementIndicesAt(
00259     SemSolver::Point<2, X> const &point) const

```

```

00260 {
00261     std::vector<unsigned> indices;
00262     for(unsigned i=0; i<size(); ++i)
00263         if(element(i).contains(point))
00264             indices.push_back(i);
00265     return indices;
00266 };
00267
00268 template<class X>
00269 inline SemSolver::Polygonation<2, X>::Element::Element()
00270 {};
00271
00272 template<class X>
00273 inline SemSolver::Polygonation<2, X>::Element::Element(Polygon<2, X> const &geome
try,
00274                                                         std::vector<int> const &ne
ighbours)
00275 {
00276     polygon = geometry;
00277     neighbours_vector = neighbours;
00278 };
00279
00280 template<class X>
00281 inline SemSolver::Polygon<2, X> const &SemSolver::Polygonation<2, X>::Element::ge
ometry()
00282     const
00283 {
00284     return polygon;
00285 }
00286
00287 template<class X>
00288 unsigned SemSolver::Polygonation<2, X>::Element::vertexPosition(Point<2, X> const
&vertex)
00289     const
00290 {
00291     for(unsigned i=0; i<polygon.size(); ++i)
00292     {
00293         if(polygon.vertex(i) == vertex)
00294         {
00295             return i;
00296         }
00297     }
00298     return -1;
00299 };
00300
00301 template<class X>
00302 inline void SemSolver::Polygonation<2, X>::Element::clear()
00303 {
00304     polygon.clear();
00305     neighbours_vector.clear();
00306 };
00307
00308 template<class X>
00309 inline SemSolver::Point<2, X> SemSolver::Polygonation<2, X>::Element::vertex(
int const &index) const
00310 {
00311     return polygon.vertex(index);
00312 };
00313
00314 template<class X>
00315 inline int SemSolver::Polygonation<2, X>::Element::size() const
00316 {
00317

```

```

00318     return polygon.size();
00319 };
00320
00321 template<class X>
00322 inline typename SemSolver::Polygon<2, X>::Vertex_const_iterator
00323     SemSolver::Polygonation<2, X>::Element::verticesBegin() const
00324 {
00325     return polygon.vertices_begin();
00326 };
00327
00328 template<class X>
00329 inline typename SemSolver::Polygon<2, X>::Vertex_const_iterator
00330     SemSolver::Polygonation<2, X>::Element::verticesEnd() const
00331 {
00332     return polygon.vertices_end();
00333 };
00334
00335 template<class X>
00336 inline std::vector<int>::const_iterator
00337     SemSolver::Polygonation<2, X>::Element::neighboursBegin() const
00338 {
00339     return neighbours_vector.begin();
00340 };
00341
00342 template<class X>
00343 inline std::vector<int>::const_iterator
00344     SemSolver::Polygonation<2, X>::Element::neighboursEnd() const
00345 {
00346     return neighbours_vector.end();
00347 };
00348
00349 template<class X>
00350 inline void SemSolver::Polygonation<2, X>::Element::setGeometry(Point<2,X> const
    *first,
00351                                     Point<2,X> const
    *last)
00352 {
00353     polygon = Polygon<2,X>(first, last);
00354     neighbours_vector.resize(polygon.size(), 0);
00355 };
00356
00357 template<class X>
00358 inline void SemSolver::Polygonation<2, X>::Element::setNeighbour(const unsigned &
    index,
00359                                     int const &id)
00360 {
00361     #ifdef SEMDEBUG
00362         if(index>neighbours_vector.size())
00363             qFatal("SemSolver::Polygonation::Element::setNeighbour - ERROR : index ou
    t of bo\"
00364                 "unds.");
00365     #endif //SEMDEBUG
00366     neighbours_vector[index] = id;
00367 };
00368
00369 template<class X>
00370 inline int SemSolver::Polygonation<2, X>::Element::neighbour(const unsigned &inde
    x)
00371     const
00372 {
00373     #ifdef SEMDEBUG
00374         if(index>neighbours_vector.size())

```

```

00375         qFatal("SemSolver::Polygonation::Element::neighbour - ERROR : index out o
f bound" \
00376             "s.");
00377 #endif //SEMDEBUG
00378     return neighbours_vector[index];
00379 };
00380
00381 template<class X>
00382 inline bool SemSolver::Polygonation<2, X>::Element::contains(Point<2, X> const &p
oint)
00383     const
00384 {
00385     return polygon.contains(point);
00386 };
00387
00388 #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

```

00001 #ifndef POLYGONWITHHOLES_HPP
00002 #define POLYGONWITHHOLES_HPP
00003
00004 namespace SemSolver
00005 {
00006     template <int d, class X>
00007     class PolygonWithHoles;
00008 }
00009 #include <CGAL/Cartesian.h>

```

```

00020 #include <CGAL/Filtered_kernel.h>
00021 #include <CGAL/Polygon_with_holes_2.h>
00022
00023 #include <SemSolver/polygon.hpp>
00024
00026 namespace SemSolver
00027 {
00028
00030
00036     template<class X>
00037     class PolygonWithHoles<2,X>
00038     : public CGAL::Polygon_with_holes_2<
00039         CGAL::Filtered_kernel< CGAL::Simple_cartesian<X> > >
00040     {
00042         typedef CGAL::Polygon_with_holes_2<
00043             CGAL::Filtered_kernel< CGAL::Simple_cartesian<X> > >
00044             CGAL_Polygon_with_holes;
00045     public:
00046         inline CGAL_Polygon_with_holes const &cgal()
00047         {
00048             return *this;
00049         };
00050     };
00051 };
00052
00053 #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 handling the mathematical concept of polynomials over X.

Namespaces

- namespace [SemSolver](#)
Project main namespace.

7.74 polynomial.hpp

```

00001 #ifndef POLYNOMIAL_HPP
00002 #define POLYNOMIAL_HPP

```

```

00003
00004 #include <vector>
00005
00006 #include <SemSolver/function.hpp>
00007 #include <SemSolver/matrix.hpp>
00008
00010 namespace SemSolver
00011 {
00013     template<class X>
00014     class Polynomial
00015     {
00016     private:
00017         // members
00018         std::vector<X> _coefficients;
00019
00020     public:
00022         Polynomial()
00023             : _coefficients(1,0.)
00024         {
00025         }
00026
00029         Polynomial(X const &scalar)
00030             : _coefficients(1,scalar)
00031         {
00032         }
00033
00036         Polynomial(std::vector<X> const &coefficients)
00037             : _coefficients(coefficients)
00038         {
00039         }
00040
00043         Polynomial(Polynomial const &poly)
00044             : _coefficients(poly._coefficients)
00045         {
00046         }
00047
00049         ~Polynomial()
00050         {};
00051
00054         Polynomial & operator = (Polynomial const &poly)
00055         {
00056             _coefficients = poly._coefficients;
00057             return *this;
00058         }
00059
00062         Polynomial & operator += (Polynomial const &poly)
00063         {
00064             int deg = poly.degree();
00065             if(deg>degree())
00066                 _coefficients.resize(deg+1,0.);
00067             for(int i=0; i<=deg; ++i)
00068                 _coefficients[i] += poly._coefficients[i];
00069             return *this;
00070         }
00071
00074         Polynomial & operator -= (Polynomial const &poly)
00075         {
00076             int deg = poly.degree();
00077             if(deg>degree())
00078                 _coefficients.resize(deg+1,0.);
00079             for(int i=0; i<=deg; ++i)
00080                 _coefficients[i] -= poly._coefficients[i];

```

```

00081         return *this;
00082     }
00083
00086     Polynomial & operator *= (Polynomial const &poly)
00087     {
00088         int deg = degree();
00089         int deg_ = poly.degree();
00090         std::vector<double> product;
00091         product.resize(deg+deg_+1,0.);
00092         for(int i=0; i<=deg; ++i)
00093             for(int j=0; j<=deg_; ++j)
00094                 product[i+j] += _coefficients[i]*poly._coefficients[j];
00095         _coefficients = product;
00096         return *this;
00097     }
00098
00101     Polynomial & operator /= (Polynomial const &poly)
00102     {
00103         int deg = degree();
00104         int deg_ = poly.degree();
00105         Polynomial quotient, residual;
00106         residual = *this;
00107         {
00108             for(int i=deg; i>=deg_; --i)
00109             {
00110                 if (i>=0 && residual._coefficients[i])
00111                 {
00112                     Polynomial temp;
00113                     temp._coefficients.resize(deg-deg_+1,double(0));
00114                     temp._coefficients[i-deg_] =
00115                         residual._coefficients[i]/poly._coefficients[deg_
00116 ];
00117                     quotient+=temp;
00118                     residual-=poly*temp;
00119                 }
00120             }
00121             _coefficients = quotient._coefficients;
00122             return *this;
00123         }
00124
00127     Polynomial & operator %= (const Polynomial &poly)
00128     {
00129         int deg = degree();
00130         int deg_ = poly.degree();
00131         Polynomial quotient, residual;
00132         residual = *this;
00133         {
00134             for(int i=deg; i>=deg_; --i)
00135             {
00136                 if (residual._coefficients[i])
00137                 {
00138                     Polynomial temp;
00139                     temp._coefficients.resize(deg-deg_+1,0.);
00140                     temp._coefficients[i-deg_] =
00141                         residual._coefficients[i]/poly._coefficients[deg_
00142 ];
00143                     quotient+=temp;
00144                     residual-=poly*temp;
00145                 }
00146             }

```

```

00147         _coefficients = residual._coefficients;
00148         return *this;
00149     }
00150
00152     Polynomial operator + () const
00153     {
00154         return *this;
00155     }
00156
00158     Polynomial operator - () const
00159     {
00160         Polynomial result(*this);
00161         for(std::vector<double>::iterator it=result._coefficients.begin();
00162             it!=result._coefficients.end(); ++it)
00163             *it = -*it;
00164         return result;
00165     }
00166
00169     Polynomial operator + (const Polynomial &poly) const
00170     {
00171         Polynomial sum = *this;
00172         sum += poly;
00173         return sum;
00174     }
00175
00178     Polynomial operator - (const Polynomial &poly) const
00179     {
00180         Polynomial difference = *this;
00181         difference -= poly;
00182         return difference;
00183     }
00184
00187     Polynomial operator * (const Polynomial &poly) const
00188     {
00189         Polynomial product = *this;
00190         product *= poly;
00191         return product;
00192     }
00193
00196     Polynomial operator / (const Polynomial &poly) const
00197     {
00198         Polynomial quotient = *this;
00199         quotient /= poly;
00200         return quotient;
00201     }
00202
00205     Polynomial operator % (const Polynomial &poly) const
00206     {
00207         Polynomial residual = *this;
00208         residual %= poly;
00209         return residual;
00210     }
00211
00213     bool operator == (const Polynomial &poly) const
00214     {
00215         if(degree()!=poly.degree())
00216             return false;
00217         else
00218             for(int i=0; i<degree(); ++i)
00219                 if(_coefficients[i]!=poly._coefficients[i])
00220                     return 0;
00221         return 1;

```



```

00222     }
00223
00225     bool operator != (const Polynomial &poly) const;
00226
00227     double operator () (const X &variable) const
00228     {
00229         double value=0;
00230         for(int i=degree(); i>0; --i)
00231             value = (value + _coefficients[i]) * variable;
00232         return value+_coefficients[0];
00233     }
00234
00237     inline double coefficient(const int &order) const
00238     {
00239 #ifdef SEMDEBUG
00240         if(order<0 || order>degree())
00241             qFatal("SemSolver::Polynomial::coefficient - ERROR : order must be
00242 e a non"\
00243                 " negative value not greater than degree");
00244 #endif
00245         return _coefficients[order];
00246     }
00248     int degree() const
00249     {
00250         int deg = _coefficients.size()-1;
00251         while(deg>=0 && !_coefficients[deg])
00252             --deg;
00253         return deg;
00254     }
00255
00259     inline void setCoefficient(const int &order, X const &coefficient)
00260     {
00261 #ifdef SEMDEBUG
00262         if(order<0 || (long)order>=(long)_coefficients.size() )
00263             qFatal("SemSolver::Polynomial::setCoefficient - ERROR : order "\
00264                 "out of range.");
00265 #endif
00266         _coefficients[order]=coefficient;
00267     };
00268
00271     inline void setDegree(const int &degree)
00272     {
00273 #ifdef SEMDEBUG
00274         if(degree<-1)
00275             qFatal("SemSolver::Polynomial::setDegree - ERROR : degree must be
00276 not le"\
00277                 "ss than -1.");
00278 #endif
00279         _coefficients.resize(degree+1,0.);
00280     };
00282     Polynomial derivative() const
00283     {
00284         int deg = degree();
00285         if(deg < 1)
00286             return Polynomial(0.);
00287         std::vector<double> derivative;
00288         derivative.resize(deg,0.);
00289         for(int i=0; i<deg; ++i)
00290             derivative[i] = (i+1)*_coefficients[i+1];
00291         return Polynomial(derivative);

```

```

00292     }
00293
00296     Polynomial ruffini(X const &zero) const
00297     {
00298         std::vector<double> result;
00299         int deg=degree();
00300         result.resize(deg+1,double(0.));
00301         for(int i=deg; i>0; --i)
00302             result[i-1] = _coefficients[i] + zero * result[i];
00303         return Polynomial(result);
00304     }
00305
00307     std::vector<X> zeros() const
00308     {
00309         int deg = degree();
00310         Matrix<X> companion(deg,deg,0.);
00311         for(int i=0; i<deg-1; ++i)
00312         {
00313             companion[i+1][i] = 1;
00314             companion[i][deg-1] = -_coefficients[i]/_coefficients[deg];
00315         }
00316         Vector<X> eigenvalues(companion.realEigenvalues());
00317         std::vector<double> zeros;
00318         for(int i=0; i< deg; ++i)
00319             zeros.push_back(eigenvalues[i]);
00320         return zeros;
00321     }
00322 };
00323 }
00324
00325 #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

```

00001 #ifndef POLYNOMIALFUNCTION_HPP
00002 #define POLYNOMIALFUNCTION_HPP
00003
00004 #include <SemSolver/polynomial.hpp>
00005 #include <SemSolver/function.hpp>
00006 #include <SemSolver/point.hpp>
00007
00008 namespace SemSolver
00009 {
00010
00011     template<int d, class X>
00012     class PolynomialFunction : public Function< Point<d,X>, X >
00013     {
00014     private:
00015         Polynomial<X> _polynomials[d];
00016     public:
00017         PolynomialFunction()
00018         {
00019             for(int i=0; i<d; ++i)
00020                 _polynomials[i] = 0.;
00021         };
00022         PolynomialFunction(Polynomial<X> const *polynomials)
00023         {
00024             for(int i=0; i<d; ++i)
00025                 _polynomials[i] = polynomials[i];
00026         };
00027         PolynomialFunction(PolynomialFunction<d,X> const &polynomial_function)
00028         {
00029             for(int i=0; i<d; ++i)
00030             {
00031                 _polynomials[i] = polynomial_function.polynomial(i);
00032             }
00033         };
00034         ~PolynomialFunction()
00035         {};
00036         PolynomialFunction<d,X> &operator=(PolynomialFunction<d,X> const &polynomial_function)
00037         {
00038             for(int i=0; i<d; ++i)
00039                 _polynomials[i] = polynomial_function.polynomial(i);
00040             return *this;
00041         };
00042         Polynomial<X> const &polynomial(int const &index) const
00043         {
00044             #ifdef SEMDEBUG
00045                 if(index<0 || index>=d)
00046                     qFatal("SemSolver::PolynomialFunction::polynomial - ERROR : index
out of" \
00047                             "range");
00048             #endif
00049             return _polynomials[index];
00050         };

```

```

00074
00076         void setPolynomial(int const &index, Polynomial<X> const &poly)
00077         {
00078 #ifdef SEMDEBUG
00079             if(index<0 || index>=d)
00080                 qFatal("SemSolver::PolynomialFunction::setPolynomial ERROR : inde
x out o"\
00081                     "f range");
00082 #endif
00083             _polynomials[index] = poly;
00084         };
00085
00087     X evaluate(Point<d,X> const &x) const
00088     {
00089         double result = 1;
00090         for(int i=0; i<d; ++i)
00091             result *= polynomial(i)(x.cartesian(i));
00092         return result;
00093     };
00094 };
00095 };
00096
00097 #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

```

00001 #ifndef PROBLEM_HPP
00002 #define PROBLEM_HPP

```

```

00003
00004 namespace SemSolver
00005 {
00006     template<int d, class X>
00007     class Problem;
00008 }
00009
00010 #include <SemSolver/semgeometry.hpp>
00011 #include <SemSolver/equation.hpp>
00012 #include <SemSolver/boundaryconditions.hpp>
00013 #include <SemSolver/semparameters.hpp>
00014
00016 namespace SemSolver
00017 {
00020     template<int d, class X>
00021     class Problem
00022     {
00023     public:
00024         const SemGeometry<d, X> *_geometry;
00025         const Equation<d, X> *_equation;
00026         const BoundaryConditions<d, X> *_boundary_conditions;
00027         const SemParameters<X> *_parameters;
00028
00029         Problem();
00030         ~Problem();
00031         inline void setGeometry(const SemGeometry<d, X> *geometry);
00032         inline void setEquation(const Equation<d, X> *equation);
00033         inline void setBoundaryConditions(
00034             const BoundaryConditions<d, X> *boundary_conditions);
00035         inline void setParameters(const SemParameters<X> *parameters);
00036
00037         inline void clearGeometry();
00038         inline void clearEquation();
00039         inline void clearBoundaryConditions();
00040         inline void clearParameters();
00041
00042         inline const SemGeometry<d, X> *geometry() const;
00043         inline const Equation<d, X> *equation() const;
00044         inline const BoundaryConditions<d, X> *boundaryConditions() const;
00045         inline const SemParameters<X> *parameters() const;
00046
00047         inline bool isDefined();
00048     };
00049 };
00050
00052 template<int d, class X>
00053 SemSolver::Problem<d, X>::Problem()
00054 {
00055     _geometry = 0;
00056     _equation = 0;
00057     _boundary_conditions = 0;
00058     _parameters = 0;
00059 };
00060
00062 template<int d, class X>
00063 SemSolver::Problem<d, X>::~~Problem()
00064 {
00065     delete _geometry;
00066     delete _equation;
00067     delete _boundary_conditions;
00068     delete _parameters;
00069 };

```

```

00070
00072 template<int d, class X>
00073 inline void SemSolver::Problem<d, X>::setGeometry(const SemGeometry<d, X> *geomet
ry)
00074 {
00075     _geometry = geometry;
00076 };
00077
00079 template<int d, class X>
00080 inline void SemSolver::Problem<d, X>::setEquation(const Equation<d, X> *equation)
00081 {
00082     _equation = equation;
00083 };
00084
00086 template<int d, class X>
00087 inline void SemSolver::Problem<d, X>::setBoundaryConditions(
00088     const BoundaryConditions<d, X> *boundary_conditions)
00089 {
00090     _boundary_conditions = boundary_conditions;
00091 };
00092
00094 template<int d, class X>
00095 inline void SemSolver::Problem<d, X>::setParameters(const SemParameters<X> *param
eters)
00096 {
00097     _parameters = parameters;
00098 };
00099
00101 template<int d, class X>
00102 inline void SemSolver::Problem<d, X>::clearGeometry()
00103 {
00104     delete _geometry;
00105     _geometry = 0;
00106 };
00107
00109 template<int d, class X>
00110 inline void SemSolver::Problem<d, X>::clearEquation()
00111 {
00112     delete _equation;
00113     _equation = 0;
00114 };
00115
00117 template<int d, class X>
00118 inline void SemSolver::Problem<d, X>::clearBoundaryConditions()
00119 {
00120     delete _boundary_conditions;
00121     _boundary_conditions = 0;
00122 };
00123
00125 template<int d, class X>
00126 inline void SemSolver::Problem<d, X>::clearParameters()
00127 {
00128     delete _parameters;
00129     _parameters = 0;
00130 };
00131
00133 template<int d, class X>
00134 inline const SemSolver::SemGeometry<d, X> *SemSolver::Problem<d, X>::geometry() c
onst
00135 {
00136     return _geometry;

```

```

00137 };
00138
00140 template<int d, class X>
00141 inline const SemSolver::Equation<d, X> *SemSolver::Problem<d, X>::equation() cons
    t
00142 {
00143     return _equation;
00144 };
00145
00147 template<int d, class X>
00148 inline const SemSolver::BoundaryConditions<d, X> *
00149     SemSolver::Problem<d, X>::boundaryConditions() const
00150 {
00151     return _boundary_conditions;
00152 };
00153
00155 template<int d, class X>
00156 inline const SemSolver::SemParameters<X> *SemSolver::Problem<d, X>::parameters()
    const
00157 {
00158     return _parameters;
00159 };
00160
00163 template<int d, class X>
00164 inline bool SemSolver::Problem<d, X>::isDefined()
00165 {
00166     return ( _geometry && _equation && _boundary_conditions && _parameters );
00167 };
00168
00169 #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

```

00001 #ifndef IO_PSLG_HPP
00002 #define IO_PSLG_HPP
00003
00004 #include <QFile>
00005
00006 #include <SemSolver/pslg.hpp>
00007 #include <SemSolver/IO/nextnonemptilinevalues.hpp>
00008
00009 namespace SemSolver
00010 {
00011     namespace IO
00012     {
00013         template<class X>
00014         bool read_PSLG(QFile *file,
00015                       PSLG<X> &pslg)
00016         {
00017             pslg.clear();
00018
00019             #ifdef SEMDEBUG
00020             bool is_open =
00021             #endif
00022                 file->open(QIODevice::ReadOnly);
00023             #ifdef SEMDEBUG
00024             if(!is_open)
00025             {
00026                 qWarning("SemSolver::IO::readPSLG - ERROR : cannot open 'poly_file'");
00027                 return false;
00028             }
00029             #endif
00030             QTextStream input(file);
00031             QStringList values;
00032             values = next_non_empty_line_values(input);
00033             #ifdef SEMDEBUG
00034             if(values.isEmpty())
00035             {
00036                 qWarning("SemSolver::IO::readPSLG - ERROR : missing 'number_of_vertices'");
00037             };
00038             file->close();
00039             return false;
00040         }
00041     #endif
00042     bool ok;
00043     int number_of_vertices = values[0].toInt(&ok);
00044     #ifdef SEMDEBUG
00045     if(!ok)
00046     {
00047         qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_vertices' must be
an inte"
00048                 "ger");
00049         file->close();
00050         return false;
00051     }
00052     if(number_of_vertices < 0)
00053     {
00054         qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_vertices' must be
non-neg"
00055                 "ative");
00056         file->close();
00057         return false;
00058     }

```



```

00059 #endif
00060     pslg.setNumberOfVertices(number_of_vertices);
00061 #ifdef SEMDEBUG
00062     if(values.size()<2)
00063     {
00064         qWarning("SemSolver::IO::readPSLG - ERROR : missing 'dimension'");
00065         file->close();
00066         return false;
00067     }
00068     int dimension = values[1].toInt(&ok);
00069     if(!ok)
00070     {
00071         qWarning("SemSolver::IO::readPSLG - ERROR : 'dimension' must be an integer");
00072         file->close();
00073         return false;
00074     }
00075     if(dimension != 2)
00076     {
00077         qWarning("SemSolver::IO::readPSLG - ERROR : 'dimension' must be 2");
00078         file->close();
00079         return false;
00080     }
00081     if(values.size()<3)
00082     {
00083         qWarning("SemSolver::IO::readPSLG - ERROR : missing 'number_of_attributes'");
00084         file->close();
00085         return false;
00086     }
00087     int number_of_vertices_attributes = values[2].toInt(&ok);
00088     if(!ok)
00089     {
00090         qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_vertices_attributes' must be an integer");
00091         file->close();
00092         return false;
00093     }
00094     if(number_of_vertices_attributes < 0)
00095     {
00096         qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_vertices_attributes' must be non-negative");
00097         file->close();
00098         return false;
00099     }
00100     if(number_of_vertices_attributes > 0)
00101     {
00102         qWarning("SemSolver::IO::readPSLG - ERROR : ignoring vertices attributes in 'pol'");
00103         file->close();
00104         return false;
00105     }
00106     pslg.setNumberOfVerticesAttributes(0);
00107 #ifdef SEMDEBUG
00108     if(values.size()<4)
00109     {
00110         qWarning("SemSolver::IO::readPSLG - ERROR : missing 'number_of_vertices_boundary_markers'");
00111         file->close();
00112         return false;
00113     }
00114 
```

```

00115     int number_of_vertices_boundary_markers = values[3].toInt(&ok);
00116     if(!ok)
00117     {
00118         qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_vertices_boundary_
markers" \
00119             "' must be an integer");
00120         file->close();
00121         return false;
00122     }
00123     if(number_of_vertices_boundary_markers!=0 && number_of_vertices_boundary_mark
ers != 1)
00124     {
00125         qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_vertices_boundary_
markers" \
00126             "' must be 0 or 1");
00127         file->close();
00128         return false;
00129     }
00130     if(number_of_vertices_boundary_markers > 0)
00131         qWarning("SemSolver::IO::readPSLG - ERROR : ignoring vertices boundary ma
rkers i" \
00132             "\n 'poly_file'");
00133 #endif
00134     pslg.setNumberOfVerticesBoundaryMarkers(0);
00135 #ifdef SEMDEBUG
00136     if(values.size()>4)
00137     {
00138         qWarning("SemSolver::IO::readPSLG - ERROR : too many inputs on first line
");
00139         file->close();
00140         return false;
00141     }
00142 #endif
00143     for(int i=0; i<number_of_vertices; ++i)
00144     {
00145         values = next_non_empty_line_values(input);
00146 #ifdef SEMDEBUG
00147         if(values.isEmpty())
00148         {
00149             qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Vertex::vertex_n
umber' ");
00150             file->close();
00151             return false;
00152         }
00153 #endif
00154         int vertex_number = values[0].toInt(&ok);
00155 #ifdef SEMDEBUG
00156         if(!ok)
00157         {
00158             qWarning("SemSolver::IO::readPSLG - ERROR : 'Vertex::vertex_number' m
ust be " \
00159                 "an integer");
00160             file->close();
00161             return false;
00162         }
00163         for(int j=0; j<i; ++j)
00164         {
00165             if(vertex_number==pslg.vertex(j).number)
00166             {
00167                 qWarning("SemSolver::IO::readPSLG - ERROR : 'Vertex::vertex_numbe
r' must" \
00168                     "be unique");

```

```

00169         file->close();
00170         return false;
00171     }
00172 }
00173 if(values.size()<2)
00174 {
00175     qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Vertex::x'");
00176     file->close();
00177     return false;
00178 }
00179 #endif
00180 double x = values[1].toDouble(&ok);
00181 #ifdef SEMDEBUG
00182     if(!ok)
00183     {
00184         qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Vertex::x' must
00185 be a do\"
00186         \"uble\"");
00187         file->close();
00188         return false;
00189     }
00190     if(values.size()<3)
00191     {
00192         qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Vertex::y'");
00193         file->close();
00194         return false;
00195     }
00196     #endif
00197     double y = values[2].toDouble(&ok);
00198     #ifdef SEMDEBUG
00199         if(!ok)
00200         {
00201             qWarning("SemSolver::IO::readPSLG - ERROR : 'Vertex::y' must be a dou
00202 ble\"");
00203             file->close();
00204             return false;
00205         }
00206         if(values.size()<3+number_of_vertices_attributes)
00207         {
00208             qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Vertex::attribut
00209 e'");
00210             file->close();
00211             return false;
00212         }
00213         if(values.size()<3+number_of_vertices_attributes
00214 +number_of_vertices_boundary_markers)
00215         {
00216             qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Vertex::boundary
00217 _marker\"
00218 \"");
00219             file->close();
00220             return false;
00221         }
00222     #endif
00223     pslg.setVertex(i,vertex_number,x,y);
00224 }
00225 values = next_non_empty_line_values(input);
00226 #ifdef SEMDEBUG
00227     if(values.isEmpty())
00228     {
00229         qWarning("SemSolver::IO::readPSLG - ERROR : missing 'number_of_segments'");
00230     }

```

```

00226         file->close();
00227         return false;
00228     }
00229 #endif
00230     int number_of_segments = values[0].toInt(&ok);
00231 #ifdef SEMDEBUG
00232     if(!ok)
00233     {
00234         qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_segments' must be
an inte"
00235                 "ger");
00236         file->close();
00237         return false;
00238     }
00239     if (number_of_segments < 0)
00240     {
00241         qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_segments' must be
non-neg"
00242                 "ative");
00243         file->close();
00244         return false;
00245     }
00246 #endif
00247     pslg.setNumberOfSegments(number_of_segments);
00248 #ifdef SEMDEBUG
00249     if(values.size()<2)
00250     {
00251         qWarning("SemSolver::IO::readPSLG - ERROR : missing 'number_of_segments_b
oundary"
00252                 "_markers'");
00253         file->close();
00254         return false;
00255     }
00256     int number_of_segments_boundary_markers = values[1].toInt(&ok);
00257     if(!ok)
00258     {
00259         qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_segments_boundary_
markers"
00260                 "' must be an integer");
00261         file->close();
00262         return false;
00263     }
00264     if (number_of_segments_boundary_markers!=0 && number_of_segments_boundary_mar
kers!=1)
00265     {
00266         qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_segments_boundary_
markers"
00267                 "' must be 0 or 1");
00268         file->close();
00269         return false;
00270     }
00271     if(number_of_segments_boundary_markers > 0)
00272         qWarning("SemSolver::IO::readPSLG - ERROR : ignoring segments boundary ma
rkers i"
00273                 "n 'poly_file'");
00274     if(values.size()>2)
00275     {
00276         qWarning("SemSolver::IO::readPSLG - ERROR : 'too many inputs on segments
header "
00277                 "line");
00278         file->close();
00279         return false;

```

```

00280     }
00281 #endif
00282     for(int i=0; i<number_of_segments; ++i)
00283     {
00284         values = next_non_empty_line_values(input);
00285 #ifdef SEMDEBUG
00286         if(values.isEmpty())
00287         {
00288             qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Segment::segment
_number"\
00289                     "'");
00290             file->close();
00291             return false;
00292         }
00293 #endif
00294         int segment_number = values[0].toInt(&ok);
00295 #ifdef SEMDEBUG
00296         if(!ok)
00297         {
00298             qWarning("SemSolver::IO::readPSLG - ERROR : 'Segment::segment_number'
must b"\
00299                     "e an integer");
00300             file->close();
00301             return false;
00302         }
00303         for(int j=0; j<i; ++j)
00304         {
00305             if(segment_number==pslg.segment(j).number)
00306             {
00307                 qWarning("SemSolver::IO::readPSLG - ERROR : 'Segment::segment_num
ber' mu"\
00308                         "st be unique");
00309                 file->close();
00310                 return false;
00311             }
00312         }
00313         if(values.size()<2)
00314         {
00315             qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Segment::source'
");
00316             file->close();
00317             return false;
00318         }
00319 #endif
00320         int source = values[1].toInt(&ok);
00321 #ifdef SEMDEBUG
00322         if(!ok)
00323         {
00324             qWarning("SemSolver::IO::readPSLG - ERROR : 'Segment::source' must b
e an in"\
00325                     "teger");
00326             file->close();
00327             return false;
00328         }
00329         if(values.size()<3)
00330         {
00331             qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Segment::target'
");
00332             file->close();
00333             return false;
00334         }
00335 #endif

```

```

00336         int target = values[2].toInt(&ok);
00337 #ifdef SEMDEBUG
00338         if(!ok)
00339         {
00340             qWarning("SemSolver::IO::readPSLG - ERROR : 'Segment::target' must be
an int"\
00341                     "eger");
00342             file->close();
00343             return false;
00344         }
00345         bool found_source = false;
00346         bool found_target = false;
00347         for(int j=0; j<number_of_vertices && !(found_source&&found_target); ++j)
00348         {
00349             if(pslg.vertex(j).number==source)
00350                 found_source = true;
00351             if(pslg.vertex(j).number==target)
00352                 found_target = true;
00353         }
00354         if(!found_source)
00355         {
00356             qWarning("SemSolver::IO::readPSLG - ERROR : 'Segment::source' must be
an exi"\
00357                     "sting 'vertex_number'");
00358             file->close();
00359             return false;
00360         }
00361         if(!found_target)
00362         {
00363             qWarning("SemSolver::IO::readPSLG - ERROR : 'Segment::target' must b
e an ex"\
00364                     "isting 'vertex_number'");
00365             file->close();
00366             return false;
00367         }
00368         if(values.size()<3+number_of_segments_boundary_markers)
00369         {
00370             qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Segment::bounda
ry_mark"\
00371                     "er'");
00372             file->close();
00373             return false;
00374         }
00375         if(values.size()>3+number_of_segments_boundary_markers)
00376         {
00377             qWarning("SemSolver::IO::readPSLG - ERROR : too many inputs on segmen
t line");
00378             file->close();
00379             return false;
00380         }
00381 #endif
00382         pslg.setSegment(i, segment_number, source, target);
00383     }
00384     values = next_non_empty_line_values(input);
00385 #ifdef SEMDEBUG
00386     if(values.isEmpty())
00387     {
00388         qWarning("SemSolver::IO::readPSLG - ERROR : missing 'number_of_holes'");
00389         file->close();
00390         return false;
00391     }
00392 #endif

```

```

00393     int number_of_holes = values[0].toInt(&ok);
00394 #ifdef SEMDEBUG
00395     if(!ok)
00396     {
00397         qWarning("SemSolver::IO::readPSLG - ERROR : 'Segment::number_of_holes' mu
st be a"\
00398             "n integer");
00399         file->close();
00400         return false;
00401     }
00402     if (number_of_holes < 0)
00403     {
00404         qWarning("SemSolver::IO::readPSLG - ERROR : 'number_of_holes' must be non
-negati"\
00405             "ve");
00406         file->close();
00407         return false;
00408     }
00409     if(values.size()>1)
00410     {
00411         qWarning("SemSolver::IO::readPSLG - ERROR : too many inputs on holes hea
der lin"\
00412             "e");
00413         file->close();
00414         return false;
00415     }
00416 #endif
00417     pslg.setNumberOfHoles(number_of_holes);
00418     for(int i=0; i<number_of_holes; ++i)
00419     {
00420         values = next_non_empty_line_values(input);
00421 #ifdef SEMDEBUG
00422         if(values.isEmpty())
00423         {
00424             qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Hole::hole_numbe
r'");
00425             file->close();
00426             return false;
00427         }
00428 #endif
00429         int hole_number = values[0].toInt(&ok);
00430 #ifdef SEMDEBUG
00431         if(!ok)
00432         {
00433             qWarning("SemSolver::IO::readPSLG - ERROR : 'Hole::hole_number' must
be an i"\
00434                 "n integer");
00435             file->close();
00436             return false;
00437         }
00438         for(int j=0; j<i; ++j)
00439         {
00440             if(hole_number==pslg.hole(j).number)
00441             {
00442                 qWarning("SemSolver::IO::readPSLG - ERROR : 'Hole::hole_number' m
ust be "\
00443                     "unique");
00444                 file->close();
00445                 return false;
00446             }
00447         }
00448         if(values.size()<2)

```

```

00449         {
00450             qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Hole::x'");
00451             file->close();
00452             return false;
00453         }
00454     #endif
00455     double x = values[1].toDouble(&ok);
00456     #ifdef SEMDEBUG
00457         if (!ok)
00458         {
00459             qWarning("SemSolver::IO::readPSLG - ERROR : 'Hole::x' must be a doubl
e");
00460             file->close();
00461             return false;
00462         }
00463         if (values.size() < 3)
00464         {
00465             qWarning("SemSolver::IO::readPSLG - ERROR : missing 'Hole::y'");
00466             file->close();
00467             return false;
00468         }
00469     #endif
00470     double y = values[2].toDouble(&ok);
00471     #ifdef SEMDEBUG
00472         if (!ok)
00473         {
00474             qWarning("SemSolver::IO::readPSLG - ERROR : 'Hole::y' must be a doubl
e");
00475             file->close();
00476             return false;
00477         }
00478         if (values.size() > 3)
00479         {
00480             qWarning("SemSolver::IO::readPSLG - ERROR : 'too many inputs on hole
line");
00481             file->close();
00482             return false;
00483         }
00484     #endif
00485     pslg.setHole(i, hole_number, x, y);
00486     }
00487     values = next_non_empty_line_values(input);
00488     #ifdef SEMDEBUG
00489         if (!values.isEmpty())
00490             qWarning("SemSolver::IO::readPSLG - ERROR : ignoring extra information in
'poly_"\
00491                 "file'");
00492     #endif
00493     file->close();
00494     return true;
00495 };
00496 };
00497 };
00498
00499 #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

```

00001 #ifndef PSLG_HPP
00002 #define PSLG_HPP
00003
00005 namespace SemSolver
00006 {
00009     template<class X>
00010     class PSLG;
00011 };
00012
00013 #include <cmath>
00014
00016 namespace SemSolver
00017 {
00019
00022     template<class X>
00024     class PSLG
00025     {
00026     public:
00027
00029
00034         struct Vertex
00035         {
00036             int number;
00037             X x;
00038             X y;
00039             double *attributes;
00040             int marker;
00041         };
00042

```

```

00044
00047     struct Segment
00048     {
00049         int number;
00050         int source;
00051         int target;
00052         int marker;
00053     };
00054
00056
00057     struct Hole
00058     {
00059         int number;
00060         X x;
00061         X y;
00062     };
00063
00064     private:
00065
00066         unsigned vertices_number;
00067         unsigned dimension;
00068         unsigned vertices_attributes_number;
00069         unsigned vertices_boundary_markers_number;
00070         Vertex *vertices_list;
00071
00072         unsigned segments_number;
00073         unsigned segments_boundary_markers_number;
00074         Segment *segments_list;
00075
00076         unsigned holes_number;
00077         Hole *holes_list;
00078
00079     public:
00080
00082         PSLG();
00083
00085         ~PSLG();
00086
00088         void clear();
00089
00093         inline Vertex const &vertex(const unsigned &index) const;
00094
00098         inline Segment const &segment(const unsigned &index) const;
00099
00103         inline Hole const &hole(const unsigned &index) const;
00104
00107         inline unsigned const &vertices() const;
00108
00111         inline unsigned const &segments() const;
00112
00115         inline unsigned const &holes() const;
00116
00119         void setNumberOfVertices(const unsigned &number);
00120
00123         void setNumberOfSegments(const unsigned &number);
00124
00127         void setNumberOfHoles(const unsigned &number);
00128
00131         void setNumberOfVerticesAttributes(const unsigned &number);
00132
00135         inline void setNumberOfVerticesBoundaryMarkers(const unsigned &number);
00136

```

```

00144         void setVertex(unsigned const &index,
00145                         int const &number,
00146                         const X &x,
00147                         const X &y,
00148                         double *attributes = 0,
00149                         int const &marker = 0);
00150
00151         void setSegment(unsigned const &index,
00152                         int const &number,
00153                         int const &source,
00154                         int const &target,
00155                         int const &marker = 0);
00156
00157         void setHole(unsigned const &index,
00158                     int const &number,
00159                     const X &x,
00160                     const X &y);
00161     };
00162 };
00163
00164 template<class X>
00165 SemSolver::PSLG<X>::PSLG()
00166 {
00167     vertices_number = 0;
00168     dimension = 2;
00169     vertices_attributes_number = 0;
00170     vertices_boundary_markers_number = 0;
00171     vertices_list = 0;
00172     segments_number = 0;
00173     segments_boundary_markers_number = 0;
00174     segments_list = 0;
00175     holes_number = 0;
00176     holes_list = 0;
00177 };
00178
00179 template<class X>
00180 SemSolver::PSLG<X>::~~PSLG()
00181 {
00182     for(unsigned i=0; i<vertices_number; ++i)
00183     {
00184         delete [] vertices_list[i].attributes;
00185         vertices_list[i].attributes = 0;
00186     }
00187     //delete [] vertices_list;
00188     //delete [] segments_list;
00189     //delete [] holes_list;
00190 };
00191
00192 template<class X>
00193 void SemSolver::PSLG<X>::clear()
00194 {
00195     for(unsigned i=0; i<vertices_number; ++i)
00196     {
00197         delete [] vertices_list[i].attributes;
00198         vertices_list[i].attributes = 0;
00199     }
00200     delete [] vertices_list;
00201     delete [] segments_list;
00202     delete [] holes_list;
00203     vertices_number = 0;
00204     vertices_attributes_number = 0;
00205     vertices_boundary_markers_number = 0;

```

```

00217     vertices_list = 0;
00218     segments_number = 0;
00219     segments_boundary_markers_number = 0;
00220     segments_list = 0;
00221     holes_number = 0;
00222     holes_list = 0;
00223 };
00224
00225 template<class X>
00226 inline const typename SemSolver::PSLG<X>::Vertex
00227     &SemSolver::PSLG<X>::vertex(const unsigned &index) const
00228 {
00229     #ifdef SEMDEBUG
00230         if(index>=vertices_number)
00231             qFatal("SemSolver::PSLG::vertex - ERROR : index out of bounds.");
00232     #endif //SEMDEBUG
00233     return vertices_list[index];
00234 };
00235
00236 template<class X>
00237 inline const typename SemSolver::PSLG<X>::Segment
00238     &SemSolver::PSLG<X>::segment(const unsigned &index) const
00239 {
00240     #ifdef SEMDEBUG
00241         if(index>=segments_number)
00242             qFatal("SemSolver::PSLG::segment - ERROR : index out of bounds.");
00243     #endif //SEMDEBUG
00244     return segments_list[index];
00245 };
00246
00247 template<class X>
00248 inline const typename SemSolver::PSLG<X>::Hole
00249     &SemSolver::PSLG<X>::hole(const unsigned &index) const
00250 {
00251     #ifdef SEMDEBUG
00252         if(index>=holes_number)
00253             qFatal("SemSolver::PSLG::hole - ERROR : index out of bounds.");
00254     #endif //SEMDEBUG
00255     return holes_list[index];
00256 };
00257
00258 template<class X>
00259 inline const unsigned &SemSolver::PSLG<X>::vertices() const
00260 {
00261     return vertices_number;
00262 };
00263
00264 template<class X>
00265 inline const unsigned &SemSolver::PSLG<X>::segments() const
00266 {
00267     return segments_number;
00268 };
00269
00270 template<class X>
00271 inline const unsigned &SemSolver::PSLG<X>::holes() const
00272 {
00273     return holes_number;
00274 };
00275
00276 template<class X>
00277 void SemSolver::PSLG<X>::setNumberOfVertices(const unsigned &number)
00278 {

```

```

00279     for(unsigned i=0; i<vertices_number; ++i)
00280     {
00281         delete [] vertices_list[i].attributes;
00282         vertices_list[i].attributes = 0;
00283     }
00284     delete [] vertices_list;
00285     vertices_number = number;
00286     if(number)
00287     {
00288         vertices_list = new Vertex[number];
00289         if(vertices_attributes_number)
00290             for(unsigned i=0; i<number; ++i)
00291                 vertices_list[i].attributes = new double[vertices_attributes_numb
er];
00292     }
00293     else
00294         for(unsigned i=0; i<number; ++i)
00295             vertices_list[i].attributes = 0;
00296     }
00297     else
00298         vertices_list = 0;
00299 };
00300 template<class X>
00301 void SemSolver::PSLG<X>::setNumberOfSegments(const unsigned &number)
00302 {
00303     delete [] segments_list;
00304     segments_number = number;
00305     if(number)
00306         segments_list = new Segment[number];
00307     else
00308         segments_list = 0;
00309 };
00310
00311 template<class X>
00312 void SemSolver::PSLG<X>::setNumberOfHoles(const unsigned &number)
00313 {
00314     delete [] holes_list;
00315     holes_number = number;
00316     if(number)
00317         holes_list = new Hole[number];
00318     else
00319         holes_list = 0;
00320 };
00321
00322 template<class X>
00323 void SemSolver::PSLG<X>::setNumberOfVerticesAttributes(const unsigned &number)
00324 {
00325     vertices_attributes_number = number;
00326     if(number)
00327     {
00328         for(unsigned i=0; i<vertices_number; ++i)
00329         {
00330             delete [] vertices_list[i].attributes;
00331             vertices_list[i].attributes = new double[number];
00332         }
00333     }
00334     else
00335     {
00336         for(unsigned i=0; i<vertices_number; ++i)
00337         {
00338             delete [] vertices_list[i].attributes;
00339             vertices_list[i].attributes = 0;

```

```

00340     }
00341 }
00342 };
00343
00344 template<class X>
00345 void SemSolver::PSLG<X>::setNumberOfVerticesBoundaryMarkers(const unsigned &numbe
r)
00346 {
00347 #ifdef SEMDEBUG
00348     if(number!=0 && number !=1)
00349         qFatal("SemSolver::PSLG::setNumberOfVerticesBoundaryMarkers - ERROR : num
ber mus"
00350             "t be 0 or 1.");
00351 #endif //SEMDEBUG
00352     vertices_boundary_markers_number = number;
00353 };
00354
00355 template<class X>
00356 void SemSolver::PSLG<X>::setVertex(const unsigned &index,
00357                                     int const &number,
00358                                     const X &x,
00359                                     const X &y,
00360                                     double *attributes,
00361                                     int const &marker)
00362 {
00363 #ifdef SEMDEBUG
00364     if(index>=vertices_number)
00365         qFatal("SemSolver::PSLG::setVertex - ERROR : index out of bounds.");
00366 #endif //SEMDEBUG
00367     vertices_list[index].number = number;
00368     vertices_list[index].x = x;
00369     vertices_list[index].y = y;
00370     delete [] vertices_list[index].attributes;
00371     if(vertices_attributes_number)
00372     {
00373         vertices_list[index].attributes = new double[vertices_attributes_number];
00374
00375         for (unsigned i=0; i<vertices_attributes_number; ++i)
00376             vertices_list[index].attributes[i] = attributes[i];
00377     }
00378     else
00379         vertices_list[index].attributes = 0;
00380     vertices_list[index].marker = marker;
00381 };
00382
00383 template<class X>
00384 void SemSolver::PSLG<X>::setSegment(const unsigned &index,
00385                                     int const &number,
00386                                     int const &source,
00387                                     int const &target,
00388                                     int const &marker)
00389 {
00390 #ifdef SEMDEBUG
00391     if(index>=segments_number)
00392         qFatal("SemSolver::PSLG::setSegment- ERROR : index out of bounds.");
00393 #endif //SEMDEBUG
00394     segments_list[index].number = number;
00395     segments_list[index].source = source;
00396     segments_list[index].target = target;
00397     segments_list[index].marker = marker;
00398 };

```

```

00399 template<class X>
00400 void SemSolver::PSLG<X>::setHole(const unsigned &index,
00401                                int const &number,
00402                                const X &x,
00403                                const X &y)
00404 {
00405     #ifdef SEMDEBUG
00406         if(index>=holes_number)
00407             qFatal("SemSolver::PSLG::setHole - ERROR : index out of bounds.");
00408     #endif //SEMDEBUG
00409     holes_list[index].number = number;
00410     holes_list[index].x = x;
00411     holes_list[index].y = y;
00412 };
00413
00414 #endif // PSLG_HPP
00415

```

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

```

00001 #ifndef QRSOLVE_HPP
00002 #define QRSOLVE_HPP
00003
00004 #if defined _WIN32 || defined _WIN64
00005     #include <SemSolver/math_defines>
00006 #endif
00007
00008 #include <jama_qr.h>

```

```

00009
00010 #include <SemSolver/matrix.hpp>
00011 #include <SemSolver/vector.hpp>
00012
00013 namespace SemSolver
00014 {
00015
00016     namespace Solver
00017     {
00018         {
00019             template<class X>
00020             bool qr_solve(Matrix<X> const &A,
00021                          Vector<X> const &b,
00022                          Vector<X> &x)
00023             {
00024                 JAMA::QR<double> qr(A);
00025 #ifdef SEMDEBUG
00026                 if(!qr.isFullRank())
00027                 {
00028                     qWarning("SemSolver::Solver::qr_solve - ERROR : Matrix A is not f
00029 ull ran"\
00030                               "k.");
00031                     return false;
00032                 }
00033 #endif
00034                 x = qr.solve(b);
00035                 return true;
00036             };
00037         };
00038     };
00039 };
00040 };
00041 };
00042
00043 #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 in scripts.

- 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

```

00001 #ifndef SCRIPTFUNCTION_HPP
00002 #define SCRIPTFUNCTION_HPP
00003
00004 namespace SemSolver
00005 {
00006     template<class X, class Y>
00007     class ScriptFunction;
00008 };
00009
00010 #include <cmath>
00011
00012 #include <SemSolver/point.hpp>
00013 #include <SemSolver/function.hpp>
00014 #include <SemSolver/vector.hpp>
00015
00016 #include <QString>
00017 #include <QStringList>
00018 #include <QScriptEngine>
00019
00020 namespace SemSolver
00021 {
00022     template<int d, class X, class Y>
00027     class ScriptFunction< Point<d, X>, Y >
00028     : public Function< Point<d, X>, Y >
00029     {
00030     public:
00031         QString script;
00032         QString program;
00033         QScriptEngine *engine;
00034     public:
00035         ScriptFunction(QString const &string);
00041         ~ScriptFunction();
00043         Y evaluate(const Point<d, X> &point) const;
00047         QString mml() const;
00050

```

```

00051     };
00052
00057     template<int d, class X, class Y>
00058     class ScriptFunction< Point<d, X>, Vector<Y> >
00059         : public Function< Point<d, X>, Vector<Y> >
00060     {
00061         QStringList scripts;
00062         QString program;
00063         QScriptEngine *engine;
00064     public:
00065
00073         ScriptFunction(QStringList const &strings);
00074
00076         ~ScriptFunction();
00077
00079         Vector<Y> evaluate(const Point<d, X> &point) const;
00080
00083         QString mml() const;
00084     };
00085
00090     template<class X, class Y>
00091     class ScriptFunction< Point<2, X>, Y >
00092         : public Function< Point<2, X>, Y >
00093     {
00094         QString script;
00095         QString program;
00096         QScriptEngine *engine;
00097     public:
00098
00104         ScriptFunction(QString const &string);
00105
00107         ~ScriptFunction();
00108
00110         Y evaluate(const Point<2, X> &point) const;
00111
00114         QString mml() const;
00115     };
00116
00121     template<class X, class Y>
00122     class ScriptFunction< Point<2, X>, Vector<Y> >
00123         : public Function< Point<2, X>, Vector<Y> >
00124     {
00125         QStringList scripts;
00126         QString program;
00127         QScriptEngine *engine;
00128     public:
00129
00136         ScriptFunction(QStringList const &strings);
00137
00139         ~ScriptFunction();
00140
00142         Vector<Y> evaluate(const Point<2, X> &point) const;
00143
00146         QString mml() const;
00147     };
00148 };
00149
00150 template<int d, class X, class Y>
00151 SemSolver::ScriptFunction< SemSolver::Point<d, X>, Y >::ScriptFunction(
00152     const QString &string)
00153 {
00154     script = string;

```

```

00155     engine = new QScriptEngine;
00156     program = "function f(";
00157     for(int i=0; i<d-1; ++i)
00158         program += "x" + QString::number(i) + ", ";
00159     program += "x" + QString::number(d) + ") { return " + script + "; }";
00160     engine->evaluate("function abs    (x)    { return Math.abs(x);    }\"\\
00161                    \"function acos   (x)    { return Math.acos(x);   }\"\\
00162                    \"function asin   (x)    { return Math.asin(x);  }\"\\
00163                    \"function atan   (x)    { return Math.atan(x);   }\"\\
00164                    \"function atan2  (x,y) { return Math.atan2(x,y); }\"\\
00165                    \"function ceil   (x)    { return Math.ceil(x);   }\"\\
00166                    \"function cos    (x)    { return Math.cos(x);    }\"\\
00167                    \"function exp    (x)    { return Math.exp(x);    }\"\\
00168                    \"function floor  (x)    { return Math.floor(x);   }\"\\
00169                    \"function log    (x)    { return Math.log(x);    }\"\\
00170                    \"function max    (x,y) { return Math.max(x,y);   }\"\\
00171                    \"function min    (x,y) { return Math.min(x,y);   }\"\\
00172                    \"function pow    (x,y) { return Math.pow(x,y);   }\"\\
00173                    \"function random() { return Math.random();   }\"\\
00174                    \"function round  (x)    { return Math.round(x);   }\"\\
00175                    \"function sin    (x)    { return Math.sin(x);    }\"\\
00176                    \"function sqrt   (x)    { return Math.sqrt(x);   }\"\\
00177                    \"function tan    (x)    { return Math.tan(x);    }\"\\
00178                    \"var pi = Math.PI;\"\\
00179                    \"var e  = Math.E ;\"\\
00180     engine->evaluate(program);
00181 };
00182
00183 template<int d, class X, class Y>
00184 SemSolver::ScriptFunction< SemSolver::Point<d, X>, Y >::~ScriptFunction()
00185 {
00186     delete engine;
00187 };
00188
00189 template<int d, class X, class Y>
00190 Y SemSolver::ScriptFunction< SemSolver::Point<d, X>, Y >::evaluate(
00191     const SemSolver::Point<d, X> &point) const
00192 {
00193     QString call = "f(";
00194     for(int i=0; i<d-1; ++i)
00195         call += QString::number(point.cartesian(i)) + ",";
00196     call += QString::number(point.cartesian(d-1)) + ")";
00197     QScriptValue value = engine->evaluate(call);
00198     return value.toNumber();
00199 };
00200
00201 template<int d, class X, class Y>
00202 QString SemSolver::ScriptFunction< SemSolver::Point<d, X>, Y >::mml() const
00203 {
00204     return "<mtext> " + script + "</mtext>";
00205 };
00206
00207 template<int d, class X, class Y>
00208 SemSolver::ScriptFunction< SemSolver::Point<d, X>, SemSolver::Vector<Y> >::Script
Function(
00209     const QStringList &strings)
00210 {
00211     scripts = strings;
00212     engine = new QScriptEngine;
00213     for(int j=0; j<scripts.size(); ++j)
00214     {
00215         program += "function f" + QString::number(j) + "(";

```

```

00216         for(int i=0; i<d-1; ++i)
00217             program += "x" + QString::number(i) + ", ";
00218         program += "x" + QString::number(d) + ") { return " + scripts[j] + "; }\n";
00219     };
00220 }
00221 engine->evaluate("function abs (x) { return Math.abs(x); }\n\
00222     \"function acos (x) { return Math.acos(x); }\n\
00223     \"function asin (x) { return Math.asin(x); }\n\
00224     \"function atan (x) { return Math.atan(x); }\n\
00225     \"function atan2 (x,y) { return Math.atan2(x,y); }\n\
00226     \"function ceil (x) { return Math.ceil(x); }\n\
00227     \"function cos (x) { return Math.cos(x); }\n\
00228     \"function exp (x) { return Math.exp(x); }\n\
00229     \"function floor (x) { return Math.floor(x); }\n\
00230     \"function log (x) { return Math.log(x); }\n\
00231     \"function max (x,y) { return Math.max(x,y); }\n\
00232     \"function min (x,y) { return Math.min(x,y); }\n\
00233     \"function pow (x,y) { return Math.pow(x,y); }\n\
00234     \"function random() { return Math.random(); }\n\
00235     \"function round (x) { return Math.round(x); }\n\
00236     \"function sin (x) { return Math.sin(x); }\n\
00237     \"function sqrt (x) { return Math.sqrt(x); }\n\
00238     \"function tan (x) { return Math.tan(x); }\n\
00239     \"var pi = Math.PI;\n\
00240     \"var e = Math.E ;\n\");
00241 engine->evaluate(program);
00242 };
00243
00244 template<int d, class X, class Y>
00245 SemSolver::ScriptFunction<SemSolver::Point<d,X>, SemSolver::Vector<Y> >::~ScriptF
00246 unction()
00247 {
00248     delete engine;
00249 };
00250 template<int d, class X, class Y>
00251 SemSolver::Vector<Y> SemSolver::ScriptFunction<
00252     SemSolver::Point<d, X>, SemSolver::Vector<Y> >::evaluate(
00253     const SemSolver::Point<d, X> &point) const
00254 {
00255     int l = scripts.size();
00256     Vector<Y> result(l);
00257     for(int j=0; j<l; ++j)
00258     {
00259         QString call = "f" + QString::number(j) + "(";
00260         for(int i=0; i<d-1; ++i)
00261             call += QString::number(point.cartesian(i)) + ", ";
00262         call += QString::number(point.cartesian(d-1)) + ")";
00263         QScriptValue value = engine->evaluate(call);
00264         result[j] = value.toNumber();
00265     };
00266     return result;
00267 };
00268
00269 template<int d, class X, class Y>
00270 QString SemSolver::ScriptFunction<SemSolver::Point<d, X>, SemSolver::Vector<Y> >:
00271 :mml(
00272     ) const
00273 {
00274     QString mml = "<mfenced open='(' close=')'><mtable>";
00275     for(int i=0; i<scripts.size(); ++i)

```

```

00275         mml += "<mtr><mt><mtext>" + scripts[i] + "</mtext></mt></mtr>";
00276         mml += "</mtable></mfenced>";
00277         return mml;
00278     };
00279
00280     template<class X, class Y>
00281     SemSolver::ScriptFunction< SemSolver::Point<2, X>, Y >::ScriptFunction(
00282         const QString &string)
00283     {
00284         script = string;
00285         engine = new QScriptEngine;
00286         program = "function f(x,y) { return " + script + "; }";
00287         engine->evaluate("function abs (x) { return Math.abs(x); }\"\\
00288             "function acos (x) { return Math.acos(x); }\"\\
00289             "function asin (x) { return Math.asin(x); }\"\\
00290             "function atan (x) { return Math.atan(x); }\"\\
00291             "function atan2 (x,y) { return Math.atan2(x,y); }\"\\
00292             "function ceil (x) { return Math.ceil(x); }\"\\
00293             "function cos (x) { return Math.cos(x); }\"\\
00294             "function exp (x) { return Math.exp(x); }\"\\
00295             "function floor (x) { return Math.floor(x); }\"\\
00296             "function log (x) { return Math.log(x); }\"\\
00297             "function max (x,y) { return Math.max(x,y); }\"\\
00298             "function min (x,y) { return Math.min(x,y); }\"\\
00299             "function pow (x,y) { return Math.pow(x,y); }\"\\
00300             "function random() { return Math.random(); }\"\\
00301             "function round (x) { return Math.round(x); }\"\\
00302             "function sin (x) { return Math.sin(x); }\"\\
00303             "function sqrt (x) { return Math.sqrt(x); }\"\\
00304             "function tan (x) { return Math.tan(x); }\"\\
00305             "var pi = Math.PI;\"\\
00306             "var e = Math.E ;\"");
00307         engine->evaluate(program);
00308     };
00309
00310     template<class X, class Y>
00311     SemSolver::ScriptFunction< SemSolver::Point<2, X>, Y >::~ScriptFunction()
00312     {
00313         delete engine;
00314     };
00315
00316     template<class X, class Y>
00317     Y SemSolver::ScriptFunction< SemSolver::Point<2, X>, Y >::evaluate(
00318         const SemSolver::Point<2, X> &point) const
00319     {
00320         QString call = "f(" + QString::number(point.x()) + "," + QString::number(poin
00321             t.y()) + ")";
00322         QScriptValue value = engine->evaluate(call);
00323         return value.toNumber();
00324     };
00325
00326     template<class X, class Y>
00327     QString SemSolver::ScriptFunction< SemSolver::Point<2, X>, Y >::mml() const
00328     {
00329         return "<mtext> " + script + "</mtext>";
00330     };
00331
00332     template<class X, class Y>
00333     SemSolver::ScriptFunction< SemSolver::Point<2, X>, SemSolver::Vector<Y> >::Script
00334         Function(
00335         const QStringList &strings)

```

```

00335 {
00336     scripts = strings;
00337     engine = new QScriptEngine;
00338     for(int j=0; j<scripts.size(); ++j)
00339         program += "function f" + QString::number(j) + "(x,y) { return " + scripts
00340 [j] + \
00341             "; }\n";
00342     engine->evaluate("function abs    (x)    { return Math.abs(x);    }"\
00343 "function acos  (x)    { return Math.acos(x);  }"\
00344 "function asin  (x)    { return Math.asin(x);  }"\
00345 "function atan   (x)    { return Math.atan(x);   }"\
00346 "function atan2  (x,y)  { return Math.atan2(x,y); }"\
00347 "function ceil   (x)    { return Math.ceil(x);   }"\
00348 "function cos    (x)    { return Math.cos(x);    }"\
00349 "function exp    (x)    { return Math.exp(x);    }"\
00350 "function floor  (x)    { return Math.floor(x);   }"\
00351 "function log    (x)    { return Math.log(x);    }"\
00352 "function max    (x,y)  { return Math.max(x,y);  }"\
00353 "function min    (x,y)  { return Math.min(x,y);  }"\
00354 "function pow    (x,y)  { return Math.pow(x,y);  }"\
00355 "function random()      { return Math.random();  }"\
00356 "function round  (x)    { return Math.round(x);   }"\
00357 "function sin    (x)    { return Math.sin(x);    }"\
00358 "function sqrt   (x)    { return Math.sqrt(x);   }"\
00359 "function tan    (x)    { return Math.tan(x);    }"\
00360 "var pi = Math.PI;      "\
00361 "var e  = Math.E ;      ");
00362     engine->evaluate(program);
00363 };
00364
00365 template<class X, class Y>
00366 SemSolver::ScriptFunction<SemSolver::Point<2,X>, SemSolver::Vector<Y> >::~ScriptF
00367 unction()
00368 {
00369     delete engine;
00370 };
00371
00372 template<class X, class Y>
00373 SemSolver::Vector<Y> SemSolver::ScriptFunction<
00374     SemSolver::Point<2, X>, SemSolver::Vector<Y> >::evaluate(
00375     const SemSolver::Point<2, X> &point) const
00376 {
00377     int l = scripts.size();
00378     Vector<Y> result(l);
00379     for(int j=0; j<l; ++j)
00380     {
00381         QString call = "f" + QString::number(j) + "(" + QString::number(point.x())
00382             + QString::number(point.y()) + ")";
00383         QScriptValue value = engine->evaluate(call);
00384         result[j] = value.toNumber();
00385     };
00386     return result;
00387 };
00388
00389 template<class X, class Y>
00390 QString SemSolver::ScriptFunction<SemSolver::Point<2, X>, SemSolver::Vector<Y> >:
00391 :mml(
00392     ) const
00393 {
00394     QString mml = "<mfenced open='(' close=')'><mtable>";

```

```

00393     for(int i=0; i<scripts.size(); ++i)
00394         mml += "<mtr><mtd><mtext>" + scripts[i] + "</mtext></mtd></mtr>";
00395     mml += "</mtable></mfenced>";
00396     return mml;
00397 };
00398
00399 #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

```

00001 #ifndef SEGMENT_HPP
00002 #define SEGMENT_HPP
00003
00004 namespace SemSolver
00005 {
00006     template<int d, class X>
00007     class Segment;
00008 };
00009
00010 #include <CGAL/Cartesian.h>
00011 #include <CGAL/Object.h>
00012 #include <CGAL/Segment_2.h>
00013 #include <CGAL/intersections.h>
00014
00015 #include <SemSolver/point.hpp>
00016
00018 namespace SemSolver
00019 {
00020     template<class X>

```

```

00021     class Segment<2,X>
00022     : public CGAL::Segment_2< CGAL::Filtered_kernel< CGAL::Simple_cartesian<X
> > >
00023     {
00024     typedef CGAL::Point_2< CGAL::Filtered_kernel< CGAL::Simple_cartesian<X> >
>
00025         CGAL_point;
00026     typedef CGAL::Segment_2< CGAL::Filtered_kernel< CGAL::Simple_cartesian<X>
> >
00027         CGAL_segment;
00028
00029     public:
00030
00031         Segment(CGAL_segment const &s) : CGAL_segment(s) {};
00032
00033         Segment(Point<2,X> const &source, Point<2,X> const &target)
00034             : CGAL_segment(source, target) {};
00035
00036         CGAL_segment const &cgal() const
00037         {
00038             return *this;
00039         };
00040
00041         bool intersect(Segment<2,X> const &segment) const
00042         {
00043             return CGAL::do_intersect(segment, *this);
00044         };
00045
00046         bool intersectInteriorly(Segment<2,X> const &segment) const
00047         {
00048             if(!intersect(segment))
00049                 return false;
00050             CGAL_point intersection;
00051             CGAL::Object result = CGAL::intersection(segment.cgal(), this->cgal()
);
00052             if(!CGAL::assign(intersection, result))
00053                 return true;
00054             if(intersection!=segment.source() && intersection!=segment.target() &
&
00055                 intersection!=this->source() && intersection!=this->target())
00056                 return true;
00057             return false;
00058         };
00059
00060         struct less
00061         {
00062             bool operator() (Segment const &s0, Segment const &s1) const
00063             {
00064                 if(s0.source().x()<s1.source().x())
00065                     return true;
00066                 if(s0.source().x()>s1.source().x())
00067                     return false;
00068                 if(s0.source().y()<s1.source().y())
00069                     return true;
00070                 if(s0.source().y()>s1.source().y())
00071                     return false;
00072                 if(s0.target().x()<s1.target().x())
00073                     return true;
00074                 if(s0.target().x()>s1.target().x())
00075                     return false;
00076                 if(s0.target().y()<s1.target().y())
00077                     return true;
00078                 if(s0.target().y()>s1.target().y())
00079                     return false;
00080                 return true;

```



```

00081             //if(s0.target().y()>s1.target().y())
00082             return false;
00083         }
00084     };
00085 };
00086 };
00087
00088 #endif // SEGMENT_HPP

```

7.89 segmentmap.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 segmentmap.hpp

```

00001 #ifndef SEGMENTSVECTOR_HPP
00002 #define SEGMENTSVECTOR_HPP
00003
00004 namespace SemSolver
00005 {
00006     template<int d, class X>
00007     class SegmentsMap;
00008 };
00009
00010 #include <map>
00011
00012 #include <SemSolver/segment.hpp>
00013 #include <SemSolver/point.hpp>
00014
00017
00023 namespace SemSolver
00024 {
00025     template<int d, class X>
00026     class SegmentsMap
00027     : std::map< int, Segment<d,X> >
00028     {
00029     private:
00030         typedef std::map< int, Segment<d,X> > Base;

```

```

00031
00032     protected:
00033         typedef typename Base::iterator Iterator;
00034
00035     public:
00036         typedef typename Base::const_iterator ConstIterator;
00037         using Base::begin;
00038         using Base::end;
00039
00040     private:
00041         int _current_id;
00042
00043     public:
00044
00046         SegmentsMap()
00047         {
00048             _current_id = 0;
00049         }
00050
00052         bool contains(Segment<d,X> const &segment) const
00053         {
00054             for(ConstIterator it=Base::begin(); it!=Base::end(); ++it)
00055                 if(it->second==segment)
00056                     return true;
00057             return false;
00058         }
00059
00061         bool has(int const &segment_id) const
00062         {
00063             for(ConstIterator it=begin(); it!=end(); ++it)
00064                 if(it->first==segment_id)
00065                     return true;
00066             return false;
00067         }
00068
00070         Segment<d,X> segmentFrom(Point<d,X> const &source) const
00071         {
00072             for(ConstIterator it=begin(); it!=end(); ++it)
00073                 if(it->second.source()==source)
00074                     return it->second;
00075             return Segment<d,X>(source, source);
00076         }
00077
00079         Segment<d,X> segmentTo(Point<d,X> const &target) const
00080         {
00081             for(ConstIterator it=begin(); it!=end(); ++it)
00082                 if(it->second.target()==target)
00083                     return it->second;
00084             return Segment<d,X>(target, target);
00085         }
00086
00088         Segment<d,X> const &segment(int const &id) const
00089         {
00090             ConstIterator it = Base::find(id);
00091 #ifdef SEMDEBUG
00092             if(it==Base::end())
00093                 qFatal("SemSolver::SegmentsMap::modify - ERROR : id out of range"
00094 );
00095 #endif
00096             return it->second;
00097         };

```

```

00100     int id(Segment<d,X> const &segment) const
00101     {
00102         for(ConstIterator it=begin(); it!=end(); ++it)
00103             if(it->second==segment)
00104                 return it->first;
00105         qFatal("SemSolver::SegmentsMap::id - ERROR : segment must exist");
00106         return 0;
00107     };
00108
00111     int add(Segment<d,X> const &segment)
00112     {
00113         int id = _new_id();
00114         insert(id,segment);
00115         return id;
00116     }
00117
00119     void insert(int const &id, Segment<d,X> const &segment)
00120     {
00121         Base::insert(typename Base::value_type(id,segment));
00122     }
00123
00125     void remove(int const &id)
00126     {
00127         Base::erase(id);
00128     }
00129
00133     inline void modify(int const &id, Segment<d,X> const &segment)
00134     {
00135         Iterator it = Base::find(id);
00136 #ifdef SEMDEBUG
00137         if(it==Base::end())
00138             qFatal("SemSolver::SegmentsMap::modify - ERROR : id out of range"
00139 );
00139 #endif
00140         it->second = segment;
00141     }
00142
00144     int segments() const
00145     {
00146         return Base::size();
00147     };
00148
00150     virtual bool intersect(Segment<d,X> const &segment) const
00151     {
00152         for(ConstIterator it=begin(); it!=end(); ++it)
00153             if(segment.intersect(it->second))
00154                 return true;
00155         return false;
00156     }
00157
00160     virtual bool intersectInteriorly(Segment<d,X> const &segment) const
00161     {
00162         for(ConstIterator it=begin(); it!=end(); ++it)
00163             if(segment.intersectInteriorly(it->second))
00164                 return true;
00165         return false;
00166     }
00167
00170     bool isConsistentWith(Segment<d,X> const &segment) const
00171     {
00172         for (ConstIterator it=begin(); it!=end(); ++it)
00173             if(it->second.source()==segment.source() ||

```

```

00174             it->second.target()==segment.target() )
00175                 return false;
00176             return true;
00177         }
00178
00179     bool haveOn(Point<d,X> const &point) const
00180     {
00181         for (ConstIterator it=begin(); it!=end(); ++it)
00182             if(it->second.has_on(point))
00183                 return true;
00184             return false;
00185     }
00186
00187
00188
00189     private:
00190         int const &_new_id()
00191         {
00192             return ++_current_id;
00193         };
00194
00195     };
00196 };
00197
00198 #endif // SEGMENTSVECTOR_HPP

```

7.91 semfunction.hpp File Reference

```

#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>

```

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

```

00001 #ifndef SEMFUNCTION_HPP
00002 #define SEMFUNCTION_HPP
00003
00004 #include <SemSolver/function.hpp>
00005 #include <SemSolver/polynomialfunction.hpp>
00006 #include <SemSolver/bilineartransformation.hpp>
00007 #include <SemSolver/point.hpp>
00008 #include <SemSolver/semgeometry.hpp>
00009 #include <SemSolver/matrix.hpp>
00010 #include <SemSolver/vector.hpp>
00011 #include <SemSolver/semparameters.hpp>
00012
00013 namespace SemSolver
00014 {
00015
00016     template <int d,class X>
00017     class SemFunction
00018     : public Function< Point<d,X>, X >
00019     {};
00020
00021
00022
00023
00024
00025     template<class X>
00026     class SemFunction<2,X>
00027     : public Function< Point<2,X>, X >
00028     {
00029     {
00030         typedef std::vector< PolynomialFunction<2,X> > PolynomialFunctionVector;
00031         typedef std::vector< BilinearTransformation<X> > BilinearTransformationsV
ector;
00032
00033         Polygonation<2,X> _polygonation;
00034         PolynomialFunctionVector _polynomials;
00035         BilinearTransformationsVector _maps;
00036
00037     public:
00038         //**** tranformation to be generalized *****/
00039         SemFunction(
00040             SemGeometry<2,X> const &geometry,
00041             PolynomialFunctionVector const &polynomials,
00042             BilinearTransformationsVector const &maps)
00043             : _polygonation(geometry.subDomains()),
00044               _polynomials(polynomials),
00045               _maps(maps)
00046         {};
00047
00048
00049         SemFunction(SemFunction const &sem_function)
00050             : _polygonation(sem_function._polygonation),
00051               _polynomials(sem_function._polynomials),
00052               _maps(sem_function._maps)
00053         {};
00054
00055         inline PolynomialFunction<2,X> const &polynomial(const unsigned &index) c
onst
00056         {
00057
00058
00059         }
00060 #if SEMDEBUG

```

```

00061         if(index>=_polygonation.size())
00062             qFatal("SemSolver::SemFunction::polynomial error : index out of r
ange");
00063     #endif
00064         return _polynomials.at(index);
00065     };
00066
00068     inline void setPolynomialComponent(const unsigned &index,
00069                                       const unsigned &component,
00070                                       const Polynomial<X> &poly)
00071     {
00072     #if SEMDEBUG
00073         if(index>=_polygonation.size())
00074             qFatal("SemSolver::SemFunction::polynomial error : index out of r
ange");
00075     #endif
00076         _polynomials.at(index).setPolynomial(component, poly);
00077     };
00078
00080     inline BilinearTransformation<X> const &map(const unsigned &index) const
00081     {
00082     #if SEMDEBUG
00083         if(index>=_polygonation.size())
00084             qFatal("SemSolver::SemFunction::polynomial error : index out of r
ange");
00085     #endif
00086         return _maps.at(index);
00087     };
00088
00090     double evaluate(Point<2,X> const &x) const
00091     {
00092         std::vector<unsigned> element_index = _polygonation.elementIndicesAt(
x);
00093         if(element_index.size()==0)
00094             return 0.;
00095         Point<2,X> x_hat = map(element_index[0]).evaluateInverse(x);
00096         return polynomial(element_index[0]).evaluate(x_hat);
00097     };
00098
00100     Vector<X> evaluateRestrictionGradient(
00101         int const &element_index,
00102         Point<2,X> const &P) const
00103     {
00104         Vector<double> gradient(2);
00105         Point<2,X> const &P_hat = map(element_index).evaluateInverse(P);
00106         Matrix<X> tIJ_phi = map(element_index).evaluateTransposeInverseJacobi
an(P_hat);
00107         X const &x_hat = P_hat.x();
00108         X const &y_hat = P_hat.y();
00109         Polynomial<X> const &px = polynomial(element_index).polynomial(0);
00110         Polynomial<X> const &py = polynomial(element_index).polynomial(1);
00111         X psi_x = px.derivative()(x_hat) * py(y_hat);
00112         X psi_y = px(x_hat) * py.derivative()(y_hat);
00113         gradient[0] = psi_x * tIJ_phi[0][0] + psi_y * tIJ_phi[0][1];
00114         gradient[1] = psi_x * tIJ_phi[1][0] + psi_y * tIJ_phi[1][1];
00115         return gradient;
00116     };
00117     };
00118 };
00119
00120 #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

```
00001 #ifndef SEMGEOMETRY_HPP
00002 #define SEMGEOMETRY_HPP
00003
00005 namespace SemSolver
00006 {
00008
00010
00011
00017     template<int d, class X>
00018     class SemGeometry
00019     {
00020     };
00021 };
00022
00023 #include <SemSolver/pslg.hpp>
00024 #include <SemSolver/polygonation.hpp>
00025
00026 namespace SemSolver
00027 {
00028
00029     template<class X>
00030     class SemGeometry<2,X>
00031     {
00032
00033         PSLG<X> pslg;
00034         Polygonation<2,X> sub_domains;
00035
00036     public:
00037
00039         inline PSLG<X> const &domain() const
00040         {
00041             return pslg;
```

```

00042 };
00043
00045     inline Polygonation<2,X> const &subDomains() const
00046 {
00047     return sub_domains;
00048 };
00049
00051     inline void setDomain(PSLG<X> const &domain)
00052     {
00053         pslg = domain;
00054 };
00055
00057     inline void setSubDomains(Polygonation<2,X> const &sub_domains)
00058     {
00059         this->sub_domains = sub_domains;
00060 };
00061
00063     inline bool contains(const Point<2, X> &point) const
00064 {
00065     return !sub_domains.elementIndicesAt(point).empty();
00066 };
00067 };
00068 };
00069
00070 #endif // SEMGEOMETRY_HPP

```

7.95 semparameters.hpp File Reference

Classes

- class [SemSolver::SemParameters< X >](#)

Namespaces

- namespace [SemSolver](#)
Project main namespace.

7.96 semparameters.hpp

```

00001 #ifndef SEMPARAMETERS_HPP
00002 #define SEMPARAMETERS_HPP
00003
00004 namespace SemSolver
00005 {
00006     template<class X>
00007     class SemParameters;
00008 };
00009
00010 namespace SemSolver
00011 {
00014     template <class X>
00015     class SemParameters

```



```

00016     {
00017         int _degree;
00018         X _tolerance;
00019         X _penality;
00020
00021     public:
00022         SemParameters()
00023         {};
00024
00025         SemParameters(int const &degree,
00026                       X const &tolerance,
00027                       X const &penality)
00028             : _degree(degree),
00029               _tolerance(tolerance),
00030               _penality(penality)
00031         {};
00032
00033         inline int const &degree() const
00034         {
00035             return _degree;
00036         };
00037
00038         inline X const &tolerance() const
00039         {
00040             return _tolerance;
00041         };
00042
00043         inline X const &penality() const
00044         {
00045             return _penality;
00046         };
00047
00048         inline void setDegree(const int &d)
00049         {
00050             _degree = d;
00051         };
00052
00053         inline void setTolerance(const X &t)
00054         {
00055             _tolerance = t;
00056         };
00057
00058         inline void setPenality(const X &p)
00059         {
00060             _penality = p;
00061         };
00062     };
00063 };
00064
00065 #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/pointsmmap.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

```
00001 #ifndef SEMSPACE_HPP
00002 #define SEMSPACE_HPP
00003
00004 namespace SemSolver
00005 {
00006     template <int d, class X>
00007         class SemSpace;
00008 }
00009
00010 #include <vector>
00011 #include <cmath>
00012
00013 #include <SemSolver/hilbertspace.hpp>
00014 #include <SemSolver/semfunction.hpp>
00015 #include <SemSolver/multiindex.hpp>
00016 #include <SemSolver/semgeometry.hpp>
00017 #include <SemSolver/semparameters.hpp>
00018 #include <SemSolver/pointsmmap.hpp>
00019
00020 namespace SemSolver
00021 {
00022
00023
00024
00025
00031     template <int d, class X>
00032         class SemSpace
00033             : public HilbertSpace< SemFunction<d,X>, X >
00034         {};
00035
00044     template<class X>
```

```

00045     class SemSpace<2,X>
00046     : public HilbertSpace< SemFunction<2,X>, X >
00047     {
00048     public:
00051         class Node
00052         {
00053             typedef std::vector< MultiIndex<2> > Indices2Vector;
00054             typedef std::vector< MultiIndex<3> > Indices3Vector;
00055
00056             Point<2,X> _point;
00057             Indices3Vector _element_indices;
00058             Indices2Vector _border_indices;
00059
00062             void addSubDomainIndex(MultiIndex<3> const &index)
00063             {
00064                 MultiIndex<3>::less compare;
00065                 Indices3Vector::iterator it;
00066                 for(it=_element_indices.begin();
00067                    it!=_element_indices.end();
00068                    ++it)
00069                 {
00070                     if(compare(*it,index))
00071                         continue;
00072                     else if(compare(index,*it))
00073                         break;
00074                 }
00075                 _element_indices.insert(it,index);
00076             };
00077
00080             void addBorderIndex(MultiIndex<2> const &index)
00081             {
00082                 MultiIndex<2>::less compare;
00083                 Indices2Vector::iterator it;
00084                 for(it=_border_indices.begin();
00085                    it!=_border_indices.end();
00086                    ++it)
00087                 {
00088                     if(compare(*it,index))
00089                         continue;
00090                     else if(compare(index,*it))
00091                         break;
00092                 }
00093                 _border_indices.insert(it,index);
00094             };
00095
00096             public:
00098             Node( Point<2,X> const &point)
00099                 : _point(point) {};
00100
00102             Point<2,X> const &point() const
00103             {
00104                 return _point;
00105             };
00106
00108             inline int supportSubDomains() const
00109             {
00110                 return _element_indices.size();
00111             };
00112
00114             MultiIndex<3> const &subDomainIndex(int const &index) const
00115             {
00116                 if(index<0 || supportSubDomains()<=index)

```

```

00117         qFatal("SemSolver::SemSpace::Node::subDomainIndex - ERROR : t
here is"\
00118             "no element with index element_index.");
00119         return _element_indices[index];
00120     };
00121
00122     inline int supportBorders() const
00123     {
00124         return _border_indices.size();
00125     };
00126
00127     MultiIndex<2> const &borderIndex(int const &index) const
00128     {
00129         if(index<0 || supportBorders()<=index)
00130             qFatal("");
00131         return _border_indices[index];
00132     };
00133     friend class SemSpace;
00134 };
00135
00136 class Element
00137 : public Function<Point<2,X>, X>
00138 {
00139     SemSpace const *_space;
00140     std::vector<X> _coefficients;
00141     std::vector< SemFunction<2,X> *> _base;
00142
00143 public:
00144     Element(SemSpace const *space, Vector<X> const &coefficients)
00145     : _space(space)
00146     {
00147         for (int i=0; i<coefficients.dim(); ++i)
00148         {
00149             _coefficients.push_back(coefficients[i]);
00150         }
00151     };
00152     ~Element() {};
00153
00154     X evaluate(Point<2,X> const &x) const
00155     {
00156         double result=0;
00157         if(_space->_geometry.contains(x))
00158         {
00159             for(unsigned i=0; i<_space->nodes(); ++i)
00160             {
00161                 result += _coefficients[i]*_space->baseFunction(i)->evalu
ate(x);
00162             }
00163         }
00164         return result;
00165     };
00166 };
00167
00168 typedef MultiIndex<2>::less Index2Order;
00169 typedef MultiIndex<3>::less Index3Order;
00170 typedef std::vector<Node> NodesVector;
00171 typedef std::vector< SemFunction<2,X> * > SemFunctionsVector;
00172 typedef PointsMap<2, X, int> NodesMap;
00173 typedef typename NodesMap::ConstIterator NodeConstIterator;
00174 typedef std::map< MultiIndex<3>, int, Index3Order > ElementsMap;
00175 typedef typename ElementsMap::const_iterator ElementConstIterator;
00176 typedef std::map< MultiIndex<2>, MultiIndex<3>, Index2Order >\

```

```

00182         BordersMap;
00183     typedef std::vector<int> BordersVector;
00184     typedef std::map< MultiIndex<3>, double, Index3Order > WeightsMap;
00185     typedef typename Polygonation<2,X>::Element SubDomain;
00186
00187     protected:
00188
00189         SemParameters<X> const &_parameters;
00190         SemGeometry<2,X> const &_geometry;
00191
00192     private:
00193         NodesVector _nodes;
00194         SemFunctionsVector _base;
00195
00196         NodesMap _point_map;
00197         ElementsMap _element_map;
00198         BordersMap _border_map;
00199         std::map<int, int> _border_ids;
00200         BordersVector _borders;
00201         WeightsMap _weights;
00202
00204     inline int addSubDomainNode(MultiIndex<3> const &index,
00205                               Point<2,X> const &point)
00206     {
00207         int i;
00208         NodeConstIterator it = _point_map.find(point);
00209         if(it==_point_map.end())
00210         {
00211             i = _nodes.size();
00212             _point_map.insert(point, i);
00213             _nodes.push_back(point);
00214         }
00215         else
00216             i = it->second;
00217         _nodes[i].addSubDomainIndex(index);
00218         _element_map[index] = i;
00219         return i;
00220     };
00221
00222     inline int addBorderNode(MultiIndex<2> const &border_index,
00223                             MultiIndex<3> const &element_index)
00224     {
00225         ElementConstIterator it = _element_map.find(element_index);
00226         #ifdef SEMDEBUG
00227             if(it==_element_map.end())
00228                 qFatal("SemSolver::SemSpace::addBorderNode - ERROR : there is no
00229 element\"
00230                        " with index element_index.");
00231         #endif
00232         int i = it->second;
00233         _nodes[i].addBorderIndex(border_index);
00234         _border_map[border_index] = element_index;
00235         return i;
00236     };
00237
00238     inline void addWeight(MultiIndex<3> const &index, double const &weight)
00239     {
00240         _weights[index] = weight;
00241     };
00242
00243     inline int setBaseRestrictionPolynomialFunciton(int const &index,

```

```

00244                                     int const &element_index,

00245                                     Polynomial<X> const &px,
00246                                     Polynomial<X> const &py)
00247     {
00248         _base[index]->setPolynomialComponent(element_index,0,px);
00249         _base[index]->setPolynomialComponent(element_index,1,py);
00250         return index;
00251     };
00252
00253 public:
00254
00255 SemSpace(SemGeometry<2,X> const &geometry,
00256          SemParameters<X> const &parameters)
00257     : _parameters(parameters),
00258       _geometry(geometry),
00259       _point_map(parameters.tolerance())
00260 {
00261     int N = parameters.degree();
00262     int M = subDomains();
00263
00264     // compute legendre polynomials
00265
00266     Polynomial<X> _l; // _legendre_0
00267     Polynomial<X> _x; // _legendre_1
00268     Polynomial<X> LN;
00269     Polynomial<X> DLN;
00270     {
00271         _l.setDegree(0);
00272         _l.setCoefficient(0,1.);
00273         _x.setDegree(1);
00274         _x.setCoefficient(0,0.);
00275         _x.setCoefficient(1,1.);
00276
00277         Polynomial<X> _legendre_k1 = _l;
00278         Polynomial<X> _legendre_k2 = _x;
00279         Polynomial<X> t1, t2;
00280         for (int k=1; k<N; ++k)
00281         {
00282             t1 = _x * _legendre_k2;
00283             t1 *= (2.*k+1.)/(k+1.);
00284             t2 = _legendre_k1;
00285             t2 *= k/(k+1.);
00286             _legendre_k1 = _legendre_k2;
00287             _legendre_k2 = t1-t2;
00288         }
00289         LN = _legendre_k2;
00290         DLN = LN.derivative();
00291     }
00292
00293     // compute GLL nodes
00294
00295     std::vector<double> gll_nodes = DLN.zeros();
00296     gll_nodes.push_back(-1.);
00297     gll_nodes.push_back(1.);
00298     std::sort(gll_nodes.begin(), gll_nodes.end());
00299
00300     // compute GLL polynomials
00301
00302     std::vector< Polynomial<X> > gll_poly(N+1);
00303     {
00304         // i = 0

```

```

00306         gll_poly[0] = _1-_x; // (1-x^2) / (x-x_i)
00307         gll_poly[0] *= DLN; // [ (1-x^2) * DLN(x) ] / (x-x_i)
00308         gll_poly[0] /= -N*(N+1)*LN(-1.); // (-1) / [ N*(N+1) ] * [ (1-x^2
    )
00309         // * DLN(x) ] / [ (x-x_i) * LN(x_i) ]
00310
00311         // 0 < i < N
00312         for (int i=1; i<N; ++i)
00313         {
00314             gll_poly[i] = DLN.ruffini(gll_nodes[i]); // DLN(x) / (x-x_i)
00315             gll_poly[i] *= _1-_x*_x; // [ (1-x^2) * DLN(x) ] / (x-x_i)
00316             gll_poly[i] /= -N*(N+1)*LN(gll_nodes[i]); // (-1) / [ N*(N+1)
    ]
00317             // * [ (1-x^2) * DLN(x) ]
00318             // / [ (x-x_i) * LN(x_i) ]
00319         }
00320
00321         // i = N
00322         gll_poly[N] = _1+_x; // (-1) * (1-x^2) / (x-x_i)
00323         gll_poly[N] *= DLN; // (-1) * [ (1-x^2) * DLN(x) ] / (x-x_i)
00324         gll_poly[N] /= N*(N+1)*LN(1.); // (-1) / [ N*(N+1) ] * [ (1-x^2)
    // * DLN(x) ] / [ (x-x_i) * LN(x_i) ]
00326     }
00327
00328     // compute GLL weights
00329
00330     std::vector<double> gll_weights(N+1);
00331     for( int i=0; i<=N; ++i)
00332     {
00333         gll_weights[i] = 2./(N*(N+1.));
00334         gll_weights[i] /= LN(gll_nodes[i]) * LN(gll_nodes[i]);
00335     }
00336
00337     // compute maps
00338
00339     std::vector< BilinearTransformation<X> > maps;
00340     for(int i=0; i<M; ++i)
00341     {
00342         SubDomain const &element = geometry.subDomains().element(i);
00343         maps.push_back( BilinearTransformation<X>(element.geometry(),
00344             _parameters.tolerance()
    ) );
00345     }
00346
00347     // get borders ids form PSLG
00348     for(unsigned i=0; i<geometry.domain().segments(); ++i)
00349         _border_ids[i] = geometry.domain().segment(i).number;
00350
00351     // compute nodes
00352
00353     Point<2,X> x_hat;
00354     Point<2,X> x;
00355     double weight;
00356     MultiIndex<3> element_index;
00357     MultiIndex<2> border_index;
00358
00359     for(int i=0; i<M; ++i)
00360     {
00361         int left = _geometry.subDomains().element(i).neighbour(0);
00362         int bottom = _geometry.subDomains().element(i).neighbour(1);
00363         int right = _geometry.subDomains().element(i).neighbour(2);
00364         int top = _geometry.subDomains().element(i).neighbour(3);

```

```

00365         if(left<0) // left is on boundary
00366         {
00367             _borders.push_back(-left-1);
00368             left = borders();
00369         }
00370         else
00371             left=0;
00372         if(bottom<0) // bottom is on boundary
00373         {
00374             _borders.push_back(-bottom-1);
00375             bottom = borders();
00376         }
00377         else
00378             bottom=0;
00379         if(right<0) // right is on boundary
00380         {
00381             _borders.push_back(-right-1);
00382             right = borders();
00383         }
00384         else
00385             right=0;
00386         if(top<0) // top is on boundary
00387         {
00388             _borders.push_back(-top-1);
00389             top = borders();
00390         }
00391         else
00392             top=0;
00393
00394         // bottom left vertex
00395         x_hat = Point<2,X>(gll_nodes[0],gll_nodes[0]);
00396         x = maps[i].evaluate(x_hat);
00397
00398         element_index.setSubIndex(0,i);
00399         element_index.setSubIndex(1,0);
00400         element_index.setSubIndex(2,0);
00401         weight = gll_weights[0]*gll_weights[0]*\
00402             std::abs(maps[i].evaluateJacobianDeterminant(x_hat));
00403         addWeight(element_index,weight);
00404         addSubDomainNode(element_index, x);
00405
00406         if(bottom)
00407         {
00408             border_index.setSubIndex(0,bottom);
00409             border_index.setSubIndex(1,0);
00410             addBorderNode(border_index,element_index);
00411         }
00412         if(left)
00413         {
00414             border_index.setSubIndex(0,left);
00415             border_index.setSubIndex(1,0);
00416             addBorderNode(border_index,element_index);
00417         }
00418
00419         // bottom edge
00420         for (int j=1; j<N; ++j)
00421         {
00422             x_hat = Point<2,X>(gll_nodes[j],gll_nodes[0]);
00423             x = maps[i].evaluate(x_hat);
00424
00425             element_index.setSubIndex(0,i);
00426             element_index.setSubIndex(1,j);

```



```

00427         element_index.setSubIndex(2, 0);
00428         weight = gll_weights[j]*gll_weights[0]*\
00429             std::abs(maps[i].evaluateJacobianDeterminant(x_hat))
;
00430         addWeight(element_index, weight);
00431         addSubDomainNode(element_index, x);
00432
00433         if (bottom)
00434         {
00435             border_index.setSubIndex(0, bottom);
00436             border_index.setSubIndex(1, j);
00437             addBorderNode(border_index, element_index);
00438         }
00439     }
00440
00441     // bottom right vertex
00442     x_hat = Point<2,X>(gll_nodes[N], gll_nodes[0]);
00443     x = maps[i].evaluate(x_hat);
00444
00445     element_index.setSubIndex(0, i);
00446     element_index.setSubIndex(1, N);
00447     element_index.setSubIndex(2, 0);
00448     weight = gll_weights[N]*gll_weights[0]*\
00449         std::abs(maps[i].evaluateJacobianDeterminant(x_hat));
00450     addWeight(element_index, weight);
00451     addSubDomainNode(element_index, x);
00452
00453     if (bottom)
00454     {
00455         border_index.setSubIndex(0, bottom);
00456         border_index.setSubIndex(1, N);
00457         addBorderNode(border_index, element_index);
00458     }
00459     if (right)
00460     {
00461         border_index.setSubIndex(0, right);
00462         border_index.setSubIndex(1, 0);
00463         addBorderNode(border_index, element_index);
00464     }
00465
00466     for (int k=1; k<N; ++k)
00467     {
00468         // left edge
00469         x_hat = Point<2,X>(gll_nodes[0], gll_nodes[k]);
00470         x = maps[i].evaluate(x_hat);
00471
00472         element_index.setSubIndex(0, i);
00473         element_index.setSubIndex(1, 0);
00474         element_index.setSubIndex(2, k);
00475         weight = gll_weights[0]*gll_weights[k]*\
00476             std::abs(maps[i].evaluateJacobianDeterminant(x_hat))
;
00477         addWeight(element_index, weight);
00478         addSubDomainNode(element_index, x);
00479
00480         if (left)
00481         {
00482             border_index.setSubIndex(0, left);
00483             border_index.setSubIndex(1, k);
00484             addBorderNode(border_index, element_index);
00485         }
00486

```

```

00487         // interior
00488         for(int j=1; j<N; ++j)
00489         {
00490             x_hat = Point<2,X>(gll_nodes[j],gll_nodes[k]);
00491             x = maps[i].evaluate(x_hat);
00492
00493             element_index.setSubIndex(0,i);
00494             element_index.setSubIndex(1,j);
00495             element_index.setSubIndex(2,k);
00496             weight = gll_weights[j]*gll_weights[k]*\
00497                 std::abs(maps[i].evaluateJacobianDeterminant(x_h
at));
00498             addWeight(element_index,weight);
00499             addSubDomainNode(element_index, x);
00500         }
00501
00502         // right edge
00503         x_hat = Point<2,X>(gll_nodes[N],gll_nodes[k]);
00504         x = maps[i].evaluate(x_hat);
00505
00506         element_index.setSubIndex(0,i);
00507         element_index.setSubIndex(1,N);
00508         element_index.setSubIndex(2,k);
00509         weight = gll_weights[N]*gll_weights[k]*\
00510             std::abs(maps[i].evaluateJacobianDeterminant(x_hat))
;
00511         addWeight(element_index,weight);
00512         addSubDomainNode(element_index, x);
00513
00514         if(right)
00515         {
00516             border_index.setSubIndex(0,right);
00517             border_index.setSubIndex(1,k);
00518             addBorderNode(border_index,element_index);
00519         }
00520     }
00521
00522     // top left vertex
00523     x_hat = Point<2,X>(gll_nodes[0],gll_nodes[N]);
00524     x = maps[i].evaluate(x_hat);
00525
00526     element_index.setSubIndex(0,i);
00527     element_index.setSubIndex(1,0);
00528     element_index.setSubIndex(2,N);
00529     weight = gll_weights[0]*gll_weights[N]*\
00530         std::abs(maps[i].evaluateJacobianDeterminant(x_hat));
00531     addWeight(element_index,weight);
00532     addSubDomainNode(element_index, x);
00533
00534     if(top)
00535     {
00536         border_index.setSubIndex(0,top);
00537         border_index.setSubIndex(1,0);
00538         addBorderNode(border_index,element_index);
00539     }
00540     if(left)
00541     {
00542         border_index.setSubIndex(0,left);
00543         border_index.setSubIndex(1,N);
00544         addBorderNode(border_index,element_index);
00545     }
00546

```

```

00547         // top edge
00548         for(int j=1; j<N; ++j)
00549         {
00550             x_hat = Point<2,X>(gll_nodes[j],gll_nodes[N]);
00551             x = maps[i].evaluate(x_hat);
00552
00553             element_index.setSubIndex(0,i);
00554             element_index.setSubIndex(1,j);
00555             element_index.setSubIndex(2,N);
00556             weight = gll_weights[j]*gll_weights[N]*\
00557                     std::abs(maps[i].evaluateJacobianDeterminant(x_hat))
00558         };
00559         addWeight(element_index,weight);
00560         addSubDomainNode(element_index, x);
00561
00562         if(top)
00563         {
00564             border_index.setSubIndex(0,top);
00565             border_index.setSubIndex(1,j);
00566             addBorderNode(border_index,element_index);
00567         }
00568
00569         // top right vertex
00570         x_hat = Point<2,X>(gll_nodes[N],gll_nodes[N]);
00571         x = maps[i].evaluate(x_hat);
00572
00573         element_index.setSubIndex(0,i);
00574         element_index.setSubIndex(1,N);
00575         element_index.setSubIndex(2,N);
00576         weight = gll_weights[N]*gll_weights[N]*\
00577                 std::abs(maps[i].evaluateJacobianDeterminant(x_hat));
00578         addWeight(element_index,weight);
00579         addSubDomainNode(element_index, x);
00580
00581         if(top)
00582         {
00583             border_index.setSubIndex(0,top);
00584             border_index.setSubIndex(1,N);
00585             addBorderNode(border_index,element_index);
00586         }
00587         if(right)
00588         {
00589             border_index.setSubIndex(0,right);
00590             border_index.setSubIndex(1,N);
00591             addBorderNode(border_index,element_index);
00592         }
00593     }
00594
00595
00596     // base functions
00597     std::vector< PolynomialFunction<2,X> > zero_polys(M);
00598     for(unsigned i=0; i<_nodes.size(); ++i)
00599         _base.push_back(new SemFunction<2,X>(_geometry, zero_polys,maps));
00600
00601     for(int i=0; i<M; ++i)
00602     {
00603         for(int j=0; j<=N; ++j)
00604         {
00605             for(int k=0; k<=N; ++k)
00606             {

```

```

00607         element_index.setSubIndex(0,i);
00608         element_index.setSubIndex(1,j);
00609         element_index.setSubIndex(2,k);
00610         int index = subDomainIndex(element_index);
00611         setBaseRestrictionPolynomialFunciton(index,i,gll_poly[j],
                                gll_poly[k]);
00612     }
00613 }
00614 }
00615 }
00616 }
00617
00619 unsigned nodes() const
00620 {
00621     return _nodes.size();
00622 };
00623
00625 inline const Node &node(const unsigned &index) const
00626 {
00627     if( index>=nodes() )
00628         qFatal("SemSolver::SemSpace::node : index out of range");
00629     return _nodes[index];
00630 };
00631
00633 inline int subDomains() const
00634 {
00635     return _geometry.subDomains().size();
00636 };
00637
00639 inline int const &subDomainIndex(MultiIndex<3> const &index) const
00640 {
00641     std::map<MultiIndex<3>, int, MultiIndex<3>::less>::const_iterator it
=
00642         _element_map.find(index);
00643 #ifdef SEMDEBUG
00644     if(it==_element_map.end())
00645         qFatal("SemSolver::SemSpace::subDomainIndex - ERROR : there is no
elemen"\
00646             "t with multi-index index.");
00647 #endif
00648     return it->second;
00649 }
00650
00652 inline Node const &subDomainNode(MultiIndex<3> const &index) const
00653 {
00654     return node(subDomainIndex(index));
00655 };
00656
00658 inline double const &subDomainWeight(MultiIndex<3> const &index) const
00659 {
00660     std::map<MultiIndex<3>, double, MultiIndex<3>::less>::const_iterator
it =
00661         _weights.find(index);
00662     if(it==_weights.end())
00663     {
00664         qFatal("SemSolver::SemSpace::subDomainWeight - ERROR : there is n
o weigh"\
00665             "t with multi-index index.");
00666     }
00667     return it->second;
00668 };
00669

```

```

00671         inline unsigned borders() const
00672         {
00673             return _borders.size();
00674         };
00675
00677         inline int border(int const &i) const
00678         {
00679             return _borders[i];
00680         };
00681
00683         inline int borderId(int const &border) const
00684         {
00685             std::map<int,int>::const_iterator it = _border_ids.find(border);
00686             if(it!=_border_ids.end())
00687                 return it->second;
00688             return -1;
00689         };
00690
00692         inline MultiIndex<3> const &borderSubDomainIndex(MultiIndex<2> const &index) const
00693         {
00694             std::map<MultiIndex<2>, MultiIndex<3>, MultiIndex<2>::less>::const_iterator it
00695             = _border_map.find(index);
00696             #ifdef SEMDEBUG
00697                 if(it==_border_map.end())
00698                     qFatal("SemSolver::SemSpace::borderSubDomainIndex - ERROR : there
is no "\
00699                         "border with multi-index index.");
00700             #endif
00701             return it->second;
00702         };
00703
00705         inline int const &borderIndex(MultiIndex<2> const &index) const
00706         {
00707             return subDomainIndex(borderSubDomainIndex(index));
00708         };
00709
00711         inline Node const &borderNode(MultiIndex<2> const &index) const
00712         {
00713             return subDomainNode(borderSubDomainIndex(index));
00714         };
00715
00717         inline double const &borderWeight(MultiIndex<2> const &index) const
00718         {
00719             return subDomainWeight(borderSubDomainIndex(index));
00720         };
00721
00723         double scalarProduct(Element first, Element second)
00724         {
00725             /******
00726             return 0;
00727             */
00728         };
00729
00730         SemFunction<2,X> const *baseFunction(const unsigned &index) const
00731         {
00732             #if SEMDEBUG
00733                 if(index>=nodes())
00734                     qFatal("SemSolver::HilbertSpace::baseFunction - ERROR : index out
of ran"\
00735                         "ge");
00736             #endif

```

```

00737         return _base[index];
00738     };
00739
00741     inline int const &degree() const
00742     {
00743         return _parameters.degree();
00744     };
00745 };
00746 }
00747
00748 #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

```

00001 #ifndef SEQUENCE_HPP
00002 #define SEQUENCE_HPP
00003
00004 #include <list>
00005
00006 namespace SemSolver
00007 {
00008     typedef std::list<int> Sequence;
00009 };
00010
00011 #endif // SEQUENCE_HPP

```

7.101 sequencelist.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

```

00001 #ifndef SEQUENCESLIST_HPP
00002 #define SEQUENCESLIST_HPP
00003
00004 #include <list>
00005
00006 #include <SemSolver/sequence.hpp>
00007
00008 namespace SemSolver
00009 {
00010     typedef std::list<Sequence> Sequences_list;
00011 };
00012
00013 #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

```

00001 #ifndef IO_SUBDOMAINS_HPP
00002 #define IO_SUBDOMAINS_HPP
00003
00004 #include <QFile>
00005 #include <QTextStream>
00006 #ifdef SEMDEBUG
00007 #   include <QDebug>
00008 #endif
00009
00010 #include <SemSolver/polygonation.hpp>
00011
00012 namespace SemSolver
00013 {
00014     namespace IO
00015     {
00016         template<class X>
00019         bool read_subdomains(QFile *file,
                                Polygonation<2, X> &sub_domains)
00020         {
00021             if(!file->open(QIODevice::ReadOnly))
00022                 return false;
00023             QTextStream input(file);
00024             sub_domains.clear();
00025             int size, polygon_size, id, neighbour;
00026             double x, y;
00027             Polygon<2,X> polygon;
00028             std::vector<int> neighbours;
00029             input >> size;
00030             for(int i=0; i<size; ++i)
00031             {
00032                 polygon.clear();
00033                 neighbours.clear();
00034                 input >> id;
00035                 #ifdef SEMDEBUG
00036                     if(id!=i+1)
00037                     {
00038                         qWarning() << "corrupted domains file";
00039                         return false;
00040                     }
00041                 #endif
00042                 input >> polygon_size;
00043                 for(int j=0; j<polygon_size; ++j)
00044                 {
00045                     input >> x;
00046                     input >> y;
00047                     input >> neighbour;
00048                     polygon.push_back(Point<2,X>(x,y));
00049                     neighbours.push_back(neighbour);
00050

```



```

00051         }
00052         sub_domains.addElement(polygon, neighbours);
00053     }
00054     file->close();
00055     return true;
00056 };
00057
00059     template<class X>
00060     bool write_subdomains(Polygonation<2, X> const &sub_domains,
00061                           QFile *file)
00062 {
00063     if(!file->open(QIODevice::WriteOnly))
00064         return false;
00065     QTextStream output(file);
00066     int n = sub_domains.size();
00067     output << n << "\n";
00068     for(int i=0; i<n; ++i)
00069     {
00070         int m = sub_domains.element(i).size();
00071         output << i+1 << "\t" << m << "\n";
00072         std::vector<int>::const_iterator it = sub_domains.element(i).neighboursBe
00073 gin();
00074         for(int j=0; j<m; ++j)
00075         {
00076             output << sub_domains.element(i).vertex(j).x() << "\t";
00077             output << sub_domains.element(i).vertex(j).y() << "\t";
00078             output << *it++ << "\n";
00079         }
00080         file->close();
00081         return true;
00082     };
00083 };
00084 };
00085
00086 #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

```

00001 #ifndef VECTOR_HPP
00002 #define VECTOR_HPP
00003
00004 namespace SemSolver
00005 {
00007     template<class X>
00008     class Vector;
00009 };
00010
00011 #include <tnt_array1d.h>
00012
00013 #include <SemSolver/point.hpp>
00014
00015 namespace SemSolver
00016 {
00017     template<class X>
00018     class Vector
00019     : public TNT::Array1D<X>
00020     {
00021     typedef TNT::Array1D<X> TNT_array_1d;
00022     public:
00024         Vector() : TNT_array_1d() {};
00025
00027         Vector(TNT_array_1d const &vector) : TNT_array_1d(vector) {};
00028
00030         Vector(int dimension) : TNT_array_1d(dimension) {};
00031         Vector(int dimension, X const &value) : TNT_array_1d(dimension, value) {}
00032     };
00033
00034     template<int d>
00035     Vector(Point<d, X> const &point)
00036     : TNT_array_1d(d)
00037     {
00038         for(int i=0; i<d; ++i)
00039             (*this)[i] = point.cartesian(i);
00040     }
00041
00043     inline int rows() const
00044     {
00045         return this->dim1();
00046     };
00047 };
00048

```

```

00050     template<class X>
00051     Vector<X> operator +(Vector<X> const &vec1, Vector<X> const &vec2)
00052     {
00053         if(vec1.dim() != vec2.dim())
00054             qFatal("");
00055         return vec1+vec2;
00056     };
00057
00059     template<class X>
00060     X scalar(Vector<X> const &vec1, Vector<X> const &vec2)
00061     {
00062         if(vec1.dim() != vec2.dim())
00063             qFatal("SemSolver::Vector::scalar - ERROR : vec1 and vec2 must have t
he same\"
00064                  \" dimension.");
00065         X product = 0;
00066         for(int i=0; i<vec1.dim(); ++i)
00067             product += vec1[i]*vec2[i];
00068         return product;
00069     };
00070 };
00071
00072 #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 ¶meters)
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

```

00001 #ifndef WORKSPACE_HPP
00002 #define WORKSPACE_HPP
00003
00004 #include <QFile>
00005 #include <QString>
00006 #include <QStringList>
00007
00008 #include <SemSolver/semgeometry.hpp>
00009 #include <SemSolver/equation.hpp>
00010 #include <SemSolver/boundaryconditions.hpp>
00011 #include <SemSolver/semparameters.hpp>
00012
00013 #include <SemSolver/IO/geometry.hpp>
00014 #include <SemSolver/IO/equation.hpp>
00015 #include <SemSolver/IO/boundaryconditions.hpp>
00016 #include <SemSolver/IO/parameters.hpp>
00017
00018 namespace SemSolver
00019 {
00020     namespace IO
00021     {
00022         {
00024             bool get_geometries_list_from_workspace(QFile *file,
00025                                                     QStringList &geometries);
00027             bool get_equations_list_from_workspace(QFile *file,
00028                                                    QStringList &equations);
00030             bool get_boundary_conditions_list_from_workspace(QFile *file,
00031                                                            QStringList &boundary_conditi
00032                                                            ons);
00033             bool get_parameters_list_from_workspace(QFile *file,
00034                                                    QStringList &parameters);
00036             template<class X>
00037             bool get_geometry_from_workspace(QFile *file,
00038                                             QString const &name,
00039                                             SemGeometry<2, X> &geometry);
00041             template<class X>
00042             bool get_equation_from_workspace(QFile *file,
00043                                             QString const &name,
00044                                             Equation<2, X> *&equation);
00046             template<class X>
00047             bool get_boundary_conditions_from_workspace(QFile *file,
00048                                                        QString const &name,
00049                                                        BoundaryConditions<2, X> &bc);
00051             template<class X>
00052             bool get_parameters_from_workspace(QFile *file,
00053                                               QString const &name,
00054                                               SemParameters<X> &bc);
00056             bool extract_file_from_workspace(QFile *workspace,
00057                                             QString const &name,
00058                                             QFile *file);
00060             bool add_file_to_workspace(QFile *workspace,
00061                                       QString const &name,
00062                                       QFile *file);
00064             bool remove_file_from_workspace(QFile *workspace,
00065                                            QString const &name);

```

```
00066     };
00067 };
00068
00069 template<class X>
00070 bool SemSolver::IO::get_geometry_from_workspace(QFile *file,
00071                                                const QString &name,
00072                                                SemGeometry<2,X> &geometry)
00073 {
00074     QTemporaryFile *temp_file = new QTemporaryFile();
00075     if(!extract_file_from_workspace(file, name + ".semgeo", temp_file))
00076     {
00077         delete temp_file;
00078         return false;
00079     }
00080     if(!read_geometry(temp_file, geometry))
00081     {
00082         delete temp_file;
00083         return false;
00084     }
00085     delete temp_file;
00086     return true;
00087 };
00088
00089 template<class X>
00090 bool SemSolver::IO::get_equation_from_workspace(QFile *file,
00091                                                const QString &name,
00092                                                Equation<2, X> *&equation)
00093 {
00094     QTemporaryFile *temp_file = new QTemporaryFile();
00095     if(!extract_file_from_workspace(file, name + ".semeqn", temp_file))
00096     {
00097         delete temp_file;
00098         return false;
00099     }
00100     if(!read_equation(temp_file, equation))
00101     {
00102         delete temp_file;
00103         return false;
00104     }
00105     delete temp_file;
00106     return true;
00107 };
00108
00109 template<class X>
00110 bool SemSolver::IO::get_boundary_conditions_from_workspace(QFile *file,
00111                                                           const QString &name,
00112                                                           BoundaryConditions<2, X> &
00113                                                           bc)
00114 {
00115     QTemporaryFile *temp_file = new QTemporaryFile();
00116     if(!extract_file_from_workspace(file, name + ".sembcs", temp_file))
00117     {
00118         delete temp_file;
00119         return false;
00120     }
00121     if(!read_boundary_conditions(temp_file, bc))
00122     {
00123         delete temp_file;
00124         return false;
00125     }
00126     delete temp_file;
00127     return true;
00128 }
```

```
00127 };
00128
00129 template<class X>
00130 bool SemSolver::IO::get_parameters_from_workspace(QFile *file,
00131                                                    const QString &name,
00132                                                    SemParameters<X> &parameters)
00133 {
00134     QTemporaryFile *temp_file = new QTemporaryFile();
00135     if(!extract_file_from_workspace(file, name + ".semprm", temp_file))
00136     {
00137         delete temp_file;
00138         return false;
00139     }
00140     if(!read_parameters(temp_file, parameters))
00141     {
00142         delete temp_file;
00143         return false;
00144     }
00145     delete temp_file;
00146     return true;
00147 };
00148
00149 #endif // WORKSPACE_HPP
```

Index

- ~BilinearTransformation
 - add_file_to_workspace
 - SemSolver::BilinearTransformation, [30](#)
- ~DiffusionConvectionReactionEquation
 - addElement
 - SemSolver::DiffusionConvectionReactionEquation, [41](#)
 - SemSolver::DiffusionConvectionReactionEquation< 2, X >, [46](#)
- ~Element
 - addValue
 - SemSolver::HilbertSpace::Element, [49](#)
 - SemSolver::SemSpace< 2, X >::Element, [55](#)
- ~Equation
 - SemSolver::Equation, [57](#)
- ~Function
 - SemSolver::Function, [59](#)
- ~Homeomorphism
 - SemSolver::Homeomorphism, [64](#)
- ~PSLG
 - SemSolver::PSLG, [117](#)
- ~Polynomial
 - SemSolver::Polynomial, [104](#)
- ~PolynomialFunction
 - SemSolver::PolynomialFunction, [110](#)
- ~Problem
 - SemSolver::Problem, [113](#)
- ~ScriptFunction
 - SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > >, [122](#)
 - SemSolver::ScriptFunction< Point< 2, X >, Y >, [124](#)
 - SemSolver::ScriptFunction< Point< d, X >, Vector< Y > >, [126](#)
 - SemSolver::ScriptFunction< Point< d, X >, Y >, [128](#)
- _geometry
 - SemSolver::SemSpace< 2, X >, [150](#)
- _parameters
 - SemSolver::SemSpace< 2, X >, [150](#)
- add
 - SemSolver::SegmentsMap, [133](#)
- add_file_to_workspace
- SemSolver::IO, [18](#)
- addElement
- SemSolver::Polygonation< 2, X >, [98](#)
- addEquation
- SemSolver::IO::Archive, [26](#)
- Archive
- SemSolver::IO::Archive, [26](#)
- archive.hpp, [155](#)
- attributes
- SemSolver::PSLG::Vertex, [152](#)
- baseFunction
- SemSolver::HilbertSpace, [61](#)
- SemSolver::SemSpace< 2, X >, [147](#)
- begin
- SemSolver::PointsMap< 2, X, Y >, [84](#)
- SemSolver::PointsSet< 2, X >, [92](#)
- BilinearTransformation
- SemSolver::BilinearTransformation, [29](#)
- bilineartransformation.hpp, [157](#)
- border
- SemSolver::SemSpace< 2, X >, [147](#)
- borderId
- SemSolver::SemSpace< 2, X >, [148](#)
- borderIndex
- SemSolver::SemSpace< 2, X >, [148](#)
- SemSolver::SemSpace< 2, X >::Node, [72](#)
- borderNode
- SemSolver::SemSpace< 2, X >, [148](#)
- borders
- SemSolver::SemSpace< 2, X >, [148](#)
- BordersMap
- SemSolver::SemSpace< 2, X >, [146](#)
- borderSubDomainIndex
- SemSolver::SemSpace< 2, X >, [148](#)

- BordersVector
 - SemSolver::SemSpace< 2, X >, 146
- borderType
 - SemSolver::BoundaryConditions< 2, X >, 36
- borderWeight
 - SemSolver::SemSpace< 2, X >, 148
- BoundaryConditions
 - SemSolver::BoundaryConditions< 2, X >, 35, 36
- boundaryConditions
 - SemSolver::Problem, 113
- boundaryconditions.hpp, 162, 169
- build_solution
 - SemSolver::PostProcessor, 21
- buildsolution.hpp, 171
- cgal
 - SemSolver::PolygonWithHoles< 2, X >, 101
 - SemSolver::Segment< 2, X >, 130
- cholesky_solve
 - SemSolver::Solver, 23
- choleskysolve.hpp, 172
- clear
 - SemSolver::BoundaryConditions< 2, X >, 36
 - SemSolver::PointsMap< 2, X, Y >, 84
 - SemSolver::PointsSet< 2, X >, 92
 - SemSolver::Polygonation< 2, X >, 98
 - SemSolver::Polygonation< 2, X >::Element, 51
 - SemSolver::PSLG, 117
- clearBoundaryConditions
 - SemSolver::Problem, 113
- clearEquation
 - SemSolver::Problem, 113
- clearGeometry
 - SemSolver::Problem, 114
- clearParameters
 - SemSolver::Problem, 114
- closeRead
 - SemSolver::IO::Archive, 26
- closeWrite
 - SemSolver::IO::Archive, 26
- coefficient
 - SemSolver::Polynomial, 104
- columns
 - SemSolver::Matrix, 68
- compute_algebraic_system
 - SemSolver::Assembler, 15
- compute_border_matrix
 - SemSolver::Assembler, 15
- compute_border_vector
 - SemSolver::Assembler, 15
- compute_convection_matrix
 - SemSolver::Assembler, 15
- compute_diffusion_matrix
 - SemSolver::Assembler, 16
- compute_forcing_vector
 - SemSolver::Assembler, 16
- compute_plot_data
 - SemSolver::PostProcessor, 21
- compute_polygon_with_holes_from_pslg
 - SemSolver::PreProcessor, 22
- compute_polygonation_from_pslg
 - SemSolver::PreProcessor, 22
- compute_reaction_matrix
 - SemSolver::Assembler, 16
- compute_solution_hull
 - SemSolver::PostProcessor, 21
- compute_vertices_sequences_from_pslg
 - SemSolver::PreProcessor, 23
- computealgebraicsystem.hpp, 173
- computebordermatrix.hpp, 176
- computebordervector.hpp, 177
- computeconvectionmatrix.hpp, 179
- computediffusionmatrix.hpp, 181
- computeforcingvector.hpp, 183
- computeplotdata.hpp, 184
- computepolygonationfrompslg.hpp, 186
- computepolygonwithholesfrompslg.hpp, 188
- computereactionmatrix.hpp, 193
- computesolutionhull.hpp, 194
- conditions
 - SemSolver::BoundaryConditions< 2, X >, 36
- ConstIterator
 - SemSolver::PointsBimap< 2, X >, 76
 - SemSolver::PointsMap< 2, X, Y >, 83
 - SemSolver::PointsSet< 2, X >, 91
 - SemSolver::SegmentsMap, 132
- contains
 - SemSolver::Polygon< 2, X >, 97
 - SemSolver::Polygonation< 2, X >::Element, 51
 - SemSolver::SegmentsMap, 133

- SemSolver::SemGeometry< 2, X >, elementIndicesAt
 - 140
- convection
 - SemSolver::DiffusionConvectionReactionEquation
 - 41
 - SemSolver::DiffusionConvectionReactionEquation< 2, X >, 46
- degree
 - SemSolver::Polynomial, 104
 - SemSolver::SemParameters, 142
 - SemSolver::SemSpace< 2, X >, 148
- derivative
 - SemSolver::Polynomial, 104
- diffusion
 - SemSolver::DiffusionConvectionReactionEquation
 - 41
 - SemSolver::DiffusionConvectionReactionEquation< 2, X >, 46
- DIFFUSION_CONVECTION_REACTION
 - SemSolver::Equation, 57
- DiffusionConvectionReactionEquation
 - SemSolver::DiffusionConvectionReactionEquation
 - 41
 - SemSolver::DiffusionConvectionReactionEquation< 2, X >, 46
- diffusionconvectionreactionequation.hpp, 196
- dimension
 - SemSolver::HilbertSpace, 61
- DIRICHLET
 - SemSolver::BoundaryConditions< 2, X >, 35
- dirichletData
 - SemSolver::BoundaryConditions< 2, X >, 37
- domain
 - SemSolver::SemGeometry< 2, X >, 140
- Element
 - SemSolver::HilbertSpace::Element, 49
 - SemSolver::Polygonation< 2, X >::Element, 51
 - SemSolver::SemSpace< 2, X >::Element, 55
- element
 - SemSolver::Polygonation< 2, X >, 99
- ElementConstIterator
 - SemSolver::SemSpace< 2, X >, 146
- elementIndicesAt
 - SemSolver::Polygonation< 2, X >, 99
- EquationMap
 - SemSolver::SemSpace< 2, X >, 146
- Equation
 - SemSolver::Equation< 2, X, Y >, 85
 - SemSolver::PointsMap< 2, X, Y >, 85
 - SemSolver::PointsSet< 2, X >, 92
- entries
 - SemSolver::IO::Archive, 27
- Equation
 - SemSolver::Equation, 57
- equation
 - SemSolver::Problem, 114
- equation.hpp, 201, 202
- erase
 - SemSolver::PointsMap< 2, X, Y >, 85
 - SemSolver::PointsSet< 2, X >, 92
- eraseId
 - SemSolver::PointsBimap< 2, X >, 77
- erasePoint
 - SemSolver::PointsBimap< 2, X >, 77
- evaluate
 - SemSolver::BilinearTransformation, 30
 - SemSolver::Function, 59
 - SemSolver::PolynomialFunction, 111
 - SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > >, 122
 - SemSolver::ScriptFunction< Point< 2, X >, Y >, 124
 - SemSolver::ScriptFunction< Point< d, X >, Vector< Y > >, 126
 - SemSolver::ScriptFunction< Point< d, X >, Y >, 128
 - SemSolver::SemFunction< 2, X >, 137
 - SemSolver::SemSpace< 2, X >::Element, 55
- evaluateInverse
 - SemSolver::BilinearTransformation, 30
 - SemSolver::Homeomorphism, 65
- evaluateJacobianDeterminant
 - SemSolver::BilinearTransformation, 30
- evaluateRestrictionGradient
 - SemSolver::SemFunction< 2, X >, 137
- evaluateTransposeInverseJacobian

- SemSolver::BilinearTransformation, 31
- extract_file_from_workspace
 - SemSolver::IO, 19
- extractFile
 - SemSolver::IO::Archive, 27
- find
 - SemSolver::PointsMap< 2, X, Y >, 86
 - SemSolver::PointsSet< 2, X >, 93
- findId
 - SemSolver::PointsBimap< 2, X >, 77, 78
- findPoint
 - SemSolver::PointsBimap< 2, X >, 78
- forcing
 - SemSolver::DiffusionConvectionReactionEquation, 42
 - SemSolver::DiffusionConvectionReactionEquationBimapspace, 208
 - 2, X >, 46
- Function
 - SemSolver::Function, 59
- function.hpp, 205
- FunctionConstIterator
 - SemSolver::BoundaryConditions< 2, X >, 34
- FunctionPtr
 - SemSolver::BoundaryConditions< 2, X >, 34
- FunctionsMap
 - SemSolver::BoundaryConditions< 2, X >, 35
- geometry
 - SemSolver::Polygonation< 2, X >::Element, 52
 - SemSolver::Problem, 114
- geometry.hpp, 206
- get_boundary_conditions_from_workspace
 - SemSolver::IO, 19
- get_boundary_conditions_list_from_workspace
 - SemSolver::IO, 19
- get_equation_from_workspace
 - SemSolver::IO, 19
- get_equations_list_from_workspace
 - SemSolver::IO, 19
- get_geometries_list_from_workspace
 - SemSolver::IO, 19
- get_geometry_from_workspace
 - SemSolver::IO, 19
- get_parameters_from_workspace
 - SemSolver::IO, 19
- get_parameters_list_from_workspace
 - SemSolver::IO, 19
- has
 - SemSolver::PointsMap< 2, X, Y >, 86
 - SemSolver::PointsSet< 2, X >, 93
 - SemSolver::SegmentsMap, 133
- hasId
 - SemSolver::PointsBimap< 2, X >, 79
- hasPoint
 - SemSolver::PointsBimap< 2, X >, 79
- hasPointOn
 - SemSolver::PointsBimap< 2, X >, 79
- hole
 - SemSolver::PSLG, 117
- holes
 - SemSolver::PSLG, 117
- Homeomorphism
 - SemSolver::Homeomorphism, 64
- homeomorphism.hpp, 209
- id
 - SemSolver::PointsBimap< 2, X >, 79
 - SemSolver::SegmentsMap, 133
- Index2Order
 - SemSolver::SemSpace< 2, X >, 146
- Index3Order
 - SemSolver::SemSpace< 2, X >, 146
- insert
 - SemSolver::PointsBimap< 2, X >, 80
 - SemSolver::PointsMap< 2, X, Y >, 86, 87
 - SemSolver::PointsSet< 2, X >, 94
 - SemSolver::SegmentsMap, 133
- insertPoint
 - SemSolver::PointsBimap< 2, X >, 80
- intersect
 - SemSolver::Segment< 2, X >, 130
 - SemSolver::SegmentsMap, 134
- intersectInteriorly
 - SemSolver::Segment< 2, X >, 130
 - SemSolver::SegmentsMap, 134
- isConsistentWith
 - SemSolver::SegmentsMap, 134

- isDefined
 - SemSolver::Problem, [114](#)
- isEmpty
 - SemSolver::PointsMap< 2, X, Y >, [88](#)
 - SemSolver::PointsSet< 2, X >, [95](#)
- isQuadrangulation
 - SemSolver::Polygonation< 2, X >, [99](#)
- Iterator
 - SemSolver::PointsMap< 2, X, Y >, [83](#)
 - SemSolver::PointsSet< 2, X >, [91](#)
 - SemSolver::SegmentsMap, [132](#)
- KeyType
 - SemSolver::PointsBimap< 2, X >, [76](#)
 - SemSolver::PointsMap< 2, X, Y >, [83](#)
 - SemSolver::PointsSet< 2, X >, [91](#)
- labels
 - SemSolver::BoundaryConditions< 2, X >, [37](#)
- lu_solve
 - SemSolver::Solver, [24](#)
- lusolve.hpp, [210](#)
- map
 - SemSolver::SemFunction< 2, X >, [138](#)
- MappedType
 - SemSolver::PointsBimap< 2, X >, [76](#)
 - SemSolver::PointsMap< 2, X, Y >, [83](#)
 - SemSolver::PointsSet< 2, X >, [91](#)
- marker
 - SemSolver::PSLG::Segment, [129](#)
 - SemSolver::PSLG::Vertex, [152](#)
- Matrix
 - SemSolver::Matrix, [67](#)
- matrix.hpp, [211](#)
- mml
 - SemSolver::DiffusionConvectionReactionEquation< 2, X >, [42](#)
 - SemSolver::DiffusionConvectionReactionEquation< 2, X >, [47](#)
 - SemSolver::Equation, [58](#)
 - SemSolver::Function, [60](#)
 - SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > >, [122](#)
 - SemSolver::ScriptFunction< Point< 2, X >, Y >, [124](#)
 - SemSolver::ScriptFunction< Point< d, X >, Vector< Y > >, [126](#)
 - SemSolver::ScriptFunction< Point< d, X >, Y >, [128](#)
- mmls
 - SemSolver::BoundaryConditions< 2, X >, [37](#)
- modify
 - SemSolver::SegmentsMap, [134](#)
- modifyId
 - SemSolver::PointsBimap< 2, X >, [80](#)
- modifyPoint
 - SemSolver::PointsBimap< 2, X >, [80](#)
- MultiIndex
 - SemSolver::MultiIndex, [70](#)
- multiindex.hpp, [214](#)
- neighbour
 - SemSolver::Polygonation< 2, X >::Element, [52](#)
- neighboursBegin
 - SemSolver::Polygonation< 2, X >::Element, [52](#)
- neighboursEnd
 - SemSolver::Polygonation< 2, X >::Element, [52](#)
- NEUMANN
 - SemSolver::BoundaryConditions< 2, X >, [35](#)
- neumannData
 - SemSolver::BoundaryConditions< 2, X >, [37](#)
- next_non_empty_line_values
 - SemSolver::IO, [20](#)
- nextnonemptilinevalues.hpp, [216](#)
- Node
 - SemSolver::SemSpace< 2, X >::Node, [72](#)
- node
 - SemSolver::SemSpace< 2, X >, [149](#)
- NodeIterator
 - SemSolver::SemSpace< 2, X >, [146](#)
- NodesMap
 - SemSolver::SemSpace< 2, X >, [146](#)
- NodesVector
 - SemSolver::SemSpace< 2, X >, [147](#)

- NONE
 - SemSolver::Equation, [57](#)
- norm
 - SemSolver::HilbertSpace, [62](#)
- number
 - SemSolver::PSLG::Hole, [63](#)
 - SemSolver::PSLG::Segment, [129](#)
 - SemSolver::PSLG::Vertex, [152](#)
- omega
 - SemSolver::BilinearTransformation, [31](#)
- openRead
 - SemSolver::IO::Archive, [27](#)
- openWrite
 - SemSolver::IO::Archive, [27](#)
- operator*
 - SemSolver, [13](#)
 - SemSolver::Polynomial, [105](#)
- operator*=
 - SemSolver::Polynomial, [105](#)
- operator()
 - SemSolver::MultiIndex::less, [66](#)
 - SemSolver::Polynomial, [105](#)
 - SemSolver::Segment< 2, X >::less, [65](#)
- operator+
 - SemSolver, [13](#)
 - SemSolver::Matrix, [69](#)
 - SemSolver::Polynomial, [106](#)
- operator+=
 - SemSolver::Polynomial, [106](#)
- operator-
 - SemSolver::Polynomial, [106](#)
- operator-=
 - SemSolver::Polynomial, [107](#)
- operator/
 - SemSolver::Polynomial, [107](#)
- operator/=
 - SemSolver::Polynomial, [107](#)
- operator=
 - SemSolver::Polynomial, [107](#)
 - SemSolver::PolynomialFunction, [111](#)
- operator==
 - SemSolver::Polynomial, [108](#)
- operator%
 - SemSolver::Polynomial, [105](#)
- operator%=
 - SemSolver::Polynomial, [105](#)
- parameters
 - SemSolver::Problem, [114](#)
- parameters.hpp, [217](#)
- penalty
 - SemSolver::SemParameters, [142](#)
- Point
 - SemSolver::Point< 2, X >, [74](#)
- point
 - SemSolver::PointsBimap< 2, X >, [81](#)
 - SemSolver::SemSpace< 2, X >::Node, [72](#)
- point.hpp, [219](#)
- PointsBimap
 - SemSolver::PointsBimap< 2, X >, [77](#)
- pointsbimap.hpp, [220](#)
- PointsMap
 - SemSolver::PointsMap< 2, X, Y >, [84](#)
- pointsmmap.hpp, [225](#)
- PointsSet
 - SemSolver::PointsSet< 2, X >, [91](#)
- pointset.hpp, [230](#)
- Polygon
 - SemSolver::Polygon< 2, X >, [96](#)
- polygon.hpp, [234](#)
- polygonation.hpp, [236](#)
- polygonwithholes.hpp, [242](#)
- Polynomial
 - SemSolver::Polynomial, [103](#)
- polynomial
 - SemSolver::PolynomialFunction, [111](#)
 - SemSolver::SemFunction< 2, X >, [138](#)
- polynomial.hpp, [243](#)
- PolynomialFunction
 - SemSolver::PolynomialFunction, [110](#)
- polynomialfunction.hpp, [248](#)
- Problem
 - SemSolver::Problem, [113](#)
- problem.hpp, [250](#)
- projection
 - SemSolver::HilbertSpace, [62](#)
- PSLG
 - SemSolver::PSLG, [117](#)
- pslg.hpp, [253](#), [262](#)
- qr_solve
 - SemSolver::Solver, [24](#)
- qrsolve.hpp, [269](#)
- reaction

- SemSolver::DiffusionConvectionReactionEquation, 42
- SemSolver::DiffusionConvectionReactionEquation< 2, X >, 47
- read_boundary_conditions
 - SemSolver::IO, 20
- read_equation
 - SemSolver::IO, 20
- read_geometry
 - SemSolver::IO, 20
- read_parameters
 - SemSolver::IO, 20
- read_PSLG
 - SemSolver::IO, 20
- read_subdomains
 - SemSolver::IO, 20
- realEigenvalues
 - SemSolver::Matrix, 68
- refine
 - SemSolver::Polygonation< 2, X >, 99
- remove
 - SemSolver::SegmentsMap, 134
- remove_file_from_workspace
 - SemSolver::IO, 20
- ROBIN
 - SemSolver::BoundaryConditions< 2, X >, 35
- robinCoefficient
 - SemSolver::BoundaryConditions< 2, X >, 38
- robinData
 - SemSolver::BoundaryConditions< 2, X >, 38
- rows
 - SemSolver::Matrix, 68
 - SemSolver::Vector, 151
- ruffini
 - SemSolver::Polynomial, 108
- scalar
 - SemSolver, 14
- scalarProduct
 - SemSolver::HilbertSpace, 62
 - SemSolver::SemSpace< 2, X >, 149
- ScriptFunction
 - SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > >, 122
 - SemSolver::ScriptFunction< Point< 2, X >, Y >, 124
- SemSolver::ScriptFunction< Point< 2, X >, Vector< Y > >, 126
- SemSolver::ScriptFunction< Point< 2, X >, Y >, 127
- scriptfunction.hpp, 270
- Segment
 - SemSolver::Segment< 2, X >, 130
- segment
 - SemSolver::PSLG, 117
 - SemSolver::SegmentsMap, 134
- segment.hpp, 277
- segmentFrom
 - SemSolver::SegmentsMap, 135
- segments
 - SemSolver::PSLG, 118
 - SemSolver::SegmentsMap, 135
- SegmentsMap
 - SemSolver::SegmentsMap, 132
- segmentsmap.hpp, 279
- segmentTo
 - SemSolver::SegmentsMap, 135
- SemFunction
 - SemSolver::SemFunction< 2, X >, 137
- semfunction.hpp, 282
- SemFunctionsVector
 - SemSolver::SemSpace< 2, X >, 147
- semgeometry.hpp, 285
- SemParameters
 - SemSolver::SemParameters, 141
- semparameters.hpp, 286
- SemSolver, 9
 - operator*, 13
 - operator+, 13
 - scalar, 14
 - Sequence, 13
 - Sequences_list, 13
- SemSolver::Assembler, 14
 - compute_algebraic_system, 15
 - compute_border_matrix, 15
 - compute_border_vector, 15
 - compute_convection_matrix, 15
 - compute_diffusion_matrix, 16
 - compute_forcing_vector, 16
 - compute_reaction_matrix, 16
- SemSolver::BilinearTransformation, 27
 - ~BilinearTransformation, 30
- BilinearTransformation, 29
 - evaluate, 30
 - evaluateInverse, 30

- evaluateJacobianDeterminant, 30
- evaluateTransposeInverseJacobian, 31
- omega, 31
- setOmega, 31
- setTolerance, 31
- tolerance, 32
- SemSolver::BoundaryConditions, 32
- SemSolver::BoundaryConditions< 2, X >, 33
 - borderType, 36
 - BoundaryConditions, 35, 36
 - clear, 36
 - conditions, 36
 - DIRICHLET, 35
 - dirichletData, 37
 - FunctionConstIterator, 34
 - FunctionPtr, 34
 - FunctionsMap, 35
 - labels, 37
 - mmls, 37
 - NEUMANN, 35
 - neumannData, 37
 - ROBIN, 35
 - robinCoefficient, 38
 - robinData, 38
 - setBorder, 38
 - Type, 35
 - TypesMap, 35
 - UNDEFINED, 35
- SemSolver::DiffusionConvectionReactionEquation, 39
 - ~DiffusionConvectionReactionEquation, 41
 - convection, 41
 - diffusion, 41
 - DiffusionConvectionReactionEquation, 41
 - forcing, 42
 - mml, 42
 - reaction, 42
 - setConvection, 42
 - setDiffusion, 43
 - setForcing, 43
 - setReaction, 43
 - Type, 41
 - type, 43
- SemSolver::DiffusionConvectionReactionEquation< 2, X >, 44
 - ~DiffusionConvectionReactionEquation, 46
 - convection, 46
 - diffusion, 46
 - DiffusionConvectionReactionEquation, 46
 - forcing, 46
 - mml, 47
 - reaction, 47
 - setConvection, 47
 - setDiffusion, 47
 - setForcing, 48
 - setReaction, 48
 - Type, 45
 - type, 48
- SemSolver::Equation, 56
 - ~Equation, 57
 - DIFFUSION_CONVECTION_REACTION, 57
 - Equation, 57
 - mml, 58
 - NONE, 57
 - Type, 57
 - type, 58
- SemSolver::Function, 58
 - ~Function, 59
 - evaluate, 59
 - Function, 59
 - mml, 60
- SemSolver::HilbertSpace, 60
 - baseFunction, 61
 - dimension, 61
 - norm, 62
 - projection, 62
 - scalarProduct, 62
- SemSolver::HilbertSpace::Element, 48
 - ~Element, 49
 - Element, 49
- SemSolver::Homeomorphism, 63
 - ~Homeomorphism, 64
 - evaluateInverse, 65
 - Homeomorphism, 64
- SemSolver::IO, 16
 - add_file_to_workspace, 18
 - extract_file_from_workspace, 18
 - get_boundary_conditions_from_workspace, 19
 - get_boundary_conditions_list_from_workspace, 19
 - get_equation_from_workspace, 19
 - get_equations_list_from_workspace, 19

- get_geometries_list_from_workspace, 19
- get_geometry_from_workspace, 19
- get_parameters_from_workspace, 19
- get_parameters_list_from_workspace, 19
- next_non_empty_line_values, 20
- read_boundary_conditions, 20
- read_equation, 20
- read_geometry, 20
- read_parameters, 20
- read_PSLG, 20
- read_subdomains, 20
- remove_file_from_workspace, 20
- write_geometry, 21
- write_subdomains, 21
- SemSolver::IO::Archive, 25
 - addFile, 26
 - addValue, 26
 - Archive, 26
 - closeRead, 26
 - closeWrite, 26
 - entries, 27
 - extractFile, 27
 - openRead, 27
 - openWrite, 27
- SemSolver::Matrix, 66
 - columns, 68
 - Matrix, 67
 - operator+, 69
 - realEigenvalues, 68
 - rows, 68
- SemSolver::MultiIndex, 69
 - MultiIndex, 70
 - setSubIndex, 70
 - subIndex, 71
- SemSolver::MultiIndex::less, 66
 - operator(), 66
- SemSolver::Point< 2, X >, 73
 - Point, 74
- SemSolver::PointsBimap< 2, X >, 74
 - ConstIterator, 76
 - eraseId, 77
 - erasePoint, 77
 - findId, 77, 78
 - findPoint, 78
 - hasId, 79
 - hasPoint, 79
 - hasPointOn, 79
 - id, 79
 - insert, 80
 - insertPoint, 80
 - KeyType, 76
 - MappedType, 76
 - modifyId, 80
 - modifyPoint, 80
 - point, 81
 - PointsBimap, 77
 - SizeType, 76
 - ValueType, 76
- SemSolver::PointsMap< 2, X, Y >, 81
 - begin, 84
 - clear, 84
 - ConstIterator, 83
 - end, 85
 - erase, 85
 - find, 86
 - has, 86
 - insert, 86, 87
 - isEmpty, 88
 - Iterator, 83
 - KeyType, 83
 - MappedType, 83
 - PointsMap, 84
 - size, 88
 - SizeType, 84
 - ValueType, 84
- SemSolver::PointsSet< 2, X >, 89
 - begin, 92
 - clear, 92
 - ConstIterator, 91
 - end, 92
 - erase, 92, 93
 - find, 93
 - has, 93
 - insert, 94
 - isEmpty, 95
 - Iterator, 91
 - KeyType, 91
 - MappedType, 91
 - PointsSet, 91
 - size, 95
 - SizeType, 91
 - ValueType, 91
- SemSolver::Polygon< 2, X >, 96
 - contains, 97
 - Polygon, 96
- SemSolver::Polygonation< 2, X >, 97
 - addElement, 98
 - clear, 98

- element, [99](#)
 - elementIndicesAt, [99](#)
 - isQuadrangulation, [99](#)
 - refine, [99](#)
 - size, [100](#)
- SemSolver::Polygonation< 2, X >::Element, [49](#)
 - clear, [51](#)
 - contains, [51](#)
 - Element, [51](#)
 - geometry, [52](#)
 - neighbour, [52](#)
 - neighboursBegin, [52](#)
 - neighboursEnd, [52](#)
 - setGeometry, [52](#)
 - setNeighbour, [53](#)
 - size, [53](#)
 - vertex, [53](#)
 - vertexPosition, [53](#)
 - verticesBegin, [54](#)
 - verticesEnd, [54](#)
- SemSolver::PolygonWithHoles< 2, X >, [100](#)
 - cgal, [101](#)
- SemSolver::Polynomial, [101](#)
 - ~Polynomial, [104](#)
 - coefficient, [104](#)
 - degree, [104](#)
 - derivative, [104](#)
 - operator*, [105](#)
 - operator*=[105](#)
 - operator(), [105](#)
 - operator+, [106](#)
 - operator+=, [106](#)
 - operator-, [106](#)
 - operator-=, [107](#)
 - operator/, [107](#)
 - operator/=[107](#)
 - operator=, [107](#)
 - operator==, [108](#)
 - operator%, [105](#)
 - operator%=[105](#)
 - Polynomial, [103](#)
 - ruffini, [108](#)
 - setCoefficient, [108](#)
 - setDegree, [108](#)
 - zeros, [108](#)
- SemSolver::PolynomialFunction, [109](#)
 - ~PolynomialFunction, [110](#)
 - evaluate, [111](#)
 - operator=, [111](#)
 - polynomial, [111](#)
 - PolynomialFunction, [110](#)
 - setPolynomial, [111](#)
- SemSolver::PostProcessor, [21](#)
 - build_solution, [21](#)
 - compute_plot_data, [21](#)
 - compute_solution_hull, [21](#)
- SemSolver::PreProcessor, [22](#)
 - compute_polygon_with_holes_from_pslg, [22](#)
 - compute_polygonation_from_pslg, [22](#)
 - compute_vertices_sequences_from_pslg, [23](#)
- SemSolver::Problem, [112](#)
 - ~Problem, [113](#)
 - boundaryConditions, [113](#)
 - clearBoundaryConditions, [113](#)
 - clearEquation, [113](#)
 - clearGeometry, [114](#)
 - clearParameters, [114](#)
 - equation, [114](#)
 - geometry, [114](#)
 - isDefined, [114](#)
 - parameters, [114](#)
 - Problem, [113](#)
 - setBoundaryConditions, [114](#)
 - setEquation, [115](#)
 - setGeometry, [115](#)
 - setParameters, [115](#)
- SemSolver::PSLG, [115](#)
 - ~PSLG, [117](#)
 - clear, [117](#)
 - hole, [117](#)
 - holes, [117](#)
 - PSLG, [117](#)
 - segment, [117](#)
 - segments, [118](#)
 - setHole, [118](#)
 - setNumberOfHoles, [118](#)
 - setNumberOfSegments, [119](#)
 - setNumberOfVertices, [119](#)
 - setNumberOfVerticesAttributes, [119](#)
 - setNumberOfVerticesBoundaryMarkers, [119](#)
 - setSegment, [119](#)
 - setVertex, [120](#)
 - vertex, [120](#)
 - vertices, [120](#)
- SemSolver::PSLG::Hole, [63](#)

- number, [63](#)
 - x, [63](#)
 - y, [63](#)
- SemSolver::PSLG::Segment, [128](#)
 - marker, [129](#)
 - number, [129](#)
 - source, [129](#)
 - target, [129](#)
- SemSolver::PSLG::Vertex, [152](#)
 - attributes, [152](#)
 - marker, [152](#)
 - number, [152](#)
 - x, [152](#)
 - y, [153](#)
- SemSolver::ScriptFunction< Point< 2, X
>, Vector< Y > >, [121](#)
 - ~ScriptFunction, [122](#)
 - evaluate, [122](#)
 - mml, [122](#)
 - ScriptFunction, [122](#)
- SemSolver::ScriptFunction< Point< 2, X
>, Y >, [123](#)
 - ~ScriptFunction, [124](#)
 - evaluate, [124](#)
 - mml, [124](#)
 - ScriptFunction, [124](#)
- SemSolver::ScriptFunction< Point< d, X
>, Vector< Y > >, [125](#)
 - ~ScriptFunction, [126](#)
 - evaluate, [126](#)
 - mml, [126](#)
 - ScriptFunction, [126](#)
- SemSolver::ScriptFunction< Point< d, X
>, Y >, [127](#)
 - ~ScriptFunction, [128](#)
 - evaluate, [128](#)
 - mml, [128](#)
 - ScriptFunction, [127](#)
- SemSolver::Segment< 2, X >, [129](#)
 - cgal, [130](#)
 - intersect, [130](#)
 - intersectInteriorly, [130](#)
 - Segment, [130](#)
- SemSolver::Segment< 2, X >::less, [65](#)
 - operator(), [65](#)
- SemSolver::SegmentsMap, [131](#)
 - add, [133](#)
 - ConstIterator, [132](#)
 - contains, [133](#)
 - has, [133](#)
 - haveOn, [133](#)
 - id, [133](#)
 - insert, [133](#)
 - intersect, [134](#)
 - intersectInteriorly, [134](#)
 - isConsistentWith, [134](#)
 - Iterator, [132](#)
 - modify, [134](#)
 - remove, [134](#)
 - segment, [134](#)
 - segmentFrom, [135](#)
 - segments, [135](#)
 - SegmentsMap, [132](#)
 - segmentTo, [135](#)
- SemSolver::SemFunction, [135](#)
- SemSolver::SemFunction< 2, X >, [136](#)
 - evaluate, [137](#)
 - evaluateRestrictionGradient, [137](#)
 - map, [138](#)
 - polynomial, [138](#)
 - SemFunction, [137](#)
 - setPolynomialComponent, [138](#)
- SemSolver::SemGeometry, [138](#)
- SemSolver::SemGeometry< 2, X >, [139](#)
 - contains, [140](#)
 - domain, [140](#)
 - setDomain, [140](#)
 - setSubDomains, [140](#)
 - subDomains, [140](#)
- SemSolver::SemParameters, [140](#)
 - degree, [142](#)
 - penalty, [142](#)
 - SemParameters, [141](#)
 - setDegree, [142](#)
 - setPenalty, [142](#)
 - setTolerance, [142](#)
 - tolerance, [142](#)
- SemSolver::SemSpace, [143](#)
- SemSolver::SemSpace< 2, X >, [143](#)
 - _geometry, [150](#)
 - _parameters, [150](#)
 - baseFunction, [147](#)
 - border, [147](#)
 - borderId, [148](#)
 - borderIndex, [148](#)
 - borderNode, [148](#)
 - borders, [148](#)
 - BordersMap, [146](#)
 - borderSubDomainIndex, [148](#)
 - BordersVector, [146](#)

- borderWeight, 148
- degree, 148
- ElementConstIterator, 146
- ElementsMap, 146
- Index2Order, 146
- Index3Order, 146
- node, 149
- NodeConstIterator, 146
- nodes, 149
- NodesMap, 146
- NodesVector, 147
- scalarProduct, 149
- SemFunctionsVector, 147
- SemSpace, 147
- SubDomain, 147
- subDomainIndex, 149
- subDomainNode, 149
- subDomains, 149
- subDomainWeight, 149
- WeightsMap, 147
- SemSolver::SemSpace< 2, X >::Element, 54
 - ~Element, 55
 - Element, 55
 - evaluate, 55
- SemSolver::SemSpace< 2, X >::Node, 71
 - borderIndex, 72
 - Node, 72
 - point, 72
 - SemSpace, 73
 - subDomainIndex, 72
 - supportBorders, 72
 - supportSubDomains, 73
- SemSolver::Solver, 23
 - cholesky_solve, 23
 - lu_solve, 24
 - qr_solve, 24
- SemSolver::Vector, 150
 - rows, 151
 - Vector, 151
- SemSpace
 - SemSolver::SemSpace< 2, X >, 147
 - SemSolver::SemSpace< 2, X >::Node, 73
- semSPACE.hpp, 287
- Sequence
 - SemSolver, 13
- sequence.hpp, 300
- Sequences_list
 - SemSolver, 13
- sequenceslist.hpp, 300
- setBorder
 - SemSolver::BoundaryConditions< 2, X >, 38
- setBoundaryConditions
 - SemSolver::Problem, 114
- setCoefficient
 - SemSolver::Polynomial, 108
- setConvection
 - SemSolver::DiffusionConvectionReactionEquation, 42
 - SemSolver::DiffusionConvectionReactionEquation< 2, X >, 47
- setDegree
 - SemSolver::Polynomial, 108
 - SemSolver::SemParameters, 142
- setDiffusion
 - SemSolver::DiffusionConvectionReactionEquation, 43
 - SemSolver::DiffusionConvectionReactionEquation< 2, X >, 47
- setDomain
 - SemSolver::SemGeometry< 2, X >, 140
- setEquation
 - SemSolver::Problem, 115
- setForcing
 - SemSolver::DiffusionConvectionReactionEquation, 43
 - SemSolver::DiffusionConvectionReactionEquation< 2, X >, 48
- setGeometry
 - SemSolver::Polygonation< 2, X >::Element, 52
 - SemSolver::Problem, 115
- setHole
 - SemSolver::PSLG, 118
- setNeighbour
 - SemSolver::Polygonation< 2, X >::Element, 53
- setNumberOfHoles
 - SemSolver::PSLG, 118
- setNumberOfSegments
 - SemSolver::PSLG, 119
- setNumberOfVertices
 - SemSolver::PSLG, 119
- setNumberOfVerticesAttributes
 - SemSolver::PSLG, 119
- setNumberOfVerticesBoundaryMarkers
 - SemSolver::PSLG, 119

- setOmega
 - SemSolver::BilinearTransformation, 31
- setParameter
 - SemSolver::Problem, 115
- setPenalty
 - SemSolver::SemParameters, 142
- setPolynomial
 - SemSolver::PolynomialFunction, 111
- setPolynomialComponent
 - SemSolver::SemFunction< 2, X >, 138
- setReaction
 - SemSolver::DiffusionConvectionReactionEquation, 43
 - SemSolver::DiffusionConvectionReactionEquation< 2, X >, 48
- setSegment
 - SemSolver::PSLG, 119
- setSubDomains
 - SemSolver::SemGeometry< 2, X >, 140
- setSubIndex
 - SemSolver::MultiIndex, 70
- setTolerance
 - SemSolver::BilinearTransformation, 31
 - SemSolver::SemParameters, 142
- setVertex
 - SemSolver::PSLG, 120
- size
 - SemSolver::PointsMap< 2, X, Y >, 88
 - SemSolver::PointsSet< 2, X >, 95
 - SemSolver::Polygonation< 2, X >, 100
 - SemSolver::Polygonation< 2, X >::Element, 53
- SizeType
 - SemSolver::PointsBimap< 2, X >, 76
 - SemSolver::PointsMap< 2, X, Y >, 84
 - SemSolver::PointsSet< 2, X >, 91
- source
 - SemSolver::PSLG::Segment, 129
- SubDomain
 - SemSolver::SemSpace< 2, X >, 147
- subDomainIndex
 - SemSolver::SemSpace< 2, X >, 149
 - SemSolver::SemSpace< 2, X >::Node, 72
- subDomainNode
 - SemSolver::SemSpace< 2, X >, 149
- subDomains
 - SemSolver::SemGeometry< 2, X >, 140
 - SemSolver::SemSpace< 2, X >, 149
- subdomains.hpp, 301
- subDomainWeight
 - SemSolver::SemSpace< 2, X >, 149
- subIndex
 - SemSolver::MultiIndex, 71
- supportBorders
 - SemSolver::SemSpace< 2, X >::Node, 72
- supportSubDomains
 - SemSolver::SemSpace< 2, X >::Node, 73
- target
 - SemSolver::PSLG::Segment, 129
- tolerance
 - SemSolver::BilinearTransformation, 32
 - SemSolver::SemParameters, 142
- Type
 - SemSolver::BoundaryConditions< 2, X >, 35
 - SemSolver::DiffusionConvectionReactionEquation, 41
 - SemSolver::DiffusionConvectionReactionEquation< 2, X >, 45
 - SemSolver::Equation, 57
 - SemSolver::DiffusionConvectionReactionEquation, 43
 - SemSolver::DiffusionConvectionReactionEquation< 2, X >, 48
 - SemSolver::Equation, 58
- TypesMap
 - SemSolver::BoundaryConditions< 2, X >, 35
- UNDEFINED
- Value
 - SemSolver::BoundaryConditions< 2, X >, 35
- ValueType
 - SemSolver::PointsBimap< 2, X >, 76
 - SemSolver::PointsMap< 2, X, Y >, 84
 - SemSolver::PointsSet< 2, X >, 91
- Vector

- SemSolver::Vector, [151](#)
- vector.hpp, [303](#)
- vertex
 - SemSolver::Polygonation< 2, X >::Element, [53](#)
 - SemSolver::PSLG, [120](#)
- vertexPosition
 - SemSolver::Polygonation< 2, X >::Element, [53](#)
- vertices
 - SemSolver::PSLG, [120](#)
- verticesBegin
 - SemSolver::Polygonation< 2, X >::Element, [54](#)
- verticesEnd
 - SemSolver::Polygonation< 2, X >::Element, [54](#)
- WeightsMap
 - SemSolver::SemSpace< 2, X >, [147](#)
- workspace.hpp, [305](#)
- write_geometry
 - SemSolver::IO, [21](#)
- write_subdomains
 - SemSolver::IO, [21](#)
- x
 - SemSolver::PSLG::Hole, [63](#)
 - SemSolver::PSLG::Vertex, [152](#)
- y
 - SemSolver::PSLG::Hole, [63](#)
 - SemSolver::PSLG::Vertex, [153](#)
- zeros
 - SemSolver::Polynomial, [108](#)