

# An Object Oriented Finite Element Library

Reference Guide

Rachid Touzani Laboratoire de Mathématiques Blaise Pascal Université Clermont Auvergne, France e-mail: rachid.touzani@uca.fr

# **Contents**

1	Module Index 1.1 Modules	. 3
2	Namespace Index 2.1 Namespace List	. 5
3	Hierarchical Index 3.1 Class Hierarchy	. 7
4	Class Index 4.1 Class List	<b>11</b> . 11
5	Module Documentation	17
	5.1 General Purpose Equations	
	5.2 Conservation Law Equations	
	5.3 Electromagnetics	
	5.4 Fluid Dynamics	. 25
	5.5 Interface Problems	
	5.6 Laplace equation	. 27
	5.7 Porous Media problems	
	5.8 Solid Mechanics	. 29
	5.9 Heat Transfer	. 30
	5.10 Input/Output	. 31
	5.11 Utilities	. 32
	5.12 Vector and Matrix	. 69
	5.13 Physical properties of media	
	5.14 Global Variables	
	5.15 Finite Element Mesh	
	5.16 Shape Function	
	5.17 Solver	
	5.18 OFELI	
6	Namespace Documentation	173
	6.1 OFELI Namespace Reference	. 173
7	Class Documentation	197
	7.1 AbsEqua < T_ > Class Template Reference	
	7.2 Bar2DL2 Class Reference	
	7.3 Beam3DL2 Class Reference	
	7.4 BiotSavart Class Reference	
	7.5 BMatrix < T_ > Class Template Reference	
	7.6 Brick Class Reference	. 216
	7.7 Circle Class Reference	

CONTENTS

7.8	DC1DL2 Class Reference	
7.9	DC2DT3 Class Reference	223
7.10	DC2DT6 Class Reference	228
7.11	DC3DAT3 Class Reference	232
7.12	DC3DT4 Class Reference	235
	DG Class Reference	
	DMatrix $<$ T $>$ Class Template Reference	
	Domain Class Reference	
	DSMatrix< T <sub>-</sub> > Class Template Reference	
	EC2D1T3 Class Reference	
	EC2D2T3 Class Reference	
	Edge Class Reference	
	EdgeList Class Reference	
	EigenProblemSolver Class Reference	
	Elas2DQ4 Class Reference	
	Elas2DT3 Class Reference	
	Elas3DH8 Class Reference	
	Elas3DT4 Class Reference	
	Element Class Reference	
	ElementList Class Reference	
	Ellipse Class Reference	
	Equa_Electromagnetics< T_, NEN_, NEE_, NSN_, NSE_ > Class Template Reference	
	Equa_Fluid< T_, NEN_, NEE_, NSN_, NSE_ > Class Template Reference	
	Equa_Laplace< T_, NEN_, NEE_, NSN_, NSE_ > Class Template Reference	
	$Equa\_Porous < T, NEN, NEE, NSN, NSE > Class\ Template\ Reference \ . \ . \ . \ .$	
	Equa_Solid < T_, NEN_, NEE_, NSN_, NSE_ > Class Template Reference	
	Equa_Therm< T_, NEN_, NEE_, NSN_, NSE_ > Class Template Reference	
	Equation $<$ T <sub>-</sub> , NEN <sub>-</sub> , NEE <sub>-</sub> , NSN <sub>-</sub> , NSE <sub>-</sub> $>$ Class Template Reference	
7.36	Estimator Class Reference	329
7.36 7.37	Estimator Class Reference	329 331
7.36 7.37 7.38	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference	329 331 333
7.36 7.37 7.38 7.39	Estimator Class Reference	329 331 333 335
7.36 7.37 7.38 7.39 7.40	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference	329 331 333 335 337
7.36 7.37 7.38 7.39 7.40 7.41	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference	329 331 333 335 337 339
7.36 7.37 7.38 7.39 7.40 7.41	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference	329 331 333 335 337 339
7.36 7.37 7.38 7.39 7.40 7.41 7.42	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference	329 331 333 335 337 339 341
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference FMM5D Class Reference FMMSolver Class Reference	329 331 333 335 337 339 341 342
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43 7.44	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference FMMSolver Class Reference Funct Class Reference	329 331 333 335 337 339 341 342
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43 7.44 7.45	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference FMMSolver Class Reference Funct Class Reference Gauss Class Reference	329 331 333 335 337 339 341 342 344
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43 7.44 7.45 7.46	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference FMMSolver Class Reference Funct Class Reference Gauss Class Reference Grid Class Reference	329 331 333 335 337 339 341 342 344 345
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43 7.44 7.45 7.46 7.47	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference FMM5olver Class Reference Funct Class Reference Gauss Class Reference Gauss Class Reference HelmholtzBT3 Class Reference	329 331 333 335 337 341 342 344 345 348
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43 7.44 7.45 7.46 7.47 7.48	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference FMMSolver Class Reference Funct Class Reference Gauss Class Reference Huct Class Reference Grid Class Reference HelmholtzBT3 Class Reference Hexa8 Class Reference	329 331 333 335 337 339 341 342 344 345 348
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43 7.44 7.45 7.46 7.47 7.48 7.49	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference FMMSolver Class Reference Funct Class Reference Gauss Class Reference Grid Class Reference HelmholtzBT3 Class Reference HelmholtzBT3 Class Reference ICPG1D Class Reference ICPG2DT Class Reference	329 331 333 335 337 339 341 342 344 345 348 349 351
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43 7.44 7.45 7.46 7.47 7.48 7.49 7.50	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference FMMSolver Class Reference Funct Class Reference Gauss Class Reference Gauss Class Reference Grid Class Reference HelmholtzBT3 Class Reference Heza8 Class Reference ICPG1D Class Reference ICPG2DT Class Reference ICPG3DT Class Reference ICPG3DT Class Reference	329 331 333 335 337 341 342 344 345 348 351 356
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43 7.44 7.45 7.46 7.47 7.48 7.50 7.51	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference FMMSolver Class Reference Funct Class Reference Gauss Class Reference Grid Class Reference HelmholtzBT3 Class Reference HelmholtzBT3 Class Reference ICPG1D Class Reference ICPG2DT Class Reference ICPG3DT Class Reference ICPG3DT Class Reference Integration Class Reference	329 331 333 335 337 339 341 342 344 345 351 356 361 363
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43 7.44 7.45 7.46 7.47 7.48 7.50 7.51 7.52	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference Fimure Class Reference FMM2D Class Reference FMM3D Class Reference FMMSolver Class Reference Funct Class Reference Gauss Class Reference Grid Class Reference HelmholtzBT3 Class Reference HelmholtzBT3 Class Reference ICPG1D Class Reference ICPG2DT Class Reference ICPG3DT Class Reference ICPG3DT Class Reference Integration Class Reference Integration Class Reference	329 331 333 335 337 339 341 342 344 345 348 351 356 361 363 363
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43 7.44 7.45 7.46 7.47 7.48 7.50 7.51 7.52 7.53	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference FMMSolver Class Reference Funct Class Reference Gauss Class Reference Grid Class Reference Grid Class Reference HelmholtzBT3 Class Reference Heza8 Class Reference ICPG1D Class Reference ICPG2DT Class Reference ICPG3DT Class Reference	329 331 333 335 337 341 342 344 345 349 351 363 363 363
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43 7.44 7.45 7.46 7.47 7.48 7.50 7.51 7.52 7.53 7.54	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference FMMSolver Class Reference FMMSolver Class Reference Funct Class Reference Gauss Class Reference Grid Class Reference HelmholtzBT3 Class Reference HelmholtzBT3 Class Reference ICPG1D Class Reference ICPG2DT Class Reference ICPG3DT Class Reference ICPG3DT Class Reference Integration CLASS REFERENCE INTE	329 331 333 335 337 341 342 344 345 351 356 361 363 367 377
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43 7.44 7.45 7.46 7.47 7.48 7.50 7.51 7.52 7.53 7.54 7.55	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference FMMSolver Class Reference Funct Class Reference Gauss Class Reference Grid Class Reference Grid Class Reference HelmholtzBT3 Class Reference HelmholtzBT3 Class Reference ICPG1D Class Reference ICPG2DT Class Reference ICPG3DT Class Reference Integration Class Reference Integration Class Reference ITEF Class Reference ITEF Class Reference ITEF Class Reference Iter< T_ > Class Template Reference Laplace1DL2 Class Reference	329 331 333 335 337 341 342 344 345 351 356 361 363 367 377 378
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43 7.44 7.45 7.46 7.47 7.48 7.50 7.51 7.52 7.53 7.54 7.55 7.56	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference FMMSolver Class Reference Funct Class Reference Gauss Class Reference Grid Class Reference Grid Class Reference HelmholtzBT3 Class Reference HelmholtzBT3 Class Reference ICPG1D Class Reference ICPG2DT Class Reference ICPG3DT Class Reference ICPG3DT Class Reference ICPG3DT Class Reference ICPG3DT Class Reference Integration Class Reference Integration Class Reference INTEGRATOR Class Reference INTEGRATOR CLASS Reference INTEGRATOR CLASS Reference Integration Class Reference IDFiclass Reference IDFiclass Reference IDFI Class Reference ILET	329 331 333 335 337 341 342 344 345 351 356 361 363 366 367 377 378 380
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43 7.44 7.45 7.46 7.47 7.48 7.50 7.51 7.52 7.53 7.54 7.55 7.56 7.57	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference FMMSolver Class Reference Funct Class Reference Gauss Class Reference Grid Class Reference Grid Class Reference HelmholtzBT3 Class Reference Hexa8 Class Reference ICPG1D Class Reference ICPG2DT Class Reference ICPG3DT Class Reference ICPG3DT Class Reference ICPG3DT Class Reference Integration CLASS RE	329 331 333 335 341 342 344 345 351 356 361 363 367 377 378 380 382
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43 7.44 7.45 7.46 7.47 7.50 7.51 7.52 7.53 7.54 7.55 7.56 7.57 7.58	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference FMMSolver Class Reference Funct Class Reference Gauss Class Reference Grid Class Reference Grid Class Reference HelmholtzBT3 Class Reference HelmholtzBT3 Class Reference ICPG1D Class Reference ICPG2DT Class Reference ICPG3DT Class Reference ICPG3DT Class Reference ICPG3DT Class Reference Integration C	329 331 333 335 341 342 344 345 351 356 361 363 367 377 378 380 382 385
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43 7.44 7.45 7.46 7.47 7.50 7.51 7.52 7.53 7.54 7.55 7.56 7.57 7.58 7.59	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference FMM5olver Class Reference FMMSolver Class Reference Funct Class Reference Gauss Class Reference Grid Class Reference Grid Class Reference HelmholtzBT3 Class Reference Hexa8 Class Reference ICPG1D Class Reference ICPG2DT Class Reference ICPG2DT Class Reference ICPG3DT Class Reference ICPG3DT Class Reference ICPG3DT Class Reference Iter <a href="mailto:top://www.com/superscripts/">top://www.com/superscripts/</a> Iter< Class Reference Integration Class Reference IFF Class Reference Iter< T_ > Class Template Reference Laplace1DL2 Class Reference Laplace1DL3 Class Reference Laplace2DT3 Class Reference Laplace2DT6 Class Reference Laplace2DT6 Class Reference LaplaceDG2DP1 Class Reference	329 331 333 335 337 341 342 344 345 351 363 363 367 377 378 380 382 385 387
7.36 7.37 7.38 7.39 7.40 7.41 7.42 7.43 7.44 7.45 7.46 7.51 7.52 7.53 7.54 7.55 7.56 7.57 7.58 7.59 7.60	Estimator Class Reference FastMarching2D Class Reference FEShape Class Reference Figure Class Reference FMM2D Class Reference FMM3D Class Reference FMMSolver Class Reference Funct Class Reference Gauss Class Reference Grid Class Reference Grid Class Reference HelmholtzBT3 Class Reference HelmholtzBT3 Class Reference ICPG1D Class Reference ICPG2DT Class Reference ICPG3DT Class Reference ICPG3DT Class Reference ICPG3DT Class Reference Integration C	329 331 333 335 341 342 344 345 351 356 361 363 367 377 378 380 382 385

CONTENTS

7.62 LCL3DT Class Reference	
7.63 Line2 Class Reference	
7.64 Line3 Class Reference	
7.65 LinearSolver $< T_{-} > Class$ Template Reference	02
7.66 LocalMatrix < T_, NR_, NC_ > Class Template Reference	.07
7.67 LocalVect $<$ T $$ , N $>$ Class Template Reference	
7.68 Material Class Reference	19
7.69 Matrix $< T_{-} > $ Class Template Reference	22
7.70 Mesh Class Reference	31
7.71 MeshAdapt Class Reference	51
7.72 Muscl Class Reference	56
7.73 Muscl1D Class Reference	60
7.74 Muscl2DT Class Reference	
7.75 Muscl3DT Class Reference	63
7.76 MyNLAS Class Reference	64
7.77 MyOpt Class Reference	66
7.78 NLASSolver Class Reference	:00
7.79 Node Class Reference	
7.80 NodeList Class Reference	://
7.81 NSP2DQ41 Class Reference	:79
7.82 ODESolver Class Reference	
7.83 OFELIException Class Reference	
7.84 OptSolver Class Reference	
7.85 Partition Class Reference	
7.86 Penta6 Class Reference	
7.87 PhaseChange Class Reference	
7.88 Point $<$ T $>$ Class Template Reference	
7.89 Point2D $<$ T $>$ Class Template Reference	
7.90 Polygon Class Reference	
7.91 $Prec < T > Class Template Reference$	
7.92 Prescription Class Reference	18
7.93 Quad4 Class Reference	19
7.94 Reconstruction Class Reference	521
7.95 Rectangle Class Reference	523
7.96 Side Class Reference	
7.97 SideList Class Reference	31
7.98 SkMatrix< T <sub>-</sub> > Class Template Reference	
	542
•	550
	552
	555
	557
	558
	60
	61
	68
	570
	72
0	74
0	76
0	78
r	79
r	81
7.115WaterPorous2D Class Reference	510

CONTENTS

# **Module Index**

# 1.1 Modules

Here is a list of all modules:	100
OFELI	126
General Purpose Equations	17
Conservation Law Equations	<b>2</b> 3
Electromagnetics	24
Fluid Dynamics	<b>2</b> 5
Interface Problems	26
Laplace equation	27
Porous Media problems	28
Solid Mechanics	29
Heat Transfer	30
Input/Output	31
Utilities	32
Physical properties of media	81
Global Variables	82
Finite Element Mesh	85
Shape Function	107
Solver	108
Vector and Matrix	69

# Namespace Index

2.1	Namespace List	
	s a list of all documented namespaces with brief descriptions:	
	A namespace to group all library classes, functions,	173

# **Hierarchical Index**

# 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:	
AbsEqua $<$ T $_{-}$ >	
$Equation < T, NEN, NEE, NSN, NSE > \dots $	
	05
1 ' ' ' '	06
	07
	10
	12
	14
$AbsEqua < complex_t > \dots $	
	17
	05
	66
	48
AbsEqua < double >	97
Equation < double, NEN_, NEE_, NSN_, NSE_ >	17
Equa_Solid < double, 8, 24, 4, 12 >	12
Elas3DH8	88
AbsEqua < real_t >	97
Equation < real_t, 3, 3, 2, 2 >	17
	39
	87
1	17
•	05
1	69
	06
	68
	06
1	70
	06
	<del>7</del> 9
	07
	78
1	07
1	

1	380
Equa_Laplace< real_t, 3, 3, 2, 2 >	307
	382
*	307
* *	307
	385
	310
1	310
	610
1	312
	203
Equa_Solid < real_t, 2, 4, 1, 2 >	312
Bar2DL2	200
Equa_Solid < real_t, 3, 6, 2, 4 >	312
1	284
	312
	290
1	312
	280
1	314
	219
Equa_Therm< real_t, 3, 3, 2, 2 >	314
DC2DT3	223
DC3DAT3	232
Equa_Therm< real_t, 4, 4, 3, 3 >	314
<u>.</u>	
DC3DT4	235
	235 314
Equa_Therm< real_t, 6, 6, 3, 3 >	314
Equa_Therm< real_t, 6, 6, 3, 3 >	314 228
Equa_Therm<	314 228 555
Equa_Therm       real_t, 6, 6, 3, 3 >	314 228 555 207
Equa_Therm       real_t, 6, 6, 3, 3 >         DC2DT6          SteklovPoincare2DBE          BiotSavart          Domain	314 228 555 207 253
Equa_Therm< real_t, 6, 6, 3, 3 >  DC2DT6  SteklovPoincare2DBE  SiotSavart  Domain  Edge	314 228 555 207 253 271
Equa_Therm< real_t, 6, 6, 3, 3 >  DC2DT6  SteklovPoincare2DBE  BiotSavart  Domain  Edge  Edge	314 228 555 207 253 271 273
Equa_Therm< real_t, 6, 6, 3, 3 > DC2DT6 SteklovPoincare2DBE  BiotSavart Domain Edge EdgeList EigenProblemSolver	314 228 555 207 253 271 273 275
Equa_Therm< real_t, 6, 6, 3, 3 >  DC2DT6  SteklovPoincare2DBE  BiotSavart  Domain  Edge  EdgeList  EigenProblemSolver  Element	314 228 555 207 253 271 273 275 292
Equa_Therm< real_t, 6, 6, 3, 3 >  DC2DT6  SteklovPoincare2DBE  BiotSavart  Domain  Edge  EdgeList  EigenProblemSolver  Element	314 228 555 207 253 271 273 275
Equa_Therm< real_t, 6, 6, 3, 3 >  DC2DT6 SteklovPoincare2DBE  BiotSavart Domain Edge EdgeList EigenProblemSolver Element Element ElementList Estimator	314 228 555 207 253 271 273 275 292
Equa_Therm< real_t, 6, 6, 3, 3 >  DC2DT6 SteklovPoincare2DBE  BiotSavart Domain Edge EdgeList EigenProblemSolver Element ElementList Estimator	314 228 555 207 253 271 273 275 292 302
Equa_Therm< real_t, 6, 6, 3, 3 > DC2DT6 SteklovPoincare2DBE  BiotSavart Domain Edge EdgeList EigenProblemSolver Element Element Estimator EastMarching2D	314 228 555 207 253 271 273 275 292 302 329
Equa_Therm< real_t, 6, 6, 3, 3 > DC2DT6 SteklovPoincare2DBE  BiotSavart Domain Edge EdgeList EigenProblemSolver Element Element ElementList Estimator EastMarching2D EEShape	314 228 555 207 253 271 273 275 292 302 329 331
Equa_Therm< real_t, 6, 6, 3, 3 > DC2DT6 SteklovPoincare2DBE  BiotSavart Domain Edge EdgeList EigenProblemSolver Element Element ElementList Estimator FastMarching2D EShape Hexa8	314 228 555 207 253 271 273 275 292 302 329 331 333
Equa_Therm< real_t, 6, 6, 3, 3 >	314 228 555 207 253 271 273 275 292 302 329 331 333 349
Equa_Therm< real_t, 6, 6, 3, 3 > DC2DT6 SteklovPoincare2DBE BiotSavart Domain Edge EdgeList EigenProblemSolver Element Element Estimator EastMarching2D EShape Hexa8 Line2 Line3	314 228 555 207 253 271 273 275 292 302 331 333 349 398
Equa_Therm< real_t, 6, 6, 3, 3 >  DC2DT6 SteklovPoincare2DBE BiotSavart Domain Edge EdgeList EigenProblemSolver Element Element Estimator EastMarching2D EEShape Hexa8 Line2 Line3 Penta6	314 228 555 207 253 271 273 275 292 302 329 331 333 349 398 400
Equa_Therm< real_t, 6, 6, 3, 3 >  DC2DT6 SteklovPoincare2DBE BiotSavart Domain Edge EdgeList EigenProblemSolver Element Element Element Element ElementList Estimator EastMarching2D EEShape Hexa8 Line2 Line3 Penta6 Quad4	314 228 555 207 253 271 273 275 292 302 329 331 333 349 398 400 503
Equa.Therm< real.t, 6, 6, 3, 3 > DC2DT6 SteklovPoincare2DBE BiotSavart Domain Edge EdgeList EigenProblemSolver Element Element ElementList Estimator FastMarching2D EEShape Hexa8 Line2 Line3 Penta6 Quad4 Tetra4	314 228 555 207 253 271 273 275 292 302 331 333 349 398 400 503 519
Equa.Therm< real_t, 6, 6, 3, 3 > DC2DT6 SteklovPoincare2DBE BiotSavart Domain Edge EdgeList EigenProblemSolver Element ElementList Estimator FastMarching2D EEShape Hexa8 Line2 Line3 Penta6 Quad4 Tetra4 triangle	314 228 555 207 253 271 273 275 292 302 329 331 333 349 400 503 519 558 576
Equa_Therm< real_t, 6, 6, 3, 3 > DC2DT6 SteklovPoincare2DBE BiotSavart Domain Edge EdgeList EigenProblemSolver Element ElementList Estimator FastMarching2D EEShape Hexa8 Line2 Line3 Penta6 Quad4 Tetra4 triangle Triang3	314 228 555 207 253 271 273 275 292 302 329 331 333 349 400 503 519 558 576 572
Equa_Therm< real_t, 6, 6, 3, 3 > DC2DT6  SteklovPoincare2DBE  BiotSavart  Domain  Edge  EdgeList  EigenProblemSolver  Element  ElementList  Estimator  FastMarching2D  EShape  Hexa8  Line2  Line3  Penta6  Quad4  Tetra4  triangle  Triang3  Triang6S	314 228 555 207 253 271 273 275 292 302 331 333 349 398 400 503 519 558 576 572
Equa.Therm< real_t, 6, 6, 3, 3 > DC2DT6  SteklovPoincare2DBE  SiotSavart  Domain  ddge  ddgeList  SigenProblemSolver  Slement  SlementList  Stimator  SastMarching2D  EShape  Hexa8  Line2  Line3  Penta6  Quad4  Tetra4  triangle  Triang3  Triang6S  Sigure	314 228 555 207 253 271 273 275 292 302 329 331 333 349 400 503 519 558 576 572 574 335
Equa_Therm< real_t, 6, 6, 3, 3 > DC2DT6  SteklovPoincare2DBE  SiotSavart  Domain  Edge EdgeList EigenProblemSolver Element ElementList Estimator FastMarching2D EShape Hexa8 Line2 Line3 Penta6 Quad4 Tetra4 triangle Triang3 Triang6S Figure Brick	314 228 555 207 253 271 273 275 292 302 331 333 349 398 400 503 519 558 576 572

Ellipse       303         Polygon       512
Rectangle
Sphere
Triangle
FMM2D
FMM3D
FMMSolver
Funct
Gauss
Grid
Integration
IOField
IPF
$Iter < T_{-} > \dots \qquad 377$
Iter< real.t >
Linear Solver $< T > \dots $ 402
LinearSolver < real_t >
LocalMatrix < T., NR., NC. >
LocalMatrix < real_t, 2, 2 >
LocalVect< T., N.>
Local Vect < OFELI::Point < real_t >, 3 >
LocalVect< size_t, 3 >
$\begin{aligned} & \text{Material} & \dots & $
BMatrix < T_ >
DMatrix $<$ T $>$
DSMatrix $\langle T_{-} \rangle$
SkMatrix $<$ T $>$
SkSMatrix< T_>
SpMatrix< T_>
$TrMatrix < T > \dots 579$
Matrix < double >
Matrix < real_t >
BMatrix< real_t >
DMatrix < real_t >
DSMatrix < real_t >
SpMatrix< real_t >
Mesh
MeshAdapt
Muscl
Muscl1D
ICPG1D
LCL1D
Muscl2DT
ICPG2DT
LCL2DT 392
Muscl3DT
ICPG3DT
LCL3DT 395

### CHAPTER 3. HIERARCHICAL INDEX

# **Class Index**

# 4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:	
AbsEqua $<$ T $_{-}>$	
Mother abstract class to describe equation	197
Bar2DL2	
To build element equations for Planar Elastic Bar element with 2 DOF (Degrees of Freedom) per node	200
Beam3DL2	
To build element equations for 3-D beam equations using 2-node lines	<b>2</b> 03
BiotSavart	
Class to compute the magnetic induction from the current density using the Biot-Savart formula	207
BMatrix< T_>	
To handle band matrices	212
Brick	
To store and treat a brick (parallelepiped) figure	216
Circle	
To store and treat a circular figure	218
DC1DL2	
Builds finite element arrays for thermal diffusion and convection in 1-D using 2-Node elements	219
DC2DT3	
Builds finite element arrays for thermal diffusion and convection in 2-D domains using 3-Node triangles	<b>22</b> 3
DC2DT6	
Builds finite element arrays for thermal diffusion and convection in 2-D domains using 6-Node triangles	228
DC3DAT3	
Builds finite element arrays for thermal diffusion and convection in 3-D domains with axisymmetry using 3-Node triangles	232
DC3DT4	
Builds finite element arrays for thermal diffusion and convection in 3-D domains using 4-Node tetrahedra	235
DG	
Enables preliminary operations and utilities for the Discontinous Galerkin method 239	od

DMatrix < T_ > To handle dense matrices	240
Domain	
To store and treat finite element geometric information $\dots$ DSMatrix $<$ $T>$	253
To handle symmetric dense matrices	258
Eddy current problems in 2-D domains using solenoidal approximation	266
EC2D2T3	
Eddy current problems in 2-D domains using transversal approximation $\dots$ Edge	269
To describe an edge	271
EdgeList  Class to construct a list of adass having some common proporties	272
Class to construct a list of edges having some common properties EigenProblemSolver	2/3
Class to find eigenvalues and corresponding eigenvectors of a given matrix in a generalized eigenproblem, <i>i.e.</i> Find scalars I and non-null vectors v such that $[K]\{v\} = I[M]\{v\}$ where $[K]$ and $[M]$ are symmetric matrices. The eigenproblem can be originated from a PDE. For this, we will refer to the matrices K and M as <i>Stiffness</i> and <i>Mass</i> matrices respectively	275
Elas2DQ4	
To build element equations for 2-D linearized elasticity using 4-node quadrilaterals	280
Elas2DT3	
To build element equations for 2-D linearized elasticity using 3-node triangles	284
Elas3DH8	200
To build element equations for 3-D linearized elasticity using 8-node hexahedra Elas3DT4	288
To build element equations for 3-D linearized elasticity using 4-node tetrahedra	290
O	292
ElementList	
Class to construct a list of elements having some common properties	302
Ellipse  To store and treat an ellipsoidal figure	303
Equa_Electromagnetics< T_, NEN_, NEE_, NSN_, NSE_ >	303
Abstract class for Electromagnetics Equation classes	305
Equa_Fluid < T_, NEN_, NEE_, NSN_, NSE_ >  Abstract class for Fluid Dynamics Equation classes	206
Abstract class for Fluid Dynamics Equation classes	300
• •	307
Equa_Porous< T_, NEN_, NEE_, NSN_, NSE_>	
Abstract class for Porous Media Finite Element classes Equa_Solid < T_, NEN_, NEE_, NSN_, NSE_ >	310
Abstract class for Solid Mechanics Finite Element classes	312
Equa_Therm< T_, NEN_, NEE_, NSN_, NSE_>	012
Abstract class for Heat transfer Finite Element classes	314
Equation < T_, NEN_, NEE_, NSN_, NSE_>	
Abstract class for all equation classes	317
Estimator  To calculate an a posteriori estimator of the solution	220
To calculate an a posteriori estimator of the solution FastMarching2D	J4 <b>9</b>
To run a Fast Marching Method on 2-D structured uniform grids	331

	333
	335
	337
	339
	345
	348
isoparame	
· perfect	
	351
perfect	
	356
	361
	363
	366
	367
	377
	378
ode line	
	380
~	382
ngle ele-	
~	385
	300
	ia using r perfect r perfect r perfect ode line ode line

LCL1D		
Class to solve the linear conservation law (Hyperbolic equation) i MUSCL Finite Volume scheme	n 1-D by a	390
LCL2DT		
Class to solve the linear hyperbolic equation in 2-D by a MUSCL Fin scheme on triangles	ite Volume	392
LCL3DT		
Class to solve the linear conservation law equation in 3-D by a MU Volume scheme on tetrahedra		395
Line2  To describe a 2-Node planar line finite element		398
Line3		
To describe a 3-Node quadratic planar line finite element Linear Solver< T_ $>$		400
Class to solve systems of linear equations by iterative methods		402
LocalMatrix < T_, NR_, NC_ >		
Handles small size matrices like element matrices, with a priori kno LocalVect $<$ T $_{-}$ $N_{-}$ $>$	own size	407
Handles small size vectors like element vectors		414
Material		117
To treat material data. This class enables reading material data in materials. It also returns these informations by means of its members .		419
Matrix $<$ $T>$ Virtual class to handle matrices for all storage formats		422
Mesh		422
To store and manipulate finite element meshes		431
MeshAdapt		451
To adapt mesh in function of given solution $\dots \dots \dots$ Muscl		451
Parent class for hyperbolic solvers with Muscl scheme Muscl1D		456
Class for 1-D hyperbolic solvers with Muscl scheme		460
Muscl2DT  Class for 2-D hyperbolic solvers with Muscl scheme		461
Muscl3DT		
Class for 3-D hyperbolic solvers with Muscl scheme using tetrahedr MyNLAS	a	463
Abstract class to define by user specified function		464
MyOpt  Abstract class to define by user specified optimization function		466
NLASSolver		
To solve a system of nonlinear algebraic equations of the form $f(u) = Node$	= 0	467
To describe a node		473
NodeList		2, 0
Class to construct a list of nodes having some common properties		477
NSP2DQ41  Builds finite element arrays for incompressible Navier-Stokes equat domains using $Q_1/P_0$ element and a penaly formulation for the incomity condition	mpressibil-	479
ODESolver	<del></del>	
To solve a system of ordinary differential equations OFELIException		481
To handle exceptions in OFELI		492

OptSolver		
To solve an optimization problem with bound constraints		492
Partition		
To partition a finite element mesh into balanced submeshes Penta6		499
Defines a 6-node pentahedral finite element using $P_1$ interpolation in l dinates (s.x,s.y) and $Q_1$ isoparametric interpolation in local coordinax,s.z) and (s.y,s.z)		503
PhaseChange		303
This class enables defining phase change laws for a given material .		505
Point $<$ T $>$ Defines a point with arbitrary type coordinates		506
Point2D< T_>		500
Defines a 2-D point with arbitrary type coordinates		509
Polygon  To store and treat a polygonal figure		512
Prec< T_>		312
To set a preconditioner		514
To prescribe various types of data by an algebraic expression. Data main boundary conditions, forces, tractions, fluxes, initial condition. All types can be defined through an enumerated variable	these data	518
Quad4		
Defines a 4-node quadrilateral finite element using $Q_1$ isoparametric in	•	519
tion		319
To perform various reconstruction operations		521
Rectangle  To store and treat a rectangular figure		523
Side		
To store and treat finite element sides (edges in 2-D or faces in 3-D) . SideList		524
Class to construct a list of sides having some common properties		531
SkMatrix $<$ T $>$ To handle square matrices in skyline storage format		533
SkSMatrix< T_>		
To handle symmetric matrices in skyline storage format		542
Sphere  To store and treat a sphere		550
SpMatrix< T_ >		
To handle matrices in sparse storage format		552
SteklovPoincare2DBE	ariria aan	
Solver of the Steklov Poincare problem in 2-D geometries using piece stant boundary elemen		555
Tabulation		
To read and manipulate tabulated functions		557
Tetra4  Defines a three-dimensional 4-node tetrahedral finite element using P	1 interpo-	
lation		558
To be a displayed time counting		ECO
To handle elapsed time counting		560
TimeStepping  To solve time stepping problems, i.e. systems of linear ordinary dequations of the form $[A2]\{y''\} + [A1]\{y'\} + [A0]\{y\} = \{b\}$		561

TINS2DT3S	
Builds finite element arrays for transient incompressible fluid flow using Navier-Stokes equations in 2-D domains. Numerical approximation uses stabilized 3-node triangle finite elements for velocity and pressure. 2nd-order projection scheme is used for time integration	568
TINS3DT4S	
Builds finite element arrays for transient incompressible fluid flow using Navier-Stokes equations in 3-D domains. Numerical approximation uses stabilized 4-node tatrahedral finite elements for velocity and pressure. 2nd-order projection scheme is used for time integration	570
Triang3	
Defines a 3-Node (P <sub>1</sub> ) triangle	572
Triang6S	
Defines a 6-Node straight triangular finite element using P <sub>2</sub> interpolation	574
triangle	
Defines a triangle. The reference element is the rectangle triangle with two unit	
edges	576
Triangle	
To store and treat a triangle	578
TrMatrix < T_ >	
To handle tridiagonal matrices	579
Vect< T_>	
To handle general purpose vectors	581
WaterPorous2D	
To solve water flow equations in porous media (1-D)	610

16 OFELI's Reference Guide

# **Module Documentation**

## 5.1 General Purpose Equations

Gathers equation related classes.

#### Classes

- class AbsEqua < T\_>
  - Mother abstract class to describe equation.
- class Equation < T\_, NEN\_, NEE\_, NSN\_, NSE\_ >
  - Abstract class for all equation classes.
- class Estimator

To calculate an a posteriori estimator of the solution.

#### **Functions**

- template < class T\_, size\_t N\_, class E\_ >
   void element\_assembly (const E\_ &e, const LocalVect < T\_, N\_ > &be, Vect < T\_ > &b)
   Assemble local vector into global vector.
- template < class T\_, size\_t N\_, class E\_ > void element\_assembly (const E\_ &e, const LocalMatrix < T\_, N\_, N\_ > &ae, Vect < T\_ > &b)
   Assemble diagonal local vector into global vector.
- template < class  $T_-$ , size\_t  $N_-$ , class  $E_-$  > void element\_assembly (const  $E_-$  &e, const LocalMatrix <  $T_-$ ,  $N_-$ ,  $N_-$  > &ae, Matrix <  $T_-$  > \*A)

Assemble local matrix into global matrix.

- template < class  $T_-$ , size\_t  $N_-$ , class  $E_-$  > void element\_assembly (const  $E_-$  &e, const LocalMatrix <  $T_-$ ,  $N_-$ ,  $N_-$  > &ae, SkMatrix <  $T_-$  > &A)
  - Assemble local matrix into global skyline matrix.
- template < class T\_, size\_t N\_, class E\_ > void element\_assembly (const E\_ &e, const LocalMatrix < T\_, N\_, N\_ > &ae, SkSMatrix < T\_ > &A)
  - Assemble local matrix into global symmetric skyline matrix.
- template < class T\_, size\_t N\_, class E\_ > void element\_assembly (const E\_ &e, const LocalMatrix < T\_, N\_, N\_ > &ae, SpMatrix < T\_ > &A)

Assemble local matrix into global sparse matrix.

• template < class  $T_-$ , size\_t  $N_->$ 

void side\_assembly (const Element &e, const LocalMatrix< T\_, N\_, N\_ > &ae, SpMatrix< T\_ > &A)

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

• template<class T\_, size\_t N\_>

void side\_assembly (const Element &e, const LocalMatrix< T\_, N\_, N\_ > &ae, SkSMatrix< T\_ > &A)

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

• template < class  $T_-$ , size  $_-$ t  $N_- >$ 

void side\_assembly (const Element &e, const Local Matrix< T\_, N\_, N\_ > &ae, SkMatrix< T\_ > &A)

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

• template < class  $T_-$ , size\_t  $N_->$ 

 $void \ \underline{side\_assembly} \ (const \ Element \ \&e, const \ Local Vect < T_-, N_- > \&be, \ Vect < T_- > \&b)$ 

Side assembly of local vector into global vector.

• ostream & operator << (ostream &s, const Estimator &r)

Output estimator vector in output stream.

#### 5.1.1 Detailed Description

Gathers equation related classes.

#### **5.1.2** Function Documentation

void element\_assembly ( const  $E_-$  & e, const LocalVect<  $T_-$ ,  $N_-$  > & be, Vect<  $T_-$  > & b

Assemble local vector into global vector.

#### **Parameters**

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

#### Author

Rachid Touzani

#### Copyright

**GNU Lesser Public License** 

void element\_assembly ( const E\_ & e, const LocalMatrix< T\_, N\_, N\_ > & ae, Vect< T\_ > & b )

Assemble diagonal local vector into global vector.

#### **Parameters**

in	e	Reference to local entity (Element or Side)
----	---	---

#### Parameters

in	ae	Local matrix
in,out	b	Global vector

#### Author

Rachid Touzani

## Copyright

GNU Lesser Public License

# void element\_assembly ( const E\_ & e, const LocalMatrix< T\_, N\_, N\_ > & ae, Matrix< T\_ > \* A )

Assemble local matrix into global matrix.

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

#### **Parameters**

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

#### Author

Rachid Touzani

### Copyright

GNU Lesser Public License

# void element\_assembly ( const E\_ & e, const LocalMatrix< T\_, N\_, N\_ > & ae, SkMatrix< T\_ > & A )

Assemble local matrix into global skyline matrix.

#### Parameters

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

### Author

Rachid Touzani

### Copyright

GNU Lesser Public License

# void element\_assembly ( const E\_ & e, const LocalMatrix< T\_, N\_, N\_ > & ae, SkSMatrix< T\_ > & A )

Assemble local matrix into global symmetric skyline matrix.

#### **Parameters**

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

#### Author

Rachid Touzani

### Copyright

**GNU Lesser Public License** 

# void element\_assembly ( const E\_ & e, const LocalMatrix< T\_, N\_, N\_ > & ae, SpMatrix< T\_ > & A )

Assemble local matrix into global sparse matrix.

#### Parameters

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

#### Author

Rachid Touzani

### Copyright

**GNU Lesser Public License** 

# void side\_assembly ( const Element & e, const LocalMatrix< T\_-, N\_-, N\_- > & ae, SpMatrix< T\_- > & A )

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

#### Parameters

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

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# void side\_assembly ( const Element & e, const LocalMatrix < T\_, N\_, N\_ > & ae, SkSMatrix < T\_ > & A )

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

#### **Parameters**

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

#### Author

Rachid Touzani

### Copyright

GNU Lesser Public License

# void side\_assembly ( const Element & e, const LocalMatrix< $T_-$ , $N_-$ , $N_-$ > & ae, SkMatrix< $T_-$ > & A)

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

#### Parameters

in	е	Reference to local Element
in	ae	Local matrix
in,out	Α	Global matrix

#### Author

Rachid Touzani

### Copyright

**GNU Lesser Public License** 

void side\_assembly ( const Element & e, const LocalVect<  $T_-$ ,  $N_-$  > & be, Vect<  $T_-$  > & b ) Side assembly of local vector into global vector.

### Parameters

in	e	Reference to local Element
in	be	Local vector
in,out	b	Global vector

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

## 5.2 Conservation Law Equations

Conservation law equations.

#### Classes

• class ICPG1D

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 1-D.

class ICPG2DT

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 2-D.

class ICPG3DT

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 3-D.

class LCL1D

Class to solve the linear conservation law (Hyperbolic equation) in 1-D by a MUSCL Finite Volume scheme.

• class LCL2DT

Class to solve the linear hyperbolic equation in 2-D by a MUSCL Finite Volume scheme on triangles.

class LCL3DT

Class to solve the linear conservation law equation in 3-D by a MUSCL Finite Volume scheme on tetrahedra.

• class Muscl

Parent class for hyperbolic solvers with Muscl scheme.

class Muscl1D

Class for 1-D hyperbolic solvers with Muscl scheme.

• class Muscl2DT

Class for 2-D hyperbolic solvers with Muscl scheme.

• class Muscl3DT

Class for 3-D hyperbolic solvers with Muscl scheme using tetrahedra.

### 5.2.1 Detailed Description

Conservation law equations.

# 5.3 Electromagnetics

Electromagnetic equations.

#### Classes

• class BiotSavart

Class to compute the magnetic induction from the current density using the Biot-Savart formula.

• class EC2D1T3

 $Eddy\ current\ problems\ in\ 2\text{-}D\ domains\ using\ solenoidal\ approximation}.$ 

• class EC2D2T3

 $\label{lem:eq:current} \textit{Eddy current problems in 2-D domains using transversal approximation}.$ 

• class Equa\_Electromagnetics< T\_, NEN\_, NEE\_, NSN\_, NSE\_>

Abstract class for Electromagnetics Equation classes.

• class HelmholtzBT3

Builds finite element arrays for Helmholtz equations in a bounded media using 3-Node triangles.

## 5.3.1 Detailed Description

Electromagnetic equations.

# 5.4 Fluid Dynamics

Fluid Dynamics equations.

#### Classes

Abstract class for Fluid Dynamics Equation classes.

class NSP2DQ41

Builds finite element arrays for incompressible Navier-Stokes equations in 2-D domains using  $Q_1/P_0$  element and a penaly formulation for the incompressibility condition.

class TINS2DT3S

Builds finite element arrays for transient incompressible fluid flow using Navier-Stokes equations in 2-D domains. Numerical approximation uses stabilized 3-node triangle finite elements for velocity and pressure. 2nd-order projection scheme is used for time integration.

• class TINS3DT4S

Builds finite element arrays for transient incompressible fluid flow using Navier-Stokes equations in 3-D domains. Numerical approximation uses stabilized 4-node tatrahedral finite elements for velocity and pressure. 2nd-order projection scheme is used for time integration.

### 5.4.1 Detailed Description

Fluid Dynamics equations.

# 5.5 Interface Problems

Interface problems, including image processing.

### Classes

- class FastMarching2D
  - To run a Fast Marching Method on 2-D structured uniform grids.
- class FMMSolver

The Fast Marching Method solver.

# 5.5.1 Detailed Description

Interface problems, including image processing.

# 5.6 Laplace equation

Laplace and Poisson equations.

#### Classes

- class Equa\_Laplace < T\_, NEN\_, NEE\_, NSN\_, NSE\_ > Abstract class for classes about the Laplace equation.
- class Laplace1DL2

To build element equation for a 1-D elliptic equation using the 2-Node line element  $(P_1)$ .

• class Laplace1DL3

To build element equation for the 1-D elliptic equation using the 3-Node line (P<sub>2</sub>).

• class Laplace2DT3

To build element equation for the Laplace equation using the 2-D triangle element  $(P_1)$ .

• class Laplace2DT6

To build element equation for the Laplace equation using the 2-D triangle element  $(P_2)$ .

• class LaplaceDG2DP1

To build and solve the linear system for the Poisson problem using the DG P<sub>1</sub> 2-D triangle element.

• class SteklovPoincare2DBE

Solver of the Steklov Poincare problem in 2-D geometries using piecewie constant boundary elemen.

### 5.6.1 Detailed Description

Laplace and Poisson equations.

# 5.7 Porous Media problems

Porous Media equation classes.

### Classes

• class Equa\_Porous< T\_, NEN\_, NEE\_, NSN\_, NSE\_ > Abstract class for Porous Media Finite Element classes.

## 5.7.1 Detailed Description

Porous Media equation classes.

### 5.8 Solid Mechanics

Solid Mechanics finite element equations.

#### Classes

- class Bar2DL2
  - To build element equations for Planar Elastic Bar element with 2 DOF (Degrees of Freedom) per node.
- class Beam3DL2
  - To build element equations for 3-D beam equations using 2-node lines.
- class Elas2DQ4
  - To build element equations for 2-D linearized elasticity using 4-node quadrilaterals.
- class Elas2DT3
  - To build element equations for 2-D linearized elasticity using 3-node triangles.
- class Elas3DH8
  - To build element equations for 3-D linearized elasticity using 8-node hexahedra.
- class Elas3DT4
  - To build element equations for 3-D linearized elasticity using 4-node tetrahedra.
- class Equa\_Solid < T\_, NEN\_, NEE\_, NSN\_, NSE\_ >
  - Abstract class for Solid Mechanics Finite Element classes.

### 5.8.1 Detailed Description

Solid Mechanics finite element equations.

### 5.9 Heat Transfer

Heat Transfer equations.

#### Classes

• class DC1DL2

Builds finite element arrays for thermal diffusion and convection in 1-D using 2-Node elements.

• class DC2DT3

Builds finite element arrays for thermal diffusion and convection in 2-D domains using 3-Node triangles.

• class DC2DT6

Builds finite element arrays for thermal diffusion and convection in 2-D domains using 6-Node triangles.

• class DC3DAT3

Builds finite element arrays for thermal diffusion and convection in 3-D domains with axisymmetry using 3-Node triangles.

• class DC3DT4

Builds finite element arrays for thermal diffusion and convection in 3-D domains using 4-Node tetrahedra.

• class Equa\_Therm< T\_, NEN\_, NEE\_, NSN\_, NSE\_>

Abstract class for Heat transfer Finite Element classes.

class PhaseChange

This class enables defining phase change laws for a given material.

### 5.9.1 Detailed Description

Heat Transfer equations.

## 5.10 Input/Output

Input/Output utility classes.

#### Classes

• class IOField

Enables working with files in the XML Format.

class IPF

To read project parameters from a file in IPF format.

• class Prescription

To prescribe various types of data by an algebraic expression. Data may consist in boundary conditions, forces, tractions, fluxes, initial condition. All these data types can be defined through an enumerated variable.

#### **Macros**

• #define MAX\_NB\_PAR 50

Maximum number of parameters.

• #define MAX\_ARRAY\_SIZE 100

Maximum array size.

• #define MAX\_INPUT\_STRING\_LENGTH 100

Maximum string length.

• #define FILENAME\_LENGTH 150

Length of a string defining a file name.

• #define MAX\_FFT\_SIZE 15

Maximal size for the FFT Table This table can be used by the FFT for any number of points from 2 up to  $MAX\_FFT\_SIZE$ . For example, if  $MAX\_FFT\_SIZE = 14$ , then we can transform anywhere from 2 to  $2^15 = 32,768$  points, using the same sine and cosine table.

### 5.10.1 Detailed Description

Input/Output utility classes.

### 5.10.2 Macro Definition Documentation

### #define MAX\_NB\_PAR 50

Maximum number of parameters.

Used in class IPF

#### #define MAX\_ARRAY\_SIZE 100

Maximum array size.

Used in class IPF

#### #define MAX\_INPUT\_STRING\_LENGTH 100

Maximum string length.

Used in class IPF

# 5.11 Utilities

Utility functions and classes.

#### **Files**

• file OFELI.h

Header file that includes all kernel classes of the library.

file OFELI\_Config.h

File that contains some macros.

• file constants.h

File that contains some widely used constants.

#### Classes

• class Funct

A simple class to parse real valued functions.

• class Tabulation

To read and manipulate tabulated functions.

• class Point< T\_>

Defines a point with arbitrary type coordinates.

• class Point2D< T\_>

Defines a 2-D point with arbitrary type coordinates.

• class OFELIException

To handle exceptions in OFELI.

• class Gauss

Calculate data for Gauss integration.

class Timer

To handle elapsed time counting.

#### Macros

- #define OFELI\_E 2.71828182845904523536028747135
- #define OFELI\_PI 3.14159265358979323846264338328

- #define OFELL\_SQRT2 1.41421356237309504880168872421
- #define OFELI\_SQRT3 1.73205080756887729352744634151
  #define OFELI\_ONEOVERPI 0.31830988618379067153776752675
- #define OFELI\_GAUSS2 0.57735026918962576450914878050196
- #define OFELLEPSMCH DBL\_EPSILON
- #define OFELI\_TOLERANCE OFELI\_EPSMCH\*10000
- #define VLG 1.e10
- #define OFELI\_IMAG std::complex<double>(0.,1.);
- #define PARSE(exp, var) the Parser. Parse(exp, var)
- #define EVAL(d) the Parser. Eval(d)
- #define CATCH\_EXCEPTION

# **Typedefs**

• typedef unsigned long lsize\_t

This type stands for type unsigned long.

• typedef double real\_t

This type stands for double.

• typedef std::complex< double > complex\_t

This type stands for type std::complex<double>

#### **Functions**

• ostream & operator<< (ostream &s, const complex\_t &x)

Output a complex number.

• ostream & operator << (ostream &s, const std::string &c)

Output a string.

• template<class T\_>

ostream & operator << (ostream &s, const vector <  $T_- >$  &v)

Output a vector instance.

• template<class T\_>

ostream & operator << (ostream &s, const std::list < T\_> &l)

Output a vector instance.

• template<class T\_>

ostream & operator << (ostream &s, const std::pair < T<sub>-</sub>, T<sub>-</sub> > &a)

Output a pair instance.

• void saveField (Vect< real\_t > &v, string output\_file, int opt)

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

void saveField (const Vect< real\_t > &v, const Mesh &mesh, string output\_file, int opt)
 Save a vector to an output file in a given file format.

• void saveField (Vect< real\_t > &v, const Grid &g, string output\_file, int opt)

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

• void saveGnuplot (string input\_file, string output\_file, string mesh\_file, int f=1)

Save a vector to an input Gnuplot file.

• void saveGnuplot (Mesh &mesh, string input\_file, string output\_file, int f=1)

Save a vector to an input Gnuplot file.

• void saveTecplot (string input\_file, string output\_file, string mesh\_file, int f=1)

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

• void saveTecplot (Mesh &mesh, string input\_file, string output\_file, int f=1)

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

void saveVTK (string input\_file, string output\_file, string mesh\_file, int f=1)

Save a vector to an output VTK file.

• void saveVTK (Mesh &mesh, string input\_file, string output\_file, int f=1)

Save a vector to an output VTK file.

• void saveGmsh (string input\_file, string output\_file, string mesh\_file, int f=1)

Save a vector to an output Gmsh file.

• void saveGmsh (Mesh &mesh, string input\_file, string output\_file, int f=1)

Save a vector to an output Gmsh file.

• ostream & operator << (ostream &s, const Tabulation &t)

Output Tabulated function data.

```
• template<class T_>
  bool operator== (const Point< T_- > \&a, const Point< T_- > \&b)
      Operator ==
• template<class T_>
  Point< T_- > operator+ (const Point< T_- > &a, const Point< T_- > &b)
      Operator +

    template < class T<sub>−</sub> >

  Point< T_- > operator+ (const Point< T_- > &a, const T_- &x)
• template<class T_>
  Point< T_- > operator- (const Point < T_- > &a)
      Unary Operator -
  template<class T_>
  Point< T_- > operator- (const Point< T_- > &a, const Point< T_- > &b)
      Operator -
• template<class T_>
  Point< T_- > operator- (const Point< T_- > &a, const T_- &x)
      Operator -
• template<class T_>
  Point< T_- > operator* (const T_- &a, const Point<math>< T_- > \&b)
      Operator *
  template<class T_>
  Point< T_- > operator* (const int &a, const Point< T_- > &b)
      Operator *.
• template<class T_>
  Point< T_- > operator* (const Point< T_- > &b, const T_- &a)
      Operator /

    template<class T<sub>-</sub>>

  Point< T_- > operator* (const Point< T_- > &b, const int &a)
      Operator *
• template<class T_>
  T_- operator* (const Point< T_- > &a, const Point< T_- > &b)
      Operator *
• template<class T_>
  Point< T_- > operator / (const Point < T_- > &b, const T_- &a)
      Operator /

    bool areClose (const Point < double > &a, const Point < double > &b, double toler=OFE ←

  LI_TOLERANCE)
      Return true if both instances of class Point < double > are distant with less then toler
• double SqrDistance (const Point< double > &a, const Point< double > &b)
      Return squared euclidean distance between points a and b

    double Distance (const Point< double > &a, const Point< double > &b)

      Return euclidean distance between points a and b
• bool operator< (const Point< size_t > &a, const Point< size_t > &b)
      Comparison operator. Returns true if all components of first vector are lower than those of second one.
• template<class T_>
  std::ostream & operator<< (std::ostream &s, const Point< T_> &a)
      Output point coordinates.
```

```
    template<class T<sub>−</sub>>

  bool operator== (const Point2D< T_-> &a, const Point2D< T_-> &b)
     Operator ==.
• template<class T_>
  Point2D< T_-> operator+ (const Point2D< T_-> &a, const Point2D< T_-> &b)
     Operator +.

    template < class T_>

  Point2D< T_-> operator+ (const Point2D< T_-> &a, const T_- &x)
     Operator +.
• template<class T_>
  Point2D< T_-> operator- (const Point2D< T_-> &a)
     Unary Operator -
• template<class T_>
  Point2D< T_-> operator- (const Point2D< T_-> &a, const Point2D< T_-> &b)
     Operator -
• template<class T_>
  Point2D< T_-> operator- (const Point2D< T_-> &a, const T_- &x)
     Operator -
• template<class T_>
  Point2D< T_-> operator* (const T_- &a, const Point2D< T_-> &b)
     Operator *.
• template<class T_>
  Point2D< T_-> operator* (const int &a, const Point2D< T_-> &b)
• template<class T_>
  Point2D< T_-> operator* (const Point2D< T_-> &b, const T_- &a)
     Operator /

    template<class T₋>

  Point2D< T_-> operator* (const Point2D< T_-> &b, const int &a)
     Operator *
• template<class T_>
  T_- operator* (const Point2D< T_- > &b, const Point2D< T_- > &a)
     Operator *.
• template<class T_>
  Point2D< T_-> operator/ (const Point2D< T_-> &b, const T_- &a)
     Operator /

    bool areClose (const Point2D < real_t > &a, const Point2D < real_t > &b, real_t toler=OFE ←

  LI_TOLERANCE)
     Return true if both instances of class Point2D<real_t> are distant with less then toler [Default: 0FEL↔
     I\_EPSMCH].
• real_t SqrDistance (const Point2D< real_t > &a, const Point2D< real_t > &b)
     Return squared euclidean distance between points a and b
• real_t Distance (const Point2D< real_t > &a, const Point2D< real_t > &b)
     Return euclidean distance between points a and b
• template<class T_>
  std::ostream & operator << (std::ostream &s, const Point2D < T_> &a)
     Output point coordinates.

    real_t Discrepancy (Vect< real_t > &x, const Vect< real_t > &y, int n, int type=1)

     Return discrepancy between 2 vectors x and y

    real_t Discrepancy (Vect< complex_t > &x, const Vect< complex_t > &y, int n, int type=1)
```

Return discrepancy between 2 vectors x and y

• void getMesh (string file, ExternalFileFormat form, Mesh &mesh, size\_t nb\_dof=1)

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

• void getBamg (string file, Mesh &mesh, size\_t nb\_dof=1)

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

• void getEasymesh (string file, Mesh &mesh, size\_t nb\_dof=1)

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

void getGambit (string file, Mesh &mesh, size\_t nb\_dof=1)

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

• void getGmsh (string file, Mesh &mesh, size\_t nb\_dof=1)

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

• void getMatlab (string file, Mesh &mesh, size\_t nb\_dof=1)

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

• void getNetgen (string file, Mesh &mesh, size\_t nb\_dof=1)

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

• void getTetgen (string file, Mesh &mesh, size\_t nb\_dof=1)

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

• void getTriangle (string file, Mesh &mesh, size\_t nb\_dof=1)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

void BSpline (size\_t n, size\_t t, Vect< Point< real\_t >> &control, Vect< Point< real\_t >> &output, size\_t num\_output)

Function to perform a B-spline interpolation.

• void banner (const string &prog="")

Outputs a banner as header of any developed program.

template<class T\_>

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

Function to sort a vector.

template<class T\_>

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

Function to sort a vector.

template < class T\_, class C\_>

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

Function to sort a vector according to a key function.

• int Sgn (real\_t a)

```
Return sign of a: - 1 or 1.

    real_t Abs2 (complex_t a)

      Return square of modulus of complex number a
• real_t Abs2 (real_t a)
      Return square of real number a
• real_t Abs (real_t a)
      Return absolute value of a

    real_t Abs (complex_t a)

      Return modulus of complex number a
• real_t Abs (const Point< real_t > &p)
      Return Norm of vector a

    real_t Conjg (real_t a)

      Return complex conjugate of real number a
• complex_t Conjg (complex_t a)
      Return complex conjugate of complex number a
• real_t Max (real_t a, real_t b, real_t c)
      Return maximum value of real numbers a, b and c
• int Max (int a, int b, int c)
      Return maximum value of integer numbers a, b and c
• real_t Min (real_t a, real_t b, real_t c)
      Return minimum value of real numbers a, b and c
• int Min (int a, int b, int c)
      Return minimum value of integer numbers a, b and c
• real_t Max (real_t a, real_t b, real_t c, real_t d)
      Return maximum value of integer numbers a, b, c and d
• int Max (int a, int b, int c, int d)
      Return maximum value of integer numbers a, b, c and d
• real_t Min (real_t a, real_t b, real_t c, real_t d)
      Return minimum value of real numbers a, b, c and d
• int Min (int a, int b, int c, int d)
      Return minimum value of integer numbers a, b, c and d
real_t Arg (complex_t x)
      Return argument of complex number x

    complex_t Log (complex_t x)

      Return principal determination of logarithm of complex number x

    template<class T<sub>-</sub>>

  T_- \operatorname{Sqr} (T_- x)
      Return square of value x
• template<class T_>
  void Scale (T_- a, const vector < T_- > &x, vector < T_- > &y)
      Mutiply vector x by a and save result in vector y
• template<class T_>
  void Scale (T_- a, const Vect< T_- > &x, Vect< T_- > &y)
      Mutiply vector x by a and save result in vector y
• template<class T_>
  void Scale (T_- a, vector< T_- > &x)
```

```
Mutiply vector x by a
• template<class T_>
  void Xpy (size_t n, T_-*x, T_-*y)
      Add array x to y
• template<class T_>
  void Xpy (const vector < T_- > &x, vector < T_- > &y)
      Add vector x to y
• template<class T_>
  void Axpy (size_t n, T_- a, T_- *x, T_- *y)
      Multiply array x by a and add result to y
• template<class T_>
  void Axpy (T_- a, const vector< T_- > &x, vector< T_- > &y)
      Multiply vector x by a and add result to y
• template<class T_>
  void Axpy (T_a, const Vect < T_s & x, Vect < T_s & y)
      Multiply vector x by a and add result to y
• template<class T_>
  void Copy (size_t n, T_- *x, T_- *y)
      Copy array x to y n is the arrays size.
• real_t Error2 (const vector< real_t > &x, const vector< real_t > &y)
      Return absolute L2 error between vectors x and y

    real_t RError2 (const vector < real_t > &x, const vector < real_t > &y)

      Return absolute L^2 error between vectors x and y
• real_t ErrorMax (const vector < real_t > &x, const vector < real_t > &y)
      Return absolute Max. error between vectors x and y
• real_t RErrorMax (const vector < real_t > &x, const vector < real_t > &y)
      Return relative Max. error between vectors x and y
• template<class T_>
  T_- Dot (size_t n, T_- *x, T_- *y)
      Return dot product of arrays x and y
• real_t Dot (const vector < real_t > &x, const vector < real_t > &y)
      Return dot product of vectors x and y.
• template<class T_>
  T_- Dot (const Point < T_- > &x, const Point < T_- > &y)
      Return dot product of x and y
real_t exprep (real_t x)
      Compute the exponential function with avoiding over and underflows.
• template<class T_>
  void Assign (vector < T_- > &v, const T_- &a)
      Assign the value a to all entries of a vector v
• template<class T_>
  void clear (vector < T_- > &v)
      Assign 0 to all entries of a vector.
• template<class T_>
  void clear (Vect< T_-> &v)
      Assign 0 to all entries of a vector.
real_t Nrm2 (size_t n, real_t *x)
      Return 2-norm of array x
```

• real\_t Nrm2 (const vector < real\_t > &x)

Return 2-norm of vector x

• template<class T\_>

real\_t Nrm2 (const Point $< T_- > &a$ )

Return 2-norm of a

• bool Equal (real\_t x, real\_t y, real\_t toler=OFELI\_EPSMCH)

Function to return true if numbers x and y are close up to a given tolerance toler

• char itoc (int i)

Function to convert an integer to a character.

• std::string itos (int i)

Function to convert an integer to a string.

• std::string itos (size\_t i)

Function to convert an integer to a string.

• std::string dtos (real\_t d)

Function to convert a real to a string.

• template<class T\_>

T<sub>-</sub> stringTo (const std::string &s)

Function to convert a string to a template type parameter.

# 5.11.1 Detailed Description

Utility functions and classes.

### 5.11.2 Macro Definition Documentation

#### #define OFELI\_E 2.71828182845904523536028747135

Value of e or exp (with 28 digits)

### #define OFELI\_PI 3.14159265358979323846264338328

Value of Pi (with 28 digits)

#### 

Value of 1/3 (with 28 digits)

#### 

Value of 1/6 (with 28 digits)

#### 

Value of 1/12 (with 28 digits)

### #define OFELI\_SQRT2 1.41421356237309504880168872421

Value of sqrt(2) (with 28 digits)

#### #define OFELI\_SQRT3 1.73205080756887729352744634151

Value of sqrt(3) (with 28 digits)

#### #define OFELI\_ONEOVERPI 0.31830988618379067153776752675

Value of 1/Pi (with 28 digits)

#### #define OFELI\_GAUSS2 0.57735026918962576450914878050196

Value of 1/sqrt(3) (with 32 digits)

#### #define OFELI\_EPSMCH DBL\_EPSILON

Value of Machine Epsilon

#### #define OFELI\_TOLERANCE OFELI\_EPSMCH\*10000

Default tolerance for an iterative process = OFELI\_EPSMCH \* 10000

#### #define VLG 1.e10

Very large number: A real number for penalty

# #define OFELI\_IMAG std::complex<double>(0.,1.);

= Unit imaginary number (i)

#### #define PARSE( exp, var ) the Parser. Parse (exp, var)

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

#### #define EVAL( d ) the Parser. Eval(d)

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

#### #define CATCH\_EXCEPTION

# Value:

```
catch(OFELIException &e) {
    std::cout << "OFELI error: " << e.what() << endl;
    return 1;
}
catch(runtime_error &e) {
    std::cout << "Runtime error: " << e.what() << endl;
    return 1;
}
catch( ... ) {
    std::cout << "Unexpected error: " << endl;
    return 1;
}</pre>
```

This macro can be inserted after a try loop to catch a thrown exception.

# 5.11.3 Function Documentation

```
ostream & operator << ( ostream & s, const complex_t & x)
Output a complex number.
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
ostream & operator << ( ostream & s, const std::string & c )
Output a string.
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
ostream & operator << ( ostream & s, const vector < T_-> & v )
Output a vector instance.
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
ostream & operator << ( ostream & s, const std::list < T_-> & l)
Output a vector instance.
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
ostream & operator << ( ostream & s, const std::pair < T_-, T_- > & a)
Output a pair instance.
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
```

# void saveField ( $Vect < real_t > \& v$ , $string output_file$ , int opt)

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

Case where the vector contains mesh information

#### Parameters

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

#### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

# void saveField ( const Vect< real\_t > & v, const Mesh & mesh, string $output_file$ , int opt )

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

Case where the vector does not contain mesh information

#### **Parameters**

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

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# void saveField ( Vect< real\_t > & v, const Grid & g, string output\_file, int opt = VTK)

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

in	v	Vect instance to save
in	8	Grid instance
in	output_file	Output file where to save the vector
in	opt	Option to choose file format to save. This is to be chosen among
42		enumerated values: GMSH, VTK  OFELI's Reference Guide

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# void saveGnuplot (string input\_file, string output\_file, string mesh\_file, int f = 1)

Save a vector to an input Gnuplot file.

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

http://www.gnuplot.info/

#### **Parameters**

in	input_file	Input file (OFELI XML file containing a field).
in	output_file	Output file (gnuplot format file)
in	mesh_file	File containing mesh data
in	f	Field is stored each f time step [Default: 1

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# void saveGnuplot (Mesh & mesh, string input\_file, string output\_file, int f = 1)

Save a vector to an input Gnuplot file.

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

http://www.gnuplot.info/

#### Parameters

in	mesh	Reference to Mesh instance
in	input_file	Input file (OFELI XML file containing a field).
in	output_file	Output file (gnuplot format file)
in	f	Field is stored each f time step [Default: 1]

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# void save Tecplot (string input\_file, string output\_file, string mesh\_file, int f = 1)

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

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

#### **Parameters**

in	input_file	Input file (OFELI XML file containing a field).
in	output_file	Output file (gnuplot format file)
in	mesh_file	File containing mesh data
in	f	Field is stored each f time step [Default: 1]

#### Author

Rachid Touzani

#### Copyright

**GNU Lesser Public License** 

# void saveTecplot (Mesh & mesh, string input\_file, string output\_file, int f = 1)

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

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

# Parameters

in	mesh	Reference to Mesh instance
in	input_file	Input file (OFELI XML file containing a field).
in	output_file	Output file (gnuplot format file)
in	f	Field is stored each f time step [Default: 1]

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# saveVTK (string input\_file, string output\_file, string mesh\_file, int f = 1)

Save a vector to an output VTK file.

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

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

#### **Parameters**

in	input_file	Input file (OFELI XML file containing a field).
in	output_file	Output file (VTK format file)
in	mesh_file	File containing mesh data
in	f	Field is stored each f time step [Default: 1]

#### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

#### saveVTK ( Mesh & mesh, string input\_file, string output\_file, int f = 1 )

Save a vector to an output VTK file.

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

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

#### Parameters

in	mesh	Reference to Mesh instance
in	input_file	Input file (OFELI XML file containing a field).
in	output_file	Output file (VTK format file)
in	f	Field is stored each f time step [Default: 1]

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# void saveGmsh (string input\_file, string output\_file, string mesh\_file, int f = 1)

Save a vector to an output Gmsh file.

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

in	input_file	Input file (OFELI XML file containing a field).
----	------------	---

in	output_file	Output file (Gmsh format file)
in	mesh_file	File containing mesh data
in	f	Field is stored each f time step [Default: 1]

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# void saveGmsh (Mesh & mesh, string input\_file, string output\_file, int f = 1)

Save a vector to an output Gmsh file.

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

#### Parameters

in	mesh	Reference to Mesh instance
in	input_file	Input file (OFELI XML file containing a field).
in	output_file	Output file (Gmsh format file)
in	f	Field is stored each f time step [Default: 1]

# Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# bool operator== ( const Point< T $_->$ & a, const Point< T $_->$ & b)

Operator ==

Return true if a=b, false if not.

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

```
Point< T_- > operator+ ( const Point< T_- > & a_r const Point< T_- > & b )
Operator +
   Return sum of two points a and b
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
Point< T_- > operator+ ( const Point< T_- > & a, const T_- & x)
Operator +
   Translate a by x
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
Point< T_- > operator- (const Point< T_- > & a)
Unary Operator -
   Return minus a
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
Point< T_- > operator- ( const Point< T_- > & a, const Point< T_- > & b)
   Return point a minus point b
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
```

```
Point< T_- > operator- (const Point< T_- > & a, const T_- & x)
Operator -
   Translate a by -x
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
Point< T_- > operator* ( const T_- & a_t const Point<math>< T_- > & b )
Operator *
   Return point b premultiplied by constant a
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
Point< T_- > operator* ( const int & a, const Point< T_- > & b )
Operator *.
   Return point b divided by integer constant a
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
Point< T_- > operator* ( const Point< T_- > \& b, const T_- \& a)
Operator /
   Return point b multiplied by constant a
Point< T_- > operator* ( const Point< T_- > & b, const int & a )
Operator *
   Return point b postmultiplied by constant a
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
```

```
T_- operator* ( const Point< T_- > & b, const Point< T_- > & a)
   Return inner (scalar) product of points a and b
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
Point< T_-> operator/ ( const Point< T_-> & b, const T_- & a)
Operator /
   Return point b divided by constant a
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
bool areClose ( const Point < double > & a, const Point < double > & b, double toler =
OFELI_TOLERANCE )
Return true if both instances of class Point<double> are distant with less then toler
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
double SqrDistance ( const Point< double > & a, const Point< double > & b)
Return squared euclidean distance between points a and b
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
```

```
double Distance ( const Point < double > & a, const Point < double > & b)
Return euclidean distance between points a and b
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
bool operator< ( const Point< size_t > & a, const Point< size_t > & b )
Comparison operator. Returns true if all components of first vector are lower than those of second
one.
   Return minus a
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
ostream & operator << ( std::ostream & s, const Point < T<sub>-</sub> > & a )
Output point coordinates.
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
bool operator== ( const Point2D< T_-> & a, const Point2D< T_-> & b)
Operator ==.
   Return true if a=b, false if not.
Author
     Rachid Touzani
```

Copyright

**GNU Lesser Public License** 

```
Point2D< T_-> operator+ ( const Point2D< T_-> & a, const Point2D< T_-> & b)
   Return sum of two points a and b
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
Point2D< T_-> operator+ ( const Point2D< T_-> & a, const T_- & x )
Operator +.
   Translate a by x
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
Point2D< T_-> operator- ( const Point2D< T_-> & a )
Unary Operator -
   Return minus a
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
Point2D< T_-> operator- ( const Point2D< T_-> & a, const Point2D< T_-> & b )
   Return point a minus point b
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
```

```
Point2D< T_-> operator- ( const Point2D< T_-> & a, const T_- & x )
Operator -
   Translate a by -x
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
Point2D< T_-> operator* ( const T_- & a, const Point2D< T_-> & b)
Operator *.
   Return point b premultiplied by constant a
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
Point2D< T_-> operator* ( const int & a, const Point2D< T_-> & b)
Operator *.
   Return point b divided by integer constant a
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
Point2D< T_-> operator* ( const Point2D< T_-> & b, const T_- & a )
   Return point b postmultiplied by constant a
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
```

```
Point2D< T_-> operator* ( const Point2D< T_-> & b, const int & a )
   Return point b postmultiplied by constant a
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
T_- operator* ( const Point2D< T_- > & b, const Point2D< T_- > & a )
Operator *.
   Return point b postmultiplied by integer constant a.
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
Point2D< T_-> operator/ ( const Point2D< T_-> & b, const T_- & a)
Operator /
   Return point b divided by constant a
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
real_t SqrDistance ( const Point2D< real_t > & a, const Point2D< real_t > & b)
Return squared euclidean distance between points a and b
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
```

# real\_t Distance ( const Point2D< real\_t > & a, const Point2D< real\_t > & b)

Return euclidean distance between points a and b

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

# ostream & operator << ( std::ostream & s, const Point2D < T $_->$ & a)

Output point coordinates.

Author

Rachid Touzani

Copyright

**GNU** Lesser Public License

# real\_t Discrepancy ( Vect< real\_t > & x, const Vect< real\_t > & y, int n, int type = 1)

Return discrepancy between 2 vectors x and y

# Parameters

in,out	x	First vector (Instance of class Vect). On output, x is assigned the vector y
in	у	Second vector (Instance of class Vect)
in	n	Type of norm
		• 1: Weighted 1-Norm
		• 2: Weighted 2-Norm
		• 0: Max-Norm
in	type	Discrepancy type (0: Absolute, 1: Relative [Default])

#### Returns

Computed discrepancy value

# real\_t Discrepancy ( Vect< complex\_t > & x, const Vect< complex\_t > & y, int n, int type = 1)

Return discrepancy between 2 vectors x and y

in,out	x	First vector (Instance of class Vect). On output, x is assigned the vector y
--------	---	--

in	у	Second vector (Instance of class Vect)
in	n	Type of norm
		• 1: Weighted 1-Norm
		• 2: Weighted 2-Norm
		• 0: Max-Norm
in	type	Discrepancy type (0: Absolute, 1: Relative [Default])

#### Returns

Computed discrepancy value

# void getMesh ( string file, ExternalFileFormat form, Mesh & mesh, size\_t $nb\_dof = 1$ )

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

in	file	Input mesh file name.
in	form	Format of the mesh file. This one can be chosen among the enumerated values:
		<ul> <li>GMSH: Mesh generator Gmsh, see site: http://www.geuz.org/gmsh/</li> </ul>
		<ul> <li>MATLAB: Matlab file, see site: http://www.mathworks.com/products/matlab/</li> </ul>
		• EASYMESH: Easymesh is a 2-D mesh generator, see site: http://web.mit.edu/easymesh_v1.4/www/easymesh.html
		• GAMBIT: Gambit is a mesh generator associated to Fluent http://www.stanford.edu/class/me469b/gambit_download.html
		• BAMG: Mesh generator Bamg, see site: http://raweb.inria.fr/rapportsactivite/RA2002/gamma/uid25.← html
		<ul> <li>NETGEN: Netgen is a 3-D mesh generator, see site: http://www.hpfem.jku.at/netgen/</li> </ul>
		<ul> <li>TETGEN: Tetgen is a 3-D mesh generator, see site: http://tetgen.berlios.de/</li> </ul>
		• TRIANGLE_FF: Triangle is a 2-D mesh generator, see site: http://www.cs.cmu.edu/~quake/triangle.html
out	mesh	Mesh instance created by the function.
in	nb_dof	Number of degrees of freedom for each node. This information is not provided, in general, by mesh generators. Its default value here is 1.

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# void getBamg ( string file, Mesh & mesh, size\_t nb\_dof = 1 )

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

#### **Parameters**

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

#### Note

**Bamg** is a 2-D mesh generator. It allows to construct adapted meshes from a given metric. It was developed at INRIA, France. Available information can be found in the site: http://raweb.inria.fr/rapportsactivite/RA2002/gamma/uid25.html

#### **Parameters**

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

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# void getEasymesh ( string file, Mesh & mesh, size\_t $nb\_dof = 1$ )

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

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

#### Note

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

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

#### Parameters

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

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

#### void getGambit ( string file, Mesh & mesh, size\_t $nb\_dof = 1$ )

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

#### Note

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

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

#### **Parameters**

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

#### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

# void getGmsh ( string file, Mesh & mesh, size\_t nb\_dof = 1 )

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

# Note

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

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

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

#### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

# void getMatlab ( string file, Mesh & mesh, size\_t nb\_dof = 1 )

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

Note

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

# Parameters

in	file	Name of a file created by Matlab by executing the script file Matlab20FELI.m
ou <sup>-</sup>	mesh	Mesh instance created by the function.
in	nb_dof	Number of degrees of freedom for each node. This information is not provided, in general, by mesh generators. Its default value here is 1.

# Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# void getNetgen ( string file, Mesh & mesh, size\_t nb\_dof = 1 )

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

#### Note

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

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

#### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

# void getTetgen ( string file, Mesh & mesh, size\_t nb\_dof = 1 )

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

Note

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

#### **Parameters**

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

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# void getTriangle ( string file, Mesh & mesh, size\_t $nb\_dof = 1$ )

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

### Note

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

 $\verb|http://people.scs.fsu.edu/~burkardt/c\_src/triangle/triangle.html/|$ 

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

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

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

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

#### **Parameters**

in	file	File where to store mesh			
in	mesh	Mesh instance to save			
in	form	Format of the mesh file. This one can be chosen among the enumerated values:			
		<ul> <li>GMSH: Mesh generator and graphical postprocessor Gmsh: http://www.geuz.org/gmsh/</li> </ul>			
		• GNUPLOT: Well known graphics software: http://www.gnuplot.info/			
		• MATLAB: Matlab file: http://www.mathworks.com/products/matlab/			
		• TECPLOT: Commercial graphics software: http://www.tecplot.com			
		<ul> <li>VTK: Graphics format for the free postprocessor ParaView: http://public.kitware.com/VTK/</li> </ul>			

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

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

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

Note

**Gmsh** is a free mesh generator that can be downloaded from the site:  $http://www.geuz. \leftarrow org/gmsh/$ 

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

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

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

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

#### Note

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

http://www.gnuplot.info/

#### Parameters

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

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

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

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

#### Note

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

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

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

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

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

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

Note

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

http://www.tecplot.com

#### **Parameters**

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

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

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

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

Note

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

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

# Parameters

out	file	Output file in <b>VTK</b> format.
in	mesh	Mesh instance to save.

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

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

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

#### **Parameters**

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

#### Note

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

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

#### Parameters

in   <i>mesh</i>   Mesh instance.		in	mesh	Mesh instance.
-----------------------------------	--	----	------	----------------

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

BSpline ( size\_t n, size\_t t, Vect< Point< real\_t >> & control, Vect< Point< real\_t >> & output, size\_t num\_output )

Function to perform a B-spline interpolation.

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

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

#### Note

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

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# void banner ( const string & prog = " ")

Outputs a banner as header of any developed program.

#### **Parameters**

in	prog	Calling program name. Enables writing a copyright notice accompanying the	1
		program.	

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

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

Function to sort a vector.

qksort uses the famous quick sorting algorithm.

#### **Parameters**

in,out	а	Vector to sort.
in	begin	index of starting iterator
in	end	index of ending iterator

The calling program must provide an overloading of the operator < for the type T\_

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# void qksort ( std::vector $< T_- > & a$ , int begin, int end )

Function to sort a vector.

qksort uses the famous quick sorting algorithm.

#### **Parameters**

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

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# void qksort ( std::vector $< T_- > & a$ , int begin, int end, $C_-$ compare )

Function to sort a vector according to a key function. qksort uses the famous quick sorting algorithm.

#### **Parameters**

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

#### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

### void Scale ( $T_- a_r$ , const vector< $T_- > \& x_r$ , vector< $T_- > \& y$ )

Mutiply vector x by a and save result in vector y x and y are instances of class vector<T\_>

# void Scale ( $T_-a$ , const Vect< $T_- > \& x$ , Vect< $T_- > \& y$ )

Mutiply vector x by a and save result in vector y x and y are instances of class Vect<T\_>

```
void Scale (T_-a, vector<T_- > \& x)
Mutiply vector x by a
```

x is an instance of class vector<T $_->$ 

void Xpy ( const vector<  $T_- > \& x$ , vector<  $T_- > \& y$ )

Add vector x to y

x and y are instances of class vector<T\_>

void Axpy ( size\_t n,  $T_-a$ ,  $T_-*x$ ,  $T_-*y$  )

Multiply array x by a and add result to y n is the arrays size.

void Axpy (  $T_-a$ , const vector<  $T_- > \& x$ , vector<  $T_- > \& y$ )

Multiply vector x by a and add result to y x and y are instances of class vector<T $_>$ 

void Axpy (  $T_-a$ , const Vect<  $T_- > \& x$ , Vect<  $T_- > \& y$ )

Multiply vector x by a and add result to y x and y are instances of class  $Vect < T_- >$ 

 $T_-$  Dot ( size\_t n,  $T_- * x$ ,  $T_- * y$  )

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

double Dot ( const vector< real\_t > & x, const vector< real\_t > & y)

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

void clear ( vector<  $T_- > \& v$  )

Assign 0 to all entries of a vector.

Parameters

in v Vector to clear

void clear ( Vect< T $_->$  & v )

Assign 0 to all entries of a vector.

**Parameters** 

in v Vector to clear

# real\_t Nrm2 ( size\_t n, real\_t \* x )

Return 2-norm of array x

### Parameters

in	n	is Array length
in	x	Array to treat

# bool Equal ( real\_t x, real\_t y, real\_t toler = OFELI\_EPSMCH )

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

# 5.12 Vector and Matrix

Vector and matrix classes.

#### Classes

```
• class BMatrix< T_>
```

To handle band matrices.

• class DMatrix< T\_>

To handle dense matrices.

• class DSMatrix< T\_>

To handle symmetric dense matrices.

• class LocalMatrix< T\_, NR\_, NC\_>

Handles small size matrices like element matrices, with a priori known size.

• class LocalVect< T\_, N\_>

Handles small size vectors like element vectors.

• class SkMatrix< T\_>

To handle square matrices in skyline storage format.

• class SkSMatrix< T\_>

To handle symmetric matrices in skyline storage format.

• class SpMatrix< T\_>

To handle matrices in sparse storage format.

• class TrMatrix< T\_>

To handle tridiagonal matrices.

class Vect< T\_>

To handle general purpose vectors.

### **Functions**

```
• template<class T_>
```

```
Vect < T_- > operator* (const BMatrix < T_- > &A, const Vect < T_- > &b)
```

Operator \* (Multiply vector by matrix and return resulting vector.

• template<class T\_>

```
BMatrix< T_- > operator* (T_- a, const BMatrix<math>< T_- > \&A)
```

Operator \* (Premultiplication of matrix by constant)

• template<class T\_>

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

Output matrix in output stream.

• template<class T\_>

```
Vect < T_- > operator* (const DMatrix < T_- > &A, const Vect < T_- > &b)
```

Operator \* (Multiply vector by matrix and return resulting vector.

• template<class T\_>

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

Output matrix in output stream.

• template<class T\_>

```
Vect < T_- > operator* (const DSMatrix < T_- > &A, const Vect < T_- > &b)
```

Operator \* (Multiply vector by matrix and return resulting vector.

• template<class T\_>

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

Output matrix in output stream.

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

Operator \* (Multiply matrix x by scalar a)

template < class T\_, size\_t NR\_, size\_t NC\_>

LocalVect< T<sub>-</sub>, NR<sub>-</sub>> operator\* (const LocalMatrix< T<sub>-</sub>, NR<sub>-</sub>, NC<sub>-</sub>> &A, const LocalVect< T<sub>-</sub>, NC<sub>-</sub>> &x)

Operator \* (Multiply matrix A by vector x)

• template < class T\_, size\_t NR\_, size\_t NC\_>

LocalMatrix < T\_, NR\_, NC\_ > operator / (T\_a, const LocalMatrix < T\_, NR\_, NC\_ > &x)

Operator / (Divide matrix x by scalar a)

• template < class T\_, size\_t NR\_, size\_t NC\_>

LocalMatrix< T $_-$ , NR $_-$ , NC $_-$  > operator+ (const LocalMatrix< T $_-$ , NR $_-$ , NC $_-$  > &x, const LocalMatrix< T $_-$ , NR $_-$ , NC $_-$  > &y)

Operator + (Add matrix x to y)

• template<class T\_, size\_t NR\_, size\_t NC\_>

 $\label{eq:localMatrix} LocalMatrix < T_-, NR_-, NC_- > \underset{}{operator-} \mbox{ (const LocalMatrix} < T_-, NR_-, NC_- > \underset{}{\&x}, \mbox{ const LocalMatrix} < T_-, NR_-, NC_- > \underset{}{\&y})$ 

Operator - (Subtract matrix y from x)

• template < class T\_, size\_t NR\_, size\_t NC\_>

ostream & operator << (ostream &s, const LocalMatrix < T\_, NR\_, NC\_ > &A)

Output vector in output stream.

• template < class T\_, size\_t N\_>

LocalVect< T $_-$ , N $_-$ > operator+ (const LocalVect< T $_-$ , N $_-$ > &x, const LocalVect< T $_-$ , N $_-$ > &y)

*Operator* + (*Add two vectors*)

• template < class  $T_-$ , size\_t  $N_->$ 

LocalVect< T $_-$ , N $_-$ > operator- (const LocalVect< T $_-$ , N $_-$ > &x, const LocalVect< T $_-$ , N $_-$ > &y)

Operator - (Subtract two vectors)

• template < class  $T_-$ , size\_t  $N_->$ 

LocalVect $< T_-, N_- > operator* (T_- a, const LocalVect<math>< T_-, N_- > \&x)$ 

Operator \* (Premultiplication of vector by constant)

• template < class T\_, size\_t N\_>

LocalVect< T $_-$ , N $_-$ > operator/ (T $_-$ a, const LocalVect< T $_-$ , N $_-$ > &x)

Operator / (Division of vector by constant)

• template < class  $T_-$ , size  $_-$ t  $N_->$ 

real\_t Dot (const LocalVect $< T_-, N_- > \&a$ , const LocalVect $< T_-, N_- > \&b$ )

Calculate dot product of 2 vectors (instances of class LocalVect)

• template < class T\_, size\_t N\_>

void Scale ( $T_-$  a, const LocalVect<  $T_-$ ,  $N_-$  > &x, LocalVect<  $T_-$ ,  $N_-$  > &y)

Multiply vector x by constant a and store result in y.

• template < class  $T_-$  , size  $_-$ t  $N_- >$ 

```
void Scale (T_- a, LocalVect< T_-, N_- > &x)
```

Multiply vector x by constant a and store result in x.

• template < class  $T_-$ , size\_t  $N_->$ 

```
void Axpy (T_a, const LocalVect< T_a, N_a > &x, LocalVect< T_a, N_a > &y)
```

Add a\*x to vector y.

```
• template<class T_, size_t N_>
  void Copy (const LocalVect< T<sub>-</sub>, N<sub>-</sub>> &x, LocalVect< T<sub>-</sub>, N<sub>-</sub>> &y)
      Copy vector x into vector y.
• template < class T_, size_t N_>
  ostream & operator << (ostream &s, const Local Vect < T<sub>-</sub>, N<sub>-</sub> > &v)
      Output vector in output stream.
• template<class T_>
  Vect < T_- > operator* (const SkMatrix < T_- > &A, const Vect < T_- > &b)
      Operator * (Multiply vector by matrix and return resulting vector.
• template<class T_>
  ostream & operator << (ostream &s, const SkMatrix < T_> &a)
      Output matrix in output stream.
• template<class T_>
  Vect < T_- > operator* (const SkSMatrix < T_- > &A, const Vect < T_- > &b)
      Operator * (Multiply vector by matrix and return resulting vector.
• template<class T_>
  ostream & operator << (ostream &s, const SkSMatrix < T_ > &a)
      Output matrix in output stream.
• template<class T_>
  Vect < T_- > operator* (const SpMatrix < T_- > &A, const Vect < T_- > &b)
      Operator * (Multiply vector by matrix and return resulting vector.
• template<class T_>
  ostream & operator << (ostream &s, const SpMatrix < T_> &A)
      Output matrix in output stream.
• template<class T_>
  Vect < T_- > operator* (const TrMatrix < T_- > &A, const Vect < T_- > &b)
      Operator * (Multiply vector by matrix and return resulting vector.
 template<class T_>
  TrMatrix < T_- > operator* (T_- a, const TrMatrix < T_- > &A)
      Operator * (Premultiplication of matrix by constant)
• template<class T_>
  ostream & operator << (ostream &s, const TrMatrix < T<sub>-</sub> > &A)
      Output matrix in output stream.
• template<class T_>
  Vect < T_- > operator + (const Vect < T_- > &x, const Vect < T_- > &y)
      Operator + (Addition of two instances of class Vect)

    template < class T_>

  Vect< T_-> operator- (const Vect< T_-> &x, const Vect< T_-> &y)
      Operator - (Difference between two vectors of class Vect)
• template<class T_>
  Vect < T_- > operator* (const T_- &a, const Vect < T_- > &x)
      Operator * (Premultiplication of vector by constant)
  template<class T_>
  Vect < T_- > operator* (const Vect < T_- > &x, const T_- &a)
      Operator * (Postmultiplication of vector by constant)

    template<class T<sub>−</sub>>

  Vect < T_- > operator/ (const Vect < T_- > &x, const T_- &a)
      Operator / (Divide vector entries by constant)
```

```
• template < class T_- > T_- Dot (const Vect < T_- > &x, const Vect < T_- > &y)
```

Calculate dot product of two vectors.

void Modulus (const Vect< complex\_t > &x, Vect< real\_t > &y)

Calculate modulus of complex vector.

• void Real (const Vect< complex\_t > &x, Vect< real\_t > &y)

Calculate real part of complex vector.

void Imag (const Vect< complex\_t > &x, Vect< real\_t > &y)

Calculate imaginary part of complex vector.

template<class T<sub>-</sub>>
 istream & operator>> (istream &s, Vect< T<sub>-</sub>> &v)

template < class T\_ > ostream & operator << (ostream &s, const Vect < T\_ > &v)

Output vector in output stream.

• real\_t operator\* (const vector< real\_t > &x, const vector< real\_t > &y)

*Operator* \* (*Dot product of 2 vector instances*)

### **Friends**

template<class TT\_>
 ostream & operator<< (ostream &s, const SpMatrix< TT\_> &A)

# 5.12.1 Detailed Description

Vector and matrix classes.

### 5.12.2 Function Documentation

```
Vect< T_-> operator* ( const BMatrix< T_-> & A, const Vect< T_-> & b )
```

Operator \* (Multiply vector by matrix and return resulting vector.

### Parameters

in	A	BMatrix instance to multiply by vector
in	b	Vect instance

### Returns

Vect instance containing A\*b

```
BMatrix< T_- > operator* ( T_- a_r const BMatrix< T_- > & A )
```

Operator \* (Premultiplication of matrix by constant)

Returns

a\*A

Vect< 
$$T_-$$
 > operator\* ( const DMatrix<  $T_-$  > &  $A$ , const Vect<  $T_-$  > &  $b$ )

Operator \* (Multiply vector by matrix and return resulting vector.

in	Α	DMatrix instance to multiply by vector
in	b	Vect instance

#### Returns

**Vect** instance containing A\*b

# Vect< $T_-$ > operator\* ( const DSMatrix< $T_-$ > & A, const Vect< $T_-$ > & b)

Operator \* (Multiply vector by matrix and return resulting vector.

### **Parameters**

in	A	DSMatrix instance to multiply by vector
in	b	Vect instance

#### Returns

Vect instance containing A\*b

# LocalMatrix< T<sub>-</sub>, NR<sub>-</sub>, NC<sub>-</sub>> operator\* ( T<sub>-</sub> a, const LocalMatrix< T<sub>-</sub>, NR<sub>-</sub>, NC<sub>-</sub>> & x )

Operator \* (Multiply matrix x by scalar a)

Returns

a\*x

Operator \* (Multiply matrix A by vector x)

This function performs a matrix-vector product and returns resulting vector as a reference to LocalVect instance

Returns

 $\mathtt{A} {*} \mathtt{x}$ 

### LocalMatrix< T<sub>-</sub>, NR<sub>-</sub>, NC<sub>-</sub>> operator/ ( T<sub>-</sub> a, const LocalMatrix< T<sub>-</sub>, NR<sub>-</sub>, NC<sub>-</sub>> & x )

Operator / (Divide matrix x by scalar a)

Returns

x/a

```
LocalMatrix< T_-, NR_-, NC_-> operator+ ( const LocalMatrix< T_-, NR_-, NC_-> \& x, const
LocalMatrix< T<sub>-</sub>, NR<sub>-</sub>, NC<sub>-</sub>> & y )
Operator + (Add matrix x to y)
Returns
      x+y
LocalMatrix< T<sub>-</sub>, NR<sub>-</sub>, NC<sub>-</sub>> operator- ( const LocalMatrix< T<sub>-</sub>, NR<sub>-</sub>, NC<sub>-</sub>> & x, const
LocalMatrix< T_-, NR_-, NC_- > \& y)
Operator – (Subtract matrix y from x)
Returns
      х-у
LocalVect< T<sub>-</sub>, N<sub>-</sub>> operator+ ( const LocalVect< T<sub>-</sub>, N<sub>-</sub>> & x, const LocalVect< T<sub>-</sub>, N<sub>-</sub>> &
Operator + (Add two vectors)
Returns
      x+y
Local Vect < T_-, N_- > operator - (\ const\ Local Vect < T_-, N_- > \&\ x,\ const\ Local Vect < T_-, N_- > \&\ x,
y )
Operator - (Subtract two vectors)
Returns
      х-у
LocalVect< T_-, N_-> operator* ( T_- a, const LocalVect< T_-, N_-> & <math>x )
Operator * (Premultiplication of vector by constant)
Returns
      a*x
LocalVect< T<sub>-</sub>, N<sub>-</sub>> operator/ ( T<sub>-</sub> a, const LocalVect< T<sub>-</sub>, N<sub>-</sub>> & x )
Operator / (Division of vector by constant)
Returns
      x/a
```

double Dot ( const LocalVect< T\_, N\_ > & a, const LocalVect< T\_, N\_ > & b )

Calculate dot product of 2 vectors (instances of class LocalVect)

Returns

Dot product

Vect<  $T_-$  > operator\* ( const SkMatrix<  $T_-$  > & A, const Vect<  $T_-$  > & b)

Operator \* (Multiply vector by matrix and return resulting vector.

### **Parameters**

in	Α	SkMatrix instance to multiply by vector
in	b	Vect instance

### Returns

Vect instance containing A\*b

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

ostream & operator << ( ostream & s, const SkMatrix < T<sub>-</sub> > & a)

Output matrix in output stream.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

Author

Rachid Touzani

Copyright

GNU Lesser Public License

Vect<  $T_->$  operator\* ( const SkSMatrix<  $T_->$  & A, const Vect<  $T_->$  & b)

Operator \* (Multiply vector by matrix and return resulting vector.

in	A	SkSMatrix instance to multiply by vector
in	b	Vect instance

### Returns

**Vect** instance containing A\*b

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# ostream & operator << ( ostream & s, const SkSMatrix < T $_->$ & a)

Output matrix in output stream.

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# Vect< $T_- >$ operator\* ( const SpMatrix< $T_- > & A$ , const Vect< $T_- > & b$ )

Operator \* (Multiply vector by matrix and return resulting vector.

### Parameters

in	A	SpMatrix instance to multiply by vector
in	b	Vect instance

# Returns

**Vect** instance containing A\*b

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

ostream & operator << ( ostream & s, const SpMatrix < T<sub>-</sub> > & A )

Output matrix in output stream.

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

Vect<  $T_- >$  operator\* ( const TrMatrix<  $T_- > \& A$ , const Vect<  $T_- > \& b$ )

Operator \* (Multiply vector by matrix and return resulting vector.

### Parameters

in	Α	TrMatrix instance to multiply by vector
in	b	Vect instance

### Returns

**Vect** instance containing A\*b

Author

Rachid Touzani

Copyright

GNU Lesser Public License

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

Operator \* (Premultiplication of matrix by constant)

Returns

a\*A

Author

Rachid Touzani

Copyright

GNU Lesser Public License

```
ostream & operator << ( ostream & s, const TrMatrix < T<sub>-</sub> > & a )
Output matrix in output stream.
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
Vect< T_- > operator+ ( const Vect< T_- > & x, const Vect< T_- > & y)
Operator + (Addition of two instances of class Vect)
Returns
     x + y
Vect< T_- > operator- ( const Vect< T_- > & x, const Vect< T_- > & y)
Operator - (Difference between two vectors of class Vect)
Returns
     х - у
Vect< T_-> operator* ( const T_- & a, const Vect< T_-> & x)
Operator * (Premultiplication of vector by constant)
Returns
     a*x
Vect< T_- > operator* ( const Vect< T_- > & x, const T_- & a)
Operator * (Postmultiplication of vector by constant)
Returns
     x*a
Vect< T_- > operator/ ( const Vect< T_- > & x, const T_- & a)
Operator / (Divide vector entries by constant)
Returns
     x/a
```

### $T_-$ Dot ( const Vect< $T_-$ > & x, const Vect< $T_-$ > & y)

Calculate dot product of two vectors.

### Returns

Dot (inner or scalar) product Calculate dot (scalar) product of two vectors

### void Modulus ( const Vect< complex\_t > & x, Vect< real\_t > & y)

Calculate modulus of complex vector.

### **Parameters**

in	x	Vector with complex value entries
out	y	Vector containing moduli of entries of x

# void Real ( const Vect< complex\_t > & x, Vect< real\_t > & y)

Calculate real part of complex vector.

#### **Parameters**

in	x	Vector with complex value entries
out	y	Vector containing real parts of entries of x

# void Imag ( const Vect< complex\_t > & x, Vect< real\_t > & y)

Calculate imaginary part of complex vector.

### Parameters

in	x	Vector with complex value entries
out	y	Vector containing imaginary parts of entries of x

### istream & operator>> ( istream & s, Vect< $T_-$ > & a)

Read vector from input stream

# ostream & operator<< ( ostream & s, const Vect< $T_-$ > & v )

Output vector in output stream.

Level of vector output depends on the global variable Verbosity

- If Verbosity=0, this function outputs vector size only.
- If Verbosity>0, this function outputs vector size, vector name, value of time, and number of components
- If Verbosity>1, this function outputs in addition the first 10 entries in vector

- If Verbosity>2, this function outputs in addition the first 50 entries in vector
- If Verbosity>3, this function outputs in addition the first 100 entries in vector
- If Verbosity>4, this function outputs all vector entries

### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

# real\_t operator\* ( const vector< real\_t > & x, const vector< real\_t > & y)

Operator \* (Dot product of 2 vector instances)

Returns

x.y

### **5.12.3** Friends

ostream & operator << ( ostream & s, const SpMatrix <  $TT_->$  & A ) [friend]

Output matrix in output stream

# 5.13 Physical properties of media

Physical properties of materials and media.

### Classes

• class Material

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

# 5.13.1 Detailed Description

Physical properties of materials and media.

# 5.14 Global Variables

All global variables in the library.

### **Variables**

• Node \* theNode

A pointer to Node.

• Element \* the Element

A pointer to Element.

• Side \* theSide

A pointer to Side.

• Edge \* theEdge

A pointer to Edge.

• int Verbosity

Verbosity parameter.

• int theStep

Time step counter.

int theIteration

Iteration counter.

• int NbTimeSteps

Number of time steps.

• int MaxNbIterations

Maximal number of iterations.

real\_t theTimeStep

Time step label.

• real\_t theTime

Time value.

• real\_t theFinalTime

Final time value.

• real\_t theTolerance

Tolerance value for convergence.

• real\_t theDiscrepancy

Value of discrepancy for an iterative procedure Its default value is 1.0.

• bool Converged

Boolean variable to say if an iterative procedure has converged.

• bool InitPetsc

# 5.14.1 Detailed Description

All global variables in the library.

### 5.14.2 Variable Documentation

### Node\* theNode

A pointer to Node.

Useful for loops on nodes

### **Element\*** the Element

A pointer to Element.

Useful for loops on elements

### Side\* theSide

A pointer to Side.

Useful for loops on sides

### Edge\* theEdge

A pointer to Edge.

Useful for loops on edges

### int Verbosity

Verbosity parameter.

Parameter for verbosity of message outputting.

The value of Verbosity can be modified anywhere in the calling programs. It allows outputting messages in function of the used class or function. To see how this parameter is used in any class, the OFELI user has to read corresponding documentation.

Its default value is 1

### int theStep

Time step counter.

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

# Remarks

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

### int the Iteration

Iteration counter.

This counter must be initialized by the user

### Remarks

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

### int NbTimeSteps

Number of time steps.

### Remarks

May be used in conjunction with the macro TimeLoop.

### int MaxNbIterations

Maximal number of iterations.

Remarks

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

### real\_t theTimeStep

Time step label.

Remarks

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

### real\_t theTime

Time value.

Remarks

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

### real\_t theFinalTime

Final time value.

Remarks

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

# real\_t theTolerance

Tolerance value for convergence.

Remarks

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

### bool Converged

Boolean variable to say if an iterative procedure has converged. Its default value is false

### bool InitPetsc

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

# 5.15 Finite Element Mesh

### Mesh management classes

#### Classes

• class Domain

To store and treat finite element geometric information.

• class Edge

To describe an edge.

• class Element

To store and treat finite element geometric information.

class Figure

To store and treat a figure (or shape) information.

• class Rectangle

To store and treat a rectangular figure.

class Brick

To store and treat a brick (parallelepiped) figure.

• class Circle

To store and treat a circular figure.

• class Sphere

To store and treat a sphere.

• class Ellipse

To store and treat an ellipsoidal figure.

• class Triangle

To store and treat a triangle.

class Polygon

To store and treat a polygonal figure.

• class Grid

To manipulate structured grids.

• class Mesh

 $To \ store \ and \ manipulate \ finite \ element \ meshes.$ 

class MeshAdapt

To adapt mesh in function of given solution.

class NodeList

Class to construct a list of nodes having some common properties.

class ElementList

Class to construct a list of elements having some common properties.

class SideList

Class to construct a list of sides having some common properties.

class EdgeList

Class to construct a list of edges having some common properties.

• class Node

To describe a node.

• class Partition

To partition a finite element mesh into balanced submeshes.

• class Side

To store and treat finite element sides (edges in 2-D or faces in 3-D)

### **Macros**

• #define GRAPH\_MEMORY 1000000

Memory necessary to store matrix graph.

• #define MAX\_NB\_ELEMENTS 10000

Maximal Number of elements.

• #define MAX\_NB\_NODES 10000

Maximal number of nodes.

• #define MAX\_NB\_SIDES 30000

Maximal number of sides in.

• #define MAX\_NB\_EDGES 30000

Maximal Number of edges.

• #define MAX\_NBDOF\_NODE 6

Maximum number of DOF supported by each node.

• #define MAX\_NBDOF\_SIDE 6

Maximum number of DOF supported by each side.

• #define MAX\_NBDOF\_EDGE 2

Maximum number of DOF supported by each edge.

• #define MAX\_NB\_ELEMENT\_NODES 20

Maximum number of nodes by element.

• #define MAX\_NB\_ELEMENT\_EDGES 10

Maximum number of edges by element.

• #define MAX\_NB\_SIDE\_NODES 9

Maximum number of nodes by side.

• #define MAX\_NB\_ELEMENT\_SIDES 8

Maximum number of sides by element.

#define MAX\_NB\_ELEMENT\_DOF 27

Maximum number of dof by element.

• #define MAX\_NB\_SIDE\_DOF 4

 $Maximum\ number\ of\ dof\ by\ side.$ 

• #define MAX\_NB\_INT\_PTS 20

Maximum number of integration points in element.

• #define MAX\_NB\_MATERIALS 10

Maximum number of materials.

- #define TheNode (\*theNode)
- #define TheElement (\*theElement)
- #define TheSide (\*theSide)
- #define TheEdge (\*theEdge)
- #define ElementLoop(m) for ((m).topElement(); (theElement=(m).getElement());)
- #define ActiveElementLoop(m) for ((m).topElement(); (theElement=(m).getActiveElement());)
- #define SideLoop(m) for ((m).topSide(); (theSide=(m).getSide());)
- #define EdgeLoop(m) for ((m).topEdge(); (theEdge=(m).getEdge());)
- #define NodeLoop(m) for ((m).topNode(); (theNode=(m).getNode());)
- #define BoundaryNodeLoop(m) for ((m).topBoundaryNode(); (theNode=(m).getBoundary → Node());)
- #define BoundarySideLoop(m) for ((m).topBoundarySide(); (theSide=(m).getBoundary ⇔ Side());)
- #define theNodeLabel theNode->n()

#define theSideLabel theSide->n()

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

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

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

• #define theElementLabel theElement->n()

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

• #define the Element Node Label (i) the Element -> get Node Label (i)

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

### **Functions**

• ostream & operator<< (ostream &s, const Edge &ed)

Output edge data.

• ostream & operator<< (ostream &s, const Element &el)

Output element data.

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

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

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

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

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

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

• ostream & operator << (ostream &s, const Material &m)

Output material data.

• ostream & operator << (ostream &s, const Mesh &ms)

Output mesh data.

• ostream & operator << (ostream &s, const MeshAdapt &a)

Output MeshAdapt class data.

ostream & operator<< (ostream &s, const NodeList &nl)</li>

Output NodeList instance.

• ostream & operator<< (ostream &s, const ElementList &el)

Output ElementList instance.

• ostream & operator<< (ostream &s, const SideList &sl)

Output SideList instance.

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

Output EdgeList instance.

• size\_t Label (const Node &nd)

Return label of a given node.

size\_t Label (const Element &el)

Return label of a given element.

• size\_t Label (const Side &sd)

Return label of a given side.

size\_t Label (const Edge &ed)

Return label of a given edge.

• size\_t NodeLabel (const Element &el, size\_t n)

Return global label of node local label in element.

• size\_t NodeLabel (const Side &sd, size\_t n)

Return global label of node local label in side.

• Point< real\_t > Coord (const Node &nd)

Return coordinates of a given node.

• int Code (const Node &nd, size\_t i=1)

Return code of a given (degree of freedom of) node.

• int Code (const Element &el)

Return code of a given element.

• int Code (const Side &sd, size\_t i=1)

Return code of a given (degree of freedom of) side.

• bool operator== (const Element &el1, const Element &el2)

Check equality between 2 elements.

• bool operator== (const Side &sd1, const Side &sd2)

Check equality between 2 sides.

• void DeformMesh (Mesh &mesh, const Vect< real\_t > &u, real\_t a=1)

Calculate deformed mesh using a displacement field.

void MeshToMesh (Mesh &m1, Mesh &m2, const Vect< real\_t > &u1, Vect< real\_t > &u2, size\_t nx, size\_t ny=0, size\_t nz=0, size\_t dof=1)

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

• void MeshToMesh (const Vect< real\_t > &u1, Vect< real\_t > &u2, size\_t nx, size\_t ny=0, size\_t nz=0, size\_t dof=1)

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

void MeshToMesh (Mesh &m1, Mesh &m2, const Vect< real\_t > &u1, Vect< real\_t > &u2, const Point< real\_t > &xmin, const Point< real\_t > &xmax, size\_t nx, size\_t ny, size\_t nz, size\_t dof=1)

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

• real\_t getMaxSize (const Mesh &m)

Return maximal size of element edges for given mesh.

• real\_t getMinSize (const Mesh &m)

Return minimal size of element edges for given mesh.

real\_t getMinElementMeasure (const Mesh &m)

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

real\_t getMaxElementMeasure (const Mesh &m)

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

• real\_t getMinSideMeasure (const Mesh &m)

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

real\_t getMaxSideMeasure (const Mesh &m)

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

• real\_t getMeanElementMeasure (const Mesh &m)

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

real\_t getMeanSideMeasure (const Mesh &m)

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

• void setNodeCodes (Mesh &m, const string &exp, int code, size\_t dof=1)

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

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

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

• void setSideCodes (Mesh &m, const string &exp, int code, size\_t dof=1)

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

• void setBoundarySideCodes (Mesh &m, const string &exp, int code, size\_t dof=1)

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

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

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

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

Say if a given node belongs to a given element.

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

Say if a given node belongs to a given side.

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

Say if a given side belongs to a given element.

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

Output node data.

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

Output side data.

# 5.15.1 Detailed Description

Mesh management classes

# 5.15.2 Macro Definition Documentation

#### #define GRAPH\_MEMORY 1000000

Memory necessary to store matrix graph.

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

### #define TheNode (\*theNode)

A macro that gives the instance pointed by the Node

### #define The Element (\*the Element)

A macro that gives the instance pointed by the Element

### #define TheSide (\*theSide)

A macro that gives the instance pointed by the Side

# #define TheEdge (\*theEdge)

A macro that gives the instance pointed by *theEdge* 

# #define ElementLoop( m ) for ((m).topElement(); (theElement=(m).getElement());)

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

Note

: Each iteration updates the pointer the Element to current Element

# #define ActiveElementLoop(m) for ((m).topElement(); (theElement=(m).getActive $\leftarrow$ Element());)

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

Note

: Each iteration updates the pointer the Element to current Element

: This macro is necessary only if adaptive meshing is used

### #define SideLoop( m ) for ((m).topSide(); (theSide=(m).getSide());)

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

Note

: Each iteration updates the pointer the Side to current Element

# #define EdgeLoop( m ) for ((m).topEdge(); (theEdge=(m).getEdge());)

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

Note

: Each iteration updates the pointer the Edge to current Edge

### #define NodeLoop( m ) for ((m).topNode(); (theNode=(m).getNode());)

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

Note

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

# #define BoundaryNodeLoop( m ) for ((m).topBoundaryNode(); (theNode=(m).getBoundaryNode());)

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

Note

: Each iteration updates the pointer the Node to current Node

# #define BoundarySideLoop(m) for ((m).topBoundarySide(); (theSide=(m).getBoundary $\leftrightarrow$ Side());)

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

Note

: Each iteration updates the pointer the Side to current Node

### #define the Node Label the Node -> n()

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

# 5.15.3 Function Documentation

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

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

#### **Parameters**

in	f1	First Figure instance
in	f2	Second Figure instance

#### Returns

Updated resulting Figure instance

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

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

#### **Parameters**

in	f1	First Figure instance
in	<i>f</i> 2	Second Figure instance

### Returns

Updated resulting Figure instance

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

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

### Parameters

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

### Returns

Updated resulting Figure instance

# ostream & operator << ( ostream & s, const Mesh & ms )

Output mesh data.

Level of mesh output depends on the global variable Verbosity

- If Verbosity=0 or Verbosity=1, this function outputs only principal mesh parameters: number of nodes, number of elements, ...
- If Verbosity>1, this function outputs in addition the list of 10 first nodes, elements and sides

- If Verbosity>2, this function outputs in addition the list of 50 first nodes, elements and sides
- If Verbosity>3, this function outputs all mesh data

```
ostream & operator << ( ostream & s, const NodeList & nl )
Output NodeList instance.
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
ostream & operator<< ( ostream & s, const ElementList & el )
Output ElementList instance.
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
ostream & operator << ( ostream & s, const SideList & sl )
Output SideList instance.
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
ostream & operator << ( ostream & s, const EdgeList & el )
Output EdgeList instance.
Author
     Rachid Touzani
Copyright
     GNU Lesser Public License
size_t Label (const Node & nd)
Return label of a given node.
```

in *nd* Reference to Node instance

Returns

Label of node

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# size\_t Label ( const Element & el )

Return label of a given element.

### **Parameters**

in	el	Reference to Element instance
----	----	-------------------------------

### Returns

Label of element

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# size\_t Label ( const Side & sd )

Return label of a given side.

### Parameters

in	sd	Reference to Side instance

Returns

Label of side

Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# size\_t Label ( const Edge & ed )

Return label of a given edge.

### Parameters

in	ed	Reference to Edge instance
----	----	----------------------------

# Returns

Label of edge

### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

# size\_t NodeLabel ( const Element & el, size\_t n )

Return global label of node local label in element.

# Parameters

in	el	Reference to Element instance
in	n	Local label of node in element

### Returns

Global label of node

# Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# size\_t NodeLabel ( const Side & sd, size\_t n )

Return global label of node local label in side.

in	sd	Reference to Side instance
in	n	Local label of node in side

### Returns

Global label of node

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# Point < real\_t > Coord ( const Node & nd )

Return coordinates of a given node.

### **Parameters**

in	nd	Reference to Node instance
----	----	----------------------------

### Returns

Coordinates of node

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# int Code ( const Node & nd, size\_t i = 1 )

Return code of a given (degree of freedom of) node.

### Parameters

in	nd	Reference to Node instance
in	i	Label of dof [Default: 1]

### Returns

Code of dof of node

Author

Rachid Touzani

Copyright

GNU Lesser Public License

### int Code (const Element & el)

Return code of a given element.

### Parameters

in	el	Reference to Element instance
----	----	-------------------------------

Returns

Code of element

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# int Code ( const Side & sd, size\_t i = 1 )

Return code of a given (degree of freedom of) side.

### Parameters

in	sd	Reference to Side instance
in	i	Label of dof [Default: 1]

Returns

Code of dof of side

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# operator== ( const Element & el1, const Element & el2 )

Check equality between 2 elements.

in	el1	Reference to first Side instance
in	el2	Reference to second Side instance

### Returns

true is elements are equal, *i.e.* if they have the same nodes, false if not.

### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

# bool operator== ( const Side & sd1, const Side & sd2 )

Check equality between 2 sides.

### Parameters

in	sd1	Reference to first Side instance
in	sd2	Reference to second Side instance

### Returns

true is sides are equal, *i.e.* if they have the same nodes, false if not.

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# void DeformMesh ( Mesh & mesh, const Vect < real\_t > & u, real\_t a = 1 )

Calculate deformed mesh using a displacement field.

### **Parameters**

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

#### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

void MeshToMesh ( Mesh & m1, Mesh & m2, const Vect< real\_t > & u1, Vect< real\_t > & u2, size\_t nx, size\_t ny = 0, size\_t nz = 0.

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

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

#### Remarks

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

#### **Parameters**

in	m1	Reference to the first mesh instance
out	m2	Reference to the second mesh instance
in	и1	Input vector of nodal values defined on first mesh
out	и2	Output vector of nodal values defined on second mesh
in	пх	Number of cells in the x-direction in the fine structured grid
in	ny	Number of cells in the y-direction in the fine structured grid The default value of ny is 0, i.e. a 1-D grid
in	nz	Number of cells in the z-direction in the fine structured grid The default value of nz is 0, i.e. a 1-D or 2-D grid
in	dof	Label of degree of freedom of vector u. Only this dof is considered. [Default: 1]

### Note

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

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

void MeshToMesh ( const Vect< real\_t > & u1, Vect< real\_t > & u2, size\_t nx, size\_t ny = 0, size\_t nz = 0, size\_t

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

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

#### Remarks

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

#### **Parameters**

in	и1	Input vector of nodal values defined on first mesh. This vector instance must contain Mesh instance
out	и2	Output vector of nodal values defined on second mesh. This vector instance must contain Mesh instance
in	пх	Number of cells in the x-direction in the fine structured grid
in	пу	Number of cells in the y-direction in the fine structured grid The default value of ny is 0, i.e. a 1-D grid
in	nz	Number of cells in the z-direction in the fine structured grid The default value of nz is 0, i.e. a 1-D or 2-D grid
in	dof	Label of degree of freedom of vector u. Only this dof is considered. [Default: 1]

#### Note

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

#### Author

Rachid Touzani

### Copyright

GNU Lesser Public License

void MeshToMesh ( Mesh & m1, Mesh & m2, const Vect< real\_t > & u1, Vect< real\_t > & u2, const Point< real\_t > & xmin, const Point< real\_t > & xmax, size\_t nx, size\_t nx,

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

The program interpolates (piecewise linear) first the vector on a finer structured grid. Then the values on the new mesh nodes are computed. In this function the grid rectangle is defined so that this one can cover only a submesh of  $\mathtt{m}1$ .

### Remarks

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

### Parameters

in	m1	Reference to the first mesh instance	
out	m2	Reference to the second mesh instance	
in	u1 Input vector of nodal values defined on first mesh		

out	и2	Output vector of nodal values defined on second mesh	
in	xmin	Point instance containing minimal coordinates of the rectangle that defines the grid	
in	xmax	Point instance containing maximal coordinates of the rectangle that defines the grid	
in	пх	Number of cells in the x-direction in the fine structured grid	
in	пу	Number of cells in the y-direction in the fine structured grid The default value of ny is 0, i.e. a 1-D grid	
in	nz	Number of cells in the z-direction in the fine structured grid The default value of nz is 0, i.e. a 1-D or 2-D grid	
in	dof	Label of degree of freedom of vector u. Only this dof is considered. [Default: 1]	

### Note

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

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# real\_t getMaxSize ( const Mesh & m )

Return maximal size of element edges for given mesh.

# Parameters

in	m	Reference to mesh instance

# Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# real\_t getMinSize ( const Mesh & m )

Return minimal size of element edges for given mesh.

# Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# real\_t getMinElementMeasure ( const Mesh & m )

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

### Parameters

in <i>m</i> Reference to mesh instar	ice
--------------------------------------	-----

### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

# real\_t getMaxElementMeasure ( const Mesh & m )

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

### Parameters

in	m	Reference to mesh instance
----	---	----------------------------

### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

# real\_t getMinSideMeasure ( const Mesh & m )

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

### Parameters

in	m	Reference to mesh instance
----	---	----------------------------

Note

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

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

# real\_t getMaxSideMeasure ( const Mesh & m )

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

#### **Parameters**

	in	m	Reference to mesh instance
--	----	---	----------------------------

Note

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

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# real\_t getMeanElementMeasure ( const Mesh & m )

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

### Parameters

in	m	Reference to mesh instance

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# real\_t getMeanSideMeasure ( const Mesh & m )

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

in	m	Reference to mesh instance
----	---	----------------------------

### Note

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

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# void setNodeCodes (Mesh & m, const string & exp, int code, size\_t dof = 1)

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

### **Parameters**

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

# void setBoundaryNodeCodes ( Mesh & m, const string & exp, int code, size\_t dof = 1 )

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

# Parameters

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

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

## void setSideCodes ( Mesh & m, const string & exp, int code, size\_t dof = 1 )

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

### **Parameters**

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

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

## void setBoundarySideCodes (Mesh & m, const string & exp, int code, size\_t dof = 1)

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

### **Parameters**

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

### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

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

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

### Parameters

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

104 OFELI's Reference Guide

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

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

Say if a given node belongs to a given element.

### Parameters

in	nd	Pointer to Node
in	el	Pointer to Element

### Returns

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

### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

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

Say if a given node belongs to a given side.

### Parameters

in	nd	Pointer to Node
in	sd	Pointer to Side

### Returns

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

### Author

Rachid Touzani

## Copyright

GNU Lesser Public License

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

Say if a given side belongs to a given element.

### Parameters

in	sd	Pointer to Side
in	el	Pointer to Element

### Returns

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

Author

Rachid Touzani

Copyright

GNU Lesser Public License

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

Output node data.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

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

Output side data.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# 5.16 Shape Function

Shape function classes.

#### Classes

• class FEShape

Parent class from which inherit all finite element shape classes.

• class triangle

Defines a triangle. The reference element is the rectangle triangle with two unit edges.

• class Hexa8

Defines a three-dimensional 8-node hexahedral finite element using Q1-isoparametric interpolation.

• class Line2

To describe a 2-Node planar line finite element.

• class Line3

To describe a 3-Node quadratic planar line finite element.

• class Penta6

Defines a 6-node pentahedral finite element using  $P_1$  interpolation in local coordinates (s.x, s.y) and  $Q_1$  isoparametric interpolation in local coordinates (s.x, s.z) and (s.y, s.z).

class Quad4

Defines a 4-node quadrilateral finite element using  $Q_1$  isoparametric interpolation.

• class Tetra4

Defines a three-dimensional 4-node tetrahedral finite element using  $P_1$  interpolation.

• class Triang3

Defines a 3-Node  $(P_1)$  triangle.

• class Triang6S

Defines a 6-Node straight triangular finite element using P<sub>2</sub> interpolation.

# 5.16.1 Detailed Description

Shape function classes.

# 5.17 Solver

Solver functions and classes.

#### Classes

• class Reconstruction

*To perform various reconstruction operations.* 

• class EigenProblemSolver

Class to find eigenvalues and corresponding eigenvectors of a given matrix in a generalized eigenproblem, i.e. Find scalars l and non-null vectors v such that  $[K]\{v\} = l[M]\{v\}$  where [K] and [M] are symmetric matrices. The eigenproblem can be originated from a PDE. For this, we will refer to the matrices K and M as Stiffness and Mass matrices respectively.

class Integration

Class for numerical integration methods.

• class Iter< T\_>

Class to drive an iterative process.

• class LinearSolver< T\_>

Class to solve systems of linear equations by iterative methods.

class MyNLAS

Abstract class to define by user specified function.

class MyOpt

Abstract class to define by user specified optimization function.

• class NLASSolver

To solve a system of nonlinear algebraic equations of the form f(u) = 0.

class ODESolver

To solve a system of ordinary differential equations.

class OptSolver

To solve an optimization problem with bound constraints.

• class Prec< T\_>

To set a preconditioner.

• class TimeStepping

To solve time stepping problems, i.e. systems of linear ordinary differential equations of the form  $[A2]\{y''\}+[A1]\{y'\}+[A0]\{y\}=\{b\}$ .

## **Macros**

• #define MAX\_NB\_EQUATIONS 5

Maximum number of equations.

#define MAX\_NB\_INPUT\_FIELDS 3

Maximum number of fields for an equation.

• #define MAX\_NB\_MESHES 10

Maximum number of meshes.

• #define TIME\_LOOP(ts, t, ft, n)

A macro to loop on time steps to integrate on time ts: Time step t: Initial time value updated at each time step ft: Final time value n: Time step index.

• #define TimeLoop

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

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

A macro to loop on iterations for an iterative procedure.

### **Functions**

• ostream & operator << (ostream &s, const Muscl3DT &m)

Output mesh data as calculated in class Muscl3DT.

• template<class T\_>

int BiCG (const SpMatrix< T $_-$  > &A, const Prec< T $_-$  > &P, const Vect< T $_-$  > &b, Vect< T $_-$  > &x, int max\_it, real\_t &toler)

Biconjugate gradient solver function.

• template<class T\_>

int BiCG (const SpMatrix $< T_- > &A$ , int prec, const Vect $< T_- > &b$ , Vect $< T_- > &x$ , int max\_it, real\_t toler)

Biconjugate gradient solver function.

• template<class T\_>

int BiCGStab (const SpMatrix  $< T_- > &A$ , const Prec $< T_- > &P$ , const Vect $< T_- > &b$ , Vect $< T_- > &x$ , int max\_it, real\_t toler)

Biconjugate gradient stabilized solver function.

• template<class T\_>

int BiCGStab (const SpMatrix<  $T_->$  &A, int prec, const Vect<  $T_->$  &b, Vect<  $T_->$  &x, int max\_it, real\_t toler)

Biconjugate gradient stabilized solver function.

• template<class T\_>

int CG (const SpMatrix<  $T_->$  &A, const Prec<  $T_->$  &P, const Vect<  $T_->$  &b, Vect<  $T_->$  &x, int max\_it, real\_t toler)

Conjugate gradient solver function.

• template<class T\_>

int CG (const SpMatrix<  $T_->$  &A, int prec, const Vect<  $T_->$  &b, Vect<  $T_->$  &x, int max\_it, real\_t toler)

Conjugate gradient solver function.

template<class T₋>

int CGS (const SpMatrix<  $T_->$  &A, const Prec<  $T_->$  &P, const Vect<  $T_->$  &b, Vect<  $T_->$  &x, int max\_it, real\_t toler)

Conjugate Gradient Squared solver function.

• template<class T\_>

int CGS (const SpMatrix $< T_- > &A$ , int prec, const Vect $< T_- > &b$ , Vect $< T_- > &x$ , int max\_it, real\_t toler)

Conjugate Gradient Squared solver function.

• ostream & operator<< (ostream &s, const EigenProblemSolver &es)

Output eigenproblem information.

• template<class T\_>

int GMRes (const SpMatrix< T $_-$  > &A, const Prec< T $_-$  > &P, const Vect< T $_-$  > &b, Vect< T $_-$  > &x, size $_-$ t m, int max $_-$ it, real $_-$ t toler)

GMRes solver function.

• template<class T\_>

int GMRes (const SpMatrix<  $T_->$  &A, int prec, const Vect<  $T_->$  &b, Vect<  $T_->$  &x, size\_t m, int max\_it, real\_t toler)

GMRes solver function.

• template<class T\_>

int GS (const SpMatrix<  $T_-$  > &A, const Vect<  $T_-$  > &b, Vect<  $T_-$  > &x, real\_t omega, int max\_it, real\_t toler)

Gauss-Seidel solver function.

template < class T\_ >
 int Jacobi (const SpMatrix < T\_ > &A, const Vect < T\_ > &b, Vect < T\_ > &x, real\_t omega, int max\_it, real\_t toler)

Jacobi solver function.

• ostream & operator << (ostream &s, const NLASSolver &nl)

Output nonlinear system information.

• ostream & operator << (ostream &s, const ODESolver &de)

Output differential system information.

• ostream & operator<< (ostream &s, const OptSolver &os)

Output differential system information.

template < class T\_, class M\_>

int Richardson (const  $M_-$  &A, const Vect<  $T_-$  > &b, Vect<  $T_-$  > &x, real\_t omega, int max\_it, real\_t toler, int verbose)

Richardson solver function.

• template<class T\_>

```
void Schur (SkMatrix< T_- > &A, SpMatrix< T_- > &U, SpMatrix< T_- > &L, SpMatrix< T_- > &D, Vect< T_- > &b, Vect< T_- > &c)
```

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

• template < class  $T_-$ , class  $M_- >$ 

int SSOR (const  $M_-$  &A, const Vect<  $T_-$  > &b, Vect<  $T_-$  > &x, int max\_it, real\_t toler) SSOR solver function.

• ostream & operator << (ostream &s, TimeStepping &ts)

Output differential system information.

## 5.17.1 Detailed Description

Solver functions and classes.

### 5.17.2 Macro Definition Documentation

### #define MAX\_NB\_EQUATIONS 5

Maximum number of equations. Useful for coupled problems

## #define MAX\_NB\_INPUT\_FIELDS 3

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

#### #define MAX\_NB\_MESHES 10

Maximum number of meshes.
Useful for coupled problems

## #define TIME\_LOOP( ts, t, ft, n)

#### Value:

```
\label{eq:norm} n = 1; $$ for (real_t t=0; t<ft+0.01*ts; t+=ts, ++n)$$
```

A macro to loop on time steps to integrate on time ts: Time step t: Initial time value updated at each time step ft: Final time value n: Time step index.

### #define TimeLoop

#### Value:

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

It uses the following global variables defined in **OFELI**: the Step, the Time, the Time Step, the Final Time

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

A macro to loop on iterations for an iterative procedure.

It uses the following global variables defined in  ${\bf OFELI:}$  the Iteration, MaxNbI terations, Converged

Warning

The variable the Iteration must be zeroed before using this macro

### 5.17.3 Function Documentation

```
int BiCG ( const SpMatrix< T_- > & A, const Prec< T_- > & P, const Vect< T_- > & b, Vect< T_- > & x, int max\_it, real_t & toler)
```

Biconjugate gradient solver function.

This function uses the preconditioned Biconjugate Conjugate Gradient algorithm to solve a linear system with a sparse matrix.

The global variable Verbosity enables choosing output message level

- Verbosity < 2 : No output message
- Verbosity > 1 : Notify executing the function CG
- Verbosity > 2: Notify convergence with number of performed iterations or divergence
- Verbosity > 3 : Output each iteration number and residual
- Verbosity > 6 : Print final solution if convergence
- Verbosity > 10 : Print obtained solution at each iteration

#### **Parameters**

in	A	Problem matrix (Instance of class SpMatrix).
in	P	Preconditioner (Instance of class Prec).
in	b	Right-hand side vector (class Vect)
in,out	x	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	max↔ _it	Maximum number of iterations.
	toler	[in] Tolerance for convergence (measured in relative weighted 2-Norm).

#### Returns

Number of performed iterations,

## **Template Parameters**

< <i>T</i> ↔	Data type (double, float, complex <double>,)</double>
_>	

### Author

Rachid Touzani

## Copyright

**GNU Lesser Public License** 

# int BiCG ( const SpMatrix< $T_-$ > & A, int prec, const Vect< $T_-$ > & b, Vect< $T_-$ > & x, int $max\_it$ , real\_t toler )

Biconjugate gradient solver function.

This function uses the preconditioned Biconjugate Conjugate Gradient algorithm to solve a linear system with a sparse matrix.

The global variable Verbosity enables choosing output message level

- Verbosity < 2 : No output message
- Verbosity > 1 : Notify executing the function CG
- Verbosity > 2: Notify convergence with number of performed iterations or divergence
- Verbosity > 3 : Output each iteration number and residual
- Verbosity > 6 : Print final solution if convergence
- Verbosity > 10 : Print obtained solution at each iteration

### Parameters

in	A	Problem matrix (Instance of class SpMatrix).
in	prec	Enum variable selecting a preconditioner, among the values IDENT_PREC, DIAG_PREC, ILU_PREC or SSOR_PREC
in	b	Right-hand side vector (class Vect)
in,out	х	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	max↔ _it	Maximum number of iterations.
	toler	[in] Tolerance for convergence (measured in relative weighted 2-Norm).

#### Returns

Number of performed iterations,

### **Template Parameters**

< <i>T</i> ←	Data type (double, float, complex <double>,)</double>	]
_>		

### Author

Rachid Touzani

## Copyright

GNU Lesser Public License

# int BiCGStab ( const SpMatrix< $T_-$ > & A, const Prec< $T_-$ > & P, const Vect< $T_-$ > & b, Vect< $T_-$ > & x, int $max\_it$ , real\_t toler )

Biconjugate gradient stabilized solver function.

This function uses the preconditioned Conjugate Gradient Stabilized algorithm to solve a linear system with a sparse matrix.

The global variable Verbosity enables choosing output message level

- Verbosity < 2 : No output message
- Verbosity > 1 : Notify executing the function CG
- Verbosity > 2: Notify convergence with number of performed iterations or divergence
- Verbosity > 3 : Output each iteration number and residual
- Verbosity > 6 : Print final solution if convergence
- Verbosity > 10 : Print obtained solution at each iteration

### Parameters

in	Α	Problem matrix (Instance of class SpMatrix).
in	P	Preconditioner (Instance of class Prec).
in	b	Right-hand side vector (class Vect)
in,out	x	Vect instance containing initial solution guess on input and solution of the linear system on output (If iterations have succeeded).
in	max↔ _it	Maximum number of iterations.
in	toler	Tolerance for convergence (measured in relative weighted 2-Norm).

### Returns

< <i>T</i> ←	Data type (double, float, complex <double>,)</double>
_>	

### Author

Rachid Touzani

## Copyright

**GNU Lesser Public License** 

# int BiCGStab ( const SpMatrix< $T_-$ > & A, int prec, const Vect< $T_-$ > & b, Vect< $T_-$ > & x, int $max\_it$ , real\_t toler)

Biconjugate gradient stabilized solver function.

This function uses the preconditioned Conjugate Gradient Stabilized algorithm to solve a linear system with a sparse matrix.

The global variable Verbosity enables choosing output message level

- Verbosity < 2 : No output message
- Verbosity > 1 : Notify executing the function CG
- Verbosity > 2: Notify convergence with number of performed iterations or divergence
- Verbosity > 3 : Output each iteration number and residual
- Verbosity > 6 : Print final solution if convergence
- Verbosity > 10: Print obtained solution at each iteration

### **Parameters**

in	A	Problem matrix (Instance of class SpMatrix).
in	prec	Enum variable selecting a preconditioner, among the values IDENT_PREC, DIAG_PREC, ILU_PREC or SSOR_PREC
in	b	Right-hand side vector (class Vect)
in,out	х	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	max↔ _it	Maximum number of iterations.
in	toler	Tolerance for convergence (measured in relative weighted 2-Norm).

### Returns

< <i>T</i> ←	Data type (double, float, complex <double>,)</double>
_>	

### Author

Rachid Touzani

## Copyright

**GNU Lesser Public License** 

# int CG ( const SpMatrix< $T_-$ > & A, const Prec< $T_-$ > & P, const Vect< $T_-$ > & b, Vect< $T_-$ > & x, int $max\_it$ , real\_t toler )

Conjugate gradient solver function.

This function uses the preconditioned Conjugate Gradient algorithm to solve a linear system with a sparse matrix.

The global variable Verbosity enables choosing output message level

- Verbosity < 2 : No output message
- Verbosity > 1 : Notify executing the function CG
- Verbosity > 2: Notify convergence with number of performed iterations or divergence
- Verbosity > 3 : Output each iteration number and residual
- Verbosity > 6 : Print final solution if convergence
- Verbosity > 10: Print obtained solution at each iteration

### **Parameters**

in	A	Problem matrix (Instance of class SpMatrix).
in	P	Preconditioner (Instance of class Prec).
in	b	Right-hand side vector (class Vect)
in,out	х	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	max↔ _it	Maximum number of iterations.
in	toler	Tolerance for convergence (measured in relative weighted 2-Norm).

## Returns

< <i>T</i> ←	Data type (double, float, complex <double>,)</double>
_>	

### Author

Rachid Touzani

## Copyright

**GNU Lesser Public License** 

# int CG ( const SpMatrix< $T_-$ > & A, int prec, const Vect< $T_-$ > & b, Vect< $T_-$ > & x, int $max\_it$ , real\_t toler )

Conjugate gradient solver function.

This function uses the preconditioned Conjugate Gradient algorithm to solve a linear system with a sparse matrix.

The global variable Verbosity enables choosing output message level

- Verbosity < 2 : No output message
- Verbosity > 1 : Notify executing the function CG
- Verbosity > 2: Notify convergence with number of performed iterations or divergence
- Verbosity > 3 : Output each iteration number and residual
- Verbosity > 6 : Print final solution if convergence
- Verbosity > 10: Print obtained solution at each iteration

### **Parameters**

in	A	Problem matrix (Instance of abstract class SpMatrix).
in	prec	Enum variable selecting a preconditioner, among the values IDENT_PREC, DIAG_PREC, ILU_PREC or SSOR_PREC
in	b	Right-hand side vector (class Vect)
in,out	x	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	max↔ _it	Maximum number of iterations.
in	toler	Tolerance for convergence (measured in relative weighted 2-Norm).

### Returns

< <i>T</i> ←	Data type (double, float, complex <double>,)</double>
_>	

### Author

Rachid Touzani

## Copyright

**GNU Lesser Public License** 

int CGS ( const SpMatrix<  $T_-$  > & A, const Prec<  $T_-$  > & P, const Vect<  $T_-$  > & b, Vect<  $T_-$  > & x, int  $max\_it$ , real\_t toler )

Conjugate Gradient Squared solver function.

This function uses the preconditioned Conjugate Gradient Squared algorithm to solve a linear system with a sparse matrix.

The global variable Verbosity enables choosing output message level

- Verbosity < 2 : No output message
- Verbosity > 1 : Notify executing the function CG
- Verbosity > 2: Notify convergence with number of performed iterations or divergence
- Verbosity > 3 : Output each iteration number and residual
- Verbosity > 6 : Print final solution if convergence
- Verbosity > 10: Print obtained solution at each iteration

### **Parameters**

in	Α	Problem matrix (Instance of class SpMatrix).
in	P	Preconditioner (Instance of class Prec).
in	b	Right-hand side vector (class Vect)
in,out	x	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	max↔ _it	Maximum number of iterations.
in	toler	Tolerance for convergence (measured in relative weighted 2-Norm).

## Returns

<t⊷< th=""><th>Data type (real_t, float, complex<real_t>,)</real_t></th></t⊷<>	Data type (real_t, float, complex <real_t>,)</real_t>
_>	

### Author

Rachid Touzani

## Copyright

**GNU Lesser Public License** 

# int CGS ( const SpMatrix< $T_-$ > & A, int prec, const Vect< $T_-$ > & b, Vect< $T_-$ > & x, int $max\_it$ , real\_t toler )

Conjugate Gradient Squared solver function.

This function uses the preconditioned Conjugate Gradient Squared algorithm to solve a linear system with a sparse matrix.

The global variable Verbosity enables choosing output message level

- Verbosity < 2 : No output message
- Verbosity > 1 : Notify executing the function CG
- Verbosity > 2 : Notify convergence with number of performed iterations or divergence
- Verbosity > 3 : Output each iteration number and residual
- Verbosity > 6 : Print final solution if convergence
- Verbosity > 10: Print obtained solution at each iteration

### **Parameters**

in	Α	Problem matrix (Instance of class SpMatrix).
in	prec	Enum variable selecting a preconditioner, among the values IDENT_PREC, DIAG_PREC, ILU_PREC or SSOR_PREC
in	b	Right-hand side vector (class Vect)
in,out	x	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	max↔ _it	Maximum number of iterations.
in	toler	Tolerance for convergence (measured in relative weighted 2-Norm).

### Returns

< <i>T</i> ←	Data type (real_t, float, complex <real_t>,)</real_t>
_>	

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# int GMRes ( const SpMatrix< T $_->$ & A, const Prec< T $_->$ & P, const Vect< T $_->$ & b, Vect< T $_->$ & x, size $_-$ t m, int $max\_it$ , real $_-$ t toler )

GMRes solver function.

This function uses the preconditioned GMRES algorithm to solve a linear system with a sparse matrix.

The global variable Verbosity enables choosing output message level

- Verbosity < 2 : No output message
- Verbosity > 1 : Notify executing the function CG
- Verbosity > 2: Notify convergence with number of performed iterations or divergence
- Verbosity > 3 : Output each iteration number and residual
- Verbosity > 6 : Print final solution if convergence
- Verbosity > 10 : Print obtained solution at each iteration

### **Parameters**

in	Α	Problem matrix (Instance of class SpMatrix).
in	P	Preconditioner (Instance of class Prec).
in	b	Right-hand side vector (class Vect)
in,out	х	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	m	Number of subspaces to generate for iterations.
in	max↔ _it	Maximum number of iterations.
in	toler	Tolerance for convergence (measured in relative weighted 2-Norm).

### Returns

< <i>T</i> ←	Data type (double, float, complex <double>,)</double>
_>	

### Author

Rachid Touzani

## Copyright

GNU Lesser Public License

int GMRes ( const SpMatrix $< T_- > & A$ , int prec, const Vect $< T_- > & b$ , Vect $< T_- > & x$ , size\_t m, int max\_it, real\_t toler )

GMRes solver function.

This function uses the preconditioned GMRES algorithm to solve a linear system with a sparse matrix.

The global variable Verbosity enables choosing output message level

- Verbosity < 2 : No output message
- Verbosity > 1 : Notify executing the function CG
- Verbosity > 2: Notify convergence with number of performed iterations or divergence
- Verbosity > 3 : Output each iteration number and residual
- Verbosity > 6 : Print final solution if convergence
- Verbosity > 10: Print obtained solution at each iteration

### **Parameters**

in	A	Problem matrix (Instance of class SpMatrix).
in	prec	Enum variable selecting a preconditioner, among the values IDENT_PREC, DIAG_PREC, ILU_PREC or SSOR_PREC
in	b	Right-hand side vector (class Vect)
in,out	х	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	m	Number of subspaces to generate for iterations.
in	max↔ _it	Maximum number of iterations.
in	toler	Tolerance for convergence (measured in relative weighted 2-Norm).

### Returns

< <i>T</i> ←	Data type (double, float, complex <double>,)</double>
_>	

### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

# int GS ( const SpMatrix< $T_-$ > & A, const Vect< $T_-$ > & b, Vect< $T_-$ > & x, real\_t omega, int max\_it, real\_t toler )

Gauss-Seidel solver function.

This function uses the relaxed Gauss-Seidel algorithm to solve a linear system with a sparse matrix.

The global variable Verbosity enables choosing output message level

- Verbosity < 2 : No output message
- Verbosity > 1 : Notify executing the function GS
- Verbosity > 2: Notify convergence with number of performed iterations or divergence
- Verbosity > 3 : Output each iteration number and residual
- Verbosity > 6 : Print final solution if convergence
- Verbosity > 10 : Print obtained solution at each iteration

### **Parameters**

in	Α	Problem matrix (Instance of class SpMatrix).
in	b	Right-hand side vector (class Vect)
in,out	x	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	omega	Relaxation parameter.
in	max← _it	Maximum number of iterations.
in	toler	Tolerance for convergence (measured in relative weighted 2-Norm).

### Returns

< <i>T</i> ←	Data type (real_t, float, complex <real_t>,)</real_t>
_>	

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# int Jacobi ( const SpMatrix< $T_-$ > & A, const Vect< $T_-$ > & b, Vect< $T_-$ > & x, real\_t omega, int max\_it, real\_t toler )

Jacobi solver function.

### Parameters

in	Α	Problem matrix (Instance of class SpMatrix).
in	b	Right-hand side vector (class Vect)
in,out	х	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	omega	Relaxation parameter.
in	max↔ _it	Maximum number of iterations.
in,out	toler	Tolerance for convergence (measured in relative weighted 2-Norm).

## Returns

Number of performed iterations,

# **Template Parameters**

<t_></t_>	Data type (real_t, float, complex <real_t>,)</real_t>
<m⊷< th=""><th>Matrix storage class</th></m⊷<>	Matrix storage class
_>	

## Author

Rachid Touzani

# Copyright

GNU Lesser Public License

int Richardson ( const  $M_-$  & A, const Vect<  $T_-$  > & b, Vect<  $T_-$  > & x, real\_t omega, int max\_it, real\_t toler, int verbose )

Richardson solver function.

### Parameters

in	Α	Problem matrix problem (Instance of abstract class <b>M</b> <sub>-</sub> ).
in	b	Right-hand side vector (class Vect)
	x	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	omega	Relaxation parameter.
in	max↔ _it	Maximum number of iterations.
in	toler	Tolerance for convergence (measured in relative weighted 2-Norm).
in	verbose	Information output parameter (0: No output, 1: Output iteration information, 2 and greater: Output iteration information and solution at each iteration.

### Returns

nb\_it Number of performed iterations,

## **Template Parameters**

<t_></t_>	Data type (real_t, float, complex <real_t>,)</real_t>
<m←< td=""><td>Matrix storage class</td></m←<>	Matrix storage class
_>	

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

void Schur ( SkMatrix< T\_ > & A, SpMatrix< T\_ > & U, SpMatrix< T\_ > & L, SpMatrix< T\_ > & D, Vect< T\_ > & b, Vect< T\_ > & c )

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

The linear system is of the form

### Parameters

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

# Template Argument:

# **Template Parameters**

< <i>T</i> ←	data type (real_t, float,)
_>	

## Author

Rachid Touzani

# Copyright

GNU Lesser Public License

int SSOR ( const  $M_-$  & A, const  $Vect < T_- >$  & b,  $Vect < T_- >$  & x, int  $max\_it$ , real\_t toler) SSOR solver function.

# Parameters

in	A	Problem matrix (Instance of abstract class $M_{-}$ ).
in	b	Right-hand side vector (class Vect)
in,out	x	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	max↔ _it	Maximum number of iterations.
in	toler	Tolerance for convergence (measured in relative weighted 2-Norm).

### Returns

Number of performed iterations,

# **Template Arguments:**

- *T*<sub>-</sub> data type (double, float, ...)
- *M*<sub>-</sub> Matrix storage class

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# ostream & operator << ( ostream & s, TimeStepping & ts )

Output differential system information.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# **5.18 OFELI**

## **Modules**

• General Purpose Equations

Gathers equation related classes.

• Conservation Law Equations

Conservation law equations.

Electromagnetics

Electromagnetic equations.

• Fluid Dynamics

Fluid Dynamics equations.

• Interface Problems

Interface problems, including image processing.

• Laplace equation

Laplace and Poisson equations.

Porous Media problems

Porous Media equation classes.

• Solid Mechanics

Solid Mechanics finite element equations.

• Heat Transfer

Heat Transfer equations.

• Input/Output

Input/Output utility classes.

• Utilities

Utility functions and classes.

• Physical properties of media

Physical properties of materials and media.

• Global Variables

All global variables in the library.

• Finite Element Mesh

Mesh management classes

• Shape Function

Shape function classes.

• Solver

Solver functions and classes.

• Vector and Matrix

Vector and matrix classes.

## **Files**

• file AbsEqua.h

Definition file for abstract class AbsEqua.

• file ICPG1D.h

Definition file for class ICPG1D.

• file ICPG2DT.h

Definition file for class ICPG2DT.

• file ICPG3DT.h

Definition file for class ICPG3DT.

• file LCL1D.h

Definition file for class LCL1D.

• file LCL2DT.h

Definition file for class LCL2DT.

• file LCL3DT.h

Definition file for class LCL3DT.

• file Muscl.h

Definition file for class Muscl.

• file Muscl1D.h

Definition file for class Muscl1D.

• file Muscl2DT.h

Definition file for class Muscl2DT.

• file Muscl3DT.h

Definition file for class Muscl3DT.

• file BiotSavart.h

Definition file for class BiotSavart.

• file EC2D1T3.h

Definition file for class EC2D1T3.

• file EC2D2T3.h

Definition file for class EC2D2T3.

• file Equa\_Electromagnetics.h

Definition file for class FE\_Electromagnetics.

• file HelmholtzBT3.h

 $Definition\ file\ for\ class\ Helmholtz BT3.$ 

• file Equation.h

Definition file for class Equation.

• file Equa\_Fluid.h

Definition file for class Equa\_Fluid.

• file NSP2DQ41.h

Definition file for class NSP2DQ41.

• file TINS2DT3S.h

Definition file for class TINS2DT3S.

• file TINS3DT4S.h

Definition file for class TINS3DT4S.

• file Equa\_Laplace.h

Definition file for class Equa\_Laplace.

• file Laplace1DL2.h

Definition file for class Laplace1DL2.

• file Laplace1DL3.h

Definition file for class Laplace1DL3.

• file Laplace2DT3.h

Definition file for class Laplace2DT3.

• file Laplace2DT6.h

Definition file for class Laplace2DT6.

• file Laplace3DT4.h

Definition file for class Laplace3DT4.

• file SteklovPoincare2DBE.h

Definition file for class SteklovPoincare2DBE.

• file Equa\_Porous.h

Definition file for class Equa\_Porous.

• file WaterPorous1D.h

Definition file for class WaterPorous1D.

• file WaterPorous2D.h

Definition file for class WaterPorous2D.

• file Bar2DL2.h

Definition file for class Bar2DL2.

• file Beam3DL2.h

Definition file for class Beam3DL2.

• file Elas2DQ4.h

Definition file for class Elas2DQ4.

• file Elas2DT3.h

Definition file for class Elas2DT3.

• file Elas3DH8.h

Definition file for class Elas3DH8.

• file Elas3DT4.h

Definition file for class Elas3DT4.

• file Equa\_Solid.h

Definition file for class Equa\_Solid.

• file DC1DL2.h

Definition file for class DC1DL2.

• file DC2DT3.h

Definition file for class DC2DT3.

• file DC2DT6.h

Definition file for class DC2DT6.

• file DC3DAT3.h

Definition file for class DC3DAT3.

• file DC3DT4.h

Definition file for class DC3DT4.

• file Equa\_Therm.h

Definition file for class Equa\_Therm.

• file PhaseChange.h

Definition file for class PhaseChange and its parent abstract class.

• file Funct.h

Definition file for class Funct.

• file IOField.h

Definition file for class IOField.

• file IPF.h

Definition file for class IPF.

• file output.h

File that contains some output utility functions.

• file Prescription.h

Definition file for class Prescription.

• file saveField.h

Prototypes for functions to save mesh in various file formats.

• file saveField.h

Prototypes for functions to save mesh in various file formats.

• file Tabulation.h

Definition file for class Tabulation.

• file BMatrix.h

Definition file for class BMatrix.

• file DMatrix.h

Definition file for class DMatrix.

file DSMatrix.h

Definition file for abstract class DSMatrix.

• file LocalMatrix.h

Definition file for class LocalMatrix.

file LocalVect.h

Definition file for class LocalVect.

• file Matrix.h

Definition file for abstract class Matrix.

• file Point.h

Definition file and implementation for class Point.

• file Point2D.h

Definition file for class Point2D.

• file SkMatrix.h

Definition file for class SkMatrix.

• file SkSMatrix.h

Definition file for class SkSMatrix.

• file SpMatrix.h

 $Definition\ file\ for\ class\ SpMatrix.$ 

• file TrMatrix.h

Definition file for class TrMatrix.

• file Domain.h

Definition file for class Domain.

• file Edge.h

Definition file for class Edge.

• file Element.h

Definition file for class Element.

• file Figure.h

Definition file for figure classes.

• file getMesh.h

Definition file for mesh conversion functions.

• file Grid.h

Definition file for class Grid.

• file Material.h

Definition file for class Material.

• file Mesh.h

Definition file for class Mesh.

• file MeshAdapt.h

Definition file for class MeshAdapt.

• file MeshExtract.h

Definition file for classes for extracting submeshes.

• file MeshUtil.h

*Definitions of utility functions for meshes.* 

• file Node.h

Definition file for class Node.

file saveMesh.h

Prototypes for functions to save mesh in various file formats.

• file Side.h

Definition file for class Side.

file FEShape.h

Definition file for class FEShape.

• file Hexa8.h

Definition file for class Hexa8.

• file Line2.h

Definition file for class Line2.

• file Line3.h

Definition file for class Line3.

• file Penta6.h

Definition file for class Penta6.

• file Quad4.h

Definition file for class Quad4.

• file Tetra4.h

Definition file for class Tetra4.

• file Triang3.h

Definition file for class Triang3.

• file Triang6S.h

Definition file for class Triang6S.

• file BiCG.h

Solves an unsymmetric linear system of equations using the BiConjugate Gradient method.

• file BSpline.h

Function to perform a B-spline interpolation.

• file CG.h

Functions to solve a symmetric positive definite linear system of equations using the Conjugate Gradient method.

• file CGS.h

Solves an unsymmetric linear system of equations using the Conjugate Gradient Squared method.

• file EigenProblemSolver.h

Definition file for class EigenProblemSolver.

file GMRes.h

Function to solve a linear system of equations using the Generalized Minimum Residual method.

• file GS.h

Function to solve a linear system of equations using the Gauss-Seidel method.

• file Integration.h

Definition file for numerical integration class.

• file Jacobi.h

Function to solve a linear system of equations using the Jacobi method.

• file MyNLAS.h

Definition file for abstract class MyNLAS.

• file MyOpt.h

Definition file for abstract class MyOpt.

• file ODESolver.h

Definition file for class ODESolver.

• file Prec.h

Definition file for preconditioning classes.

• file Richardson.h

Function to solve a linear system of equations using the Richardson method.

• file SSOR.h

Function to solve a linear system of equations using the Symmetric Successive Over Relaxation method.

• file TimeStepping.h

Definition file for class TimeStepping.

• file constants.h

File that contains some widely used constants.

• file Gauss.h

Definition file for struct Gauss.

• file qksort.h

File that contains template quick sorting function.

• file Timer.h

Definition file for class Timer.

• file util.h

File that contains various utility functions.

## **Classes**

• class AbsEqua < T\_>

Mother abstract class to describe equation.

class LocalVect< T<sub>-</sub>, N<sub>-</sub> >

Handles small size vectors like element vectors.

• class ICPG1D

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 1-D.

• class ICPG2DT

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 2-D.

class ICPG3DT

 ${\it Class\ to\ solve\ the\ Inviscid\ compressible\ fluid\ flows\ (Euler\ equations)\ for\ perfect\ gas\ in\ 3-D.}$ 

• class LCL1D

Class to solve the linear conservation law (Hyperbolic equation) in 1-D by a MUSCL Finite Volume scheme.

• class LCL2DT

Class to solve the linear hyperbolic equation in 2-D by a MUSCL Finite Volume scheme on triangles.

• class LCL3DT

Class to solve the linear conservation law equation in 3-D by a MUSCL Finite Volume scheme on tetrahedra.

• class Muscl

Parent class for hyperbolic solvers with Muscl scheme.

• class Vect< T\_>

To handle general purpose vectors.

• class Muscl1D

Class for 1-D hyperbolic solvers with Muscl scheme.

class Muscl2DT

Class for 2-D hyperbolic solvers with Muscl scheme.

class Muscl3DT

Class for 3-D hyperbolic solvers with Muscl scheme using tetrahedra.

• class BiotSavart

Class to compute the magnetic induction from the current density using the Biot-Savart formula.

• class EC2D1T3

Eddy current problems in 2-D domains using solenoidal approximation.

class EC2D2T3

Eddy current problems in 2-D domains using transversal approximation.

class Equa\_Electromagnetics< T\_, NEN\_, NEE\_, NSN\_, NSE\_>

Abstract class for Electromagnetics Equation classes.

• class HelmholtzBT3

Builds finite element arrays for Helmholtz equations in a bounded media using 3-Node triangles.

class Equation < T\_, NEN\_, NEE\_, NSN\_, NSE\_ >

Abstract class for all equation classes.

• class Equa\_Fluid < T\_, NEN\_, NEE\_, NSN\_, NSE\_ >

Abstract class for Fluid Dynamics Equation classes.

• class NSP2DQ41

Builds finite element arrays for incompressible Navier-Stokes equations in 2-D domains using  $Q_1/P_0$  element and a penaly formulation for the incompressibility condition.

class TINS2DT3S

Builds finite element arrays for transient incompressible fluid flow using Navier-Stokes equations in 2-D domains. Numerical approximation uses stabilized 3-node triangle finite elements for velocity and pressure. 2nd-order projection scheme is used for time integration.

• class TINS3DT4S

Builds finite element arrays for transient incompressible fluid flow using Navier-Stokes equations in 3-D domains. Numerical approximation uses stabilized 4-node tatrahedral finite elements for velocity and pressure. 2nd-order projection scheme is used for time integration.

class FastMarching2D

To run a Fast Marching Method on 2-D structured uniform grids.

• class FMM2D

class for the fast marching 2-D algorithm

class FMM3D

class for the 3-D fast marching algorithm

class FMMSolver

The Fast Marching Method solver.

• class Equa\_Laplace< T\_, NEN\_, NEE\_, NSN\_, NSE\_ >

Abstract class for classes about the Laplace equation.

class Laplace1DL2

To build element equation for a 1-D elliptic equation using the 2-Node line element  $(P_1)$ .

• class Laplace1DL3

To build element equation for the 1-D elliptic equation using the 3-Node line  $(P_2)$ .

class Laplace2DT3

To build element equation for the Laplace equation using the 2-D triangle element  $(P_1)$ .

• class Laplace2DT6

To build element equation for the Laplace equation using the 2-D triangle element  $(P_2)$ .

class SteklovPoincare2DBE

Solver of the Steklov Poincare problem in 2-D geometries using piecewie constant boundary elemen.

• class Equa\_Porous< T\_, NEN\_, NEE\_, NSN\_, NSE\_>

Abstract class for Porous Media Finite Element classes.

class WaterPorous2D

*To solve water flow equations in porous media (1-D)* 

• class Bar2DL2

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

• class Beam3DL2

To build element equations for 3-D beam equations using 2-node lines.

• class Elas2DQ4

To build element equations for 2-D linearized elasticity using 4-node quadrilaterals.

• class Elas2DT3

To build element equations for 2-D linearized elasticity using 3-node triangles.

class Elas3DH8

To build element equations for 3-D linearized elasticity using 8-node hexahedra.

• class Elas3DT4

To build element equations for 3-D linearized elasticity using 4-node tetrahedra.

• class Equa\_Solid < T\_, NEN\_, NEE\_, NSN\_, NSE\_ >

Abstract class for Solid Mechanics Finite Element classes.

• class DC1DL2

Builds finite element arrays for thermal diffusion and convection in 1-D using 2-Node elements.

class DC2DT3

Builds finite element arrays for thermal diffusion and convection in 2-D domains using 3-Node triangles.

• class DC2DT6

Builds finite element arrays for thermal diffusion and convection in 2-D domains using 6-Node triangles.

class DC3DAT3

Builds finite element arrays for thermal diffusion and convection in 3-D domains with axisymmetry using 3-Node triangles.

class DC3DT4

Builds finite element arrays for thermal diffusion and convection in 3-D domains using 4-Node tetrahedra.

class Equa\_Therm< T\_, NEN\_, NEE\_, NSN\_, NSE\_>

Abstract class for Heat transfer Finite Element classes.

• class PhaseChange

This class enables defining phase change laws for a given material.

class Funct

A simple class to parse real valued functions.

• class IOField

Enables working with files in the XML Format.

class IPF

To read project parameters from a file in IPF format.

• class Prescription

To prescribe various types of data by an algebraic expression. Data may consist in boundary conditions, forces, tractions, fluxes, initial condition. All these data types can be defined through an enumerated variable.

• class Tabulation

To read and manipulate tabulated functions.

• class BMatrix< T\_>

To handle band matrices.

• class DMatrix< T\_>

To handle dense matrices.

class DSMatrix< T<sub>-</sub>>

To handle symmetric dense matrices.

class SkMatrix< T<sub>-</sub>>

*To handle square matrices in skyline storage format.* 

• class SkSMatrix< T\_>

To handle symmetric matrices in skyline storage format.

class SpMatrix < T<sub>-</sub> >

To handle matrices in sparse storage format.

• class LocalMatrix< T\_, NR\_, NC\_>

Handles small size matrices like element matrices, with a priori known size.

• class Matrix< T\_>

Virtual class to handle matrices for all storage formats.

• class Point< T\_>

*Defines a point with arbitrary type coordinates.* 

• class Point2D < T\_>

 $Defines\ a\ 2-D\ point\ with\ arbitrary\ type\ coordinates.$ 

• class TrMatrix< T\_>

To handle tridiagonal matrices.

class Domain

To store and treat finite element geometric information.

class Edge

To describe an edge.

• class Element

To store and treat finite element geometric information.

• class Figure

To store and treat a figure (or shape) information.

• class Rectangle

To store and treat a rectangular figure.

• class Brick

To store and treat a brick (parallelepiped) figure.

• class Circle

To store and treat a circular figure.

class Sphere

To store and treat a sphere.

• class Ellipse

To store and treat an ellipsoidal figure.

• class Triangle

To store and treat a triangle.

class Polygon

To store and treat a polygonal figure.

class Grid

To manipulate structured grids.

class Material

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

• class Mesh

To store and manipulate finite element meshes.

class MeshAdapt

To adapt mesh in function of given solution.

• class NodeList

Class to construct a list of nodes having some common properties.

class ElementList

Class to construct a list of elements having some common properties.

class SideList

Class to construct a list of sides having some common properties.

class EdgeList

Class to construct a list of edges having some common properties.

class Node

To describe a node.

class Partition

To partition a finite element mesh into balanced submeshes.

• class Side

To store and treat finite element sides (edges in 2-D or faces in 3-D)

• class OFELIException

To handle exceptions in OFELI.

• class FEShape

Parent class from which inherit all finite element shape classes.

class triangle

Defines a triangle. The reference element is the rectangle triangle with two unit edges.

• class Hexa8

Defines a three-dimensional 8-node hexahedral finite element using Q1-isoparametric interpolation.

• class Line2

To describe a 2-Node planar line finite element.

• class Line3

To describe a 3-Node quadratic planar line finite element.

class Penta6

Defines a 6-node pentahedral finite element using  $P_1$  interpolation in local coordinates (s.x, s.y) and  $Q_1$  isoparametric interpolation in local coordinates (s.x, s.z) and (s.y, s.z).

• class Quad4

*Defines a 4-node quadrilateral finite element using*  $Q_1$  *isoparametric interpolation.* 

class Tetra4

Defines a three-dimensional 4-node tetrahedral finite element using  $P_1$  interpolation.

• class Triang3

Defines a 3-Node (P1) triangle.

• class Triang6S

Defines a 6-Node straight triangular finite element using P<sub>2</sub> interpolation.

• class Prec< T\_>

To set a preconditioner.

• class EigenProblemSolver

Class to find eigenvalues and corresponding eigenvectors of a given matrix in a generalized eigenproblem, i.e. Find scalars l and non-null vectors v such that  $[K]\{v\} = l[M]\{v\}$  where [K] and [M] are symmetric matrices. The eigenproblem can be originated from a PDE. For this, we will refer to the matrices K and M as Stiffness and Mass matrices respectively.

class Integration

Class for numerical integration methods.

• class Iter< T\_>

Class to drive an iterative process.

• class LinearSolver< T\_>

Class to solve systems of linear equations by iterative methods.

class MyNLAS

Abstract class to define by user specified function.

class MyOpt

Abstract class to define by user specified optimization function.

class ODESolver

To solve a system of ordinary differential equations.

• class TimeStepping

To solve time stepping problems, i.e. systems of linear ordinary differential equations of the form  $[A2]\{y''\}+[A1]\{y'\}+[A0]\{y\}=\{b\}$ .

• class Gauss

Calculate data for Gauss integration.

class Timer

To handle elapsed time counting.

### **Enumerations**

```
• enum PDE_Terms {
  CONSISTENT_MASS = 0x00001000,
 LUMPED\_MASS = 0x00002000,
  MASS = 0x00002000,
  CAPACITY = 0x00004000,
 CONSISTENT_CAPACITY = 0x00004000,
 LUMPED_CAPACITY = 0x00008000,
  VISCOSITY = 0x00010000,
 STIFFNESS = 0x00020000.
 DIFFUSION = 0x00040000,
  MOBILITY = 0x00040000,
  CONVECTION = 0x00080000,
  DEVIATORIC = 0x00100000,
 DILATATION = 0x00200000,
 ELECTRIC = 0 \times 00400000,
  MAGNETIC = 0x008000000,
  LOAD = 0x01000000,
 HEAT\_SOURCE = 0x020000000,
  BOUNDARY_TRACTION = 0x04000000,
 HEAT_FLUX = 0x080000000,
 CONTACT = 0x100000000,
 BUOYANCY = 0x200000000,
```

```
LORENTZ_FORCE = 0x40000000 }
enum EqDataType {
 INITIAL = 1,
 INITIAL_FIELD = 1,
 SOLUTION = 1,
 INITIAL_AUX_1 = 2,
 INITIAL_AUX_2 = 3,
 INITIAL_AUX_3 = 4,
 INITIAL_AUX_4 = 5,
 BOUNDARY_CONDITION = 6,
 BODY\_FORCE = 7,
 SOURCE = 7,
 POINT_FORCE = 8,
 BOUNDARY_FORCE = 9,
 FLUX = 9,
 TRACTION = 9,
  AUX_INPUT_FIELD_1 = 10,
  AUX_INPUT_FIELD_2 = 11,
  AUX_INPUT_FIELD_3 = 11,
  AUX_INPUT_FIELD_4 = 12,
 DISPLACEMENT_FIELD = 13,
  VELOCITY_FIELD = 14,
 PRESSURE_FIELD = 15,
 TEMPERATURE_FIELD = 16,
 CONTACT_DISTANCE = 17 }
• enum Analysis {
 STATIONARY = 0,
 STEADY\_STATE = 0,
 TRANSIENT = 1,
 TRANSIENT_ONE\_STEP = 2,
 OPTIMIZATION = 3,
 EIGEN = 4
• enum TimeScheme {
 NONE = 0,
 FORWARD_EULER = 1,
 BACKWARD_EULER = 2,
 CRANK_NICOLSON = 3,
 HEUN = 4,
 NEWMARK = 5,
 LEAP\_FROG = 6,
 ADAMS_BASHFORTH = 7,
 AB2 = 7,
 RUNGE_KUTTA = 8,
 RK4 = 8.
 RK3_TVD = 9,
 BDF2 = 10 }
• enum FEType {
 FE_2D_3N,
 FE_2D_6N,
 FE_2D_4N,
 FE_3D_AXI_3N,
 FE_3D_4N,
 FE_3D_8N }
```

```
• enum AccessType {
  IN = 1,
  OUT = 2
     Enumerated values for file access type.
enum MatrixType {
  DENSE = 1,
 SKYLINE = 2,
 SPARSE = 4,
 DIAGONAL = 8,
 TRIDIAGONAL = 16,
 BAND = 32
 SYMMETRIC = 64,
  UNSYMMETRIC = 128,
 IDENTITY = 256 }
• enum Iteration {
 DIRECT\_SOLVER = 0,
 CG\_SOLVER = 1,
 CGS\_SOLVER = 2,
 BICG_SOLVER = 3,
  BICG\_STAB\_SOLVER = 4,
 GMRES_SOLVER = 5 }
     Choose iterative solver for the linear system.
• enum Preconditioner {
 IDENT_PREC = 0,
 DIAG\_PREC = 1,
 DILU_PREC = 2,
 ILU\_PREC = 3,
 SSOR_PREC = 4 }
     Choose preconditioner for the linear system.
• enum {
  BY_VALUE,
  BY_POSITION,
  BY_FIELD }
• enum BCType {
  PERIODIC_A = 9999,
 PERIODIC_B = -9999,
 CONTACT_BC = 9998,
  CONTACT_M = 9997,
  CONTACT_S = -9997,
 SLIP = 9996 }
enum IntegrationScheme {
 LEFT_RECTANGLE = 0,
 RIGHT_RECTANGLE = 1,
 MID_RECTANGLE = 2,
 TRAPEZOIDAL = 3,
 SIMPSON = 4,
 GAUSS_LEGENDRE = 5 }
```

## **Functions**

• T\_\*A()

Return element matrix.

• T\_\*b()

Return element right-hand side.

• T\_ \* Prev ()

Return element previous vector.

• IOField ()

Default constructor.

• IOField (const string &file, AccessType access, bool compact=true)

Constructor using file name.

• IOField (const string &mesh\_file, const string &file, Mesh &ms, AccessType access, bool compact=true)

Constructor using file name, mesh file and mesh.

• IOField (const string &file, Mesh &ms, AccessType access, bool compact=true)

Constructor using file name and mesh.

IOField (const string &file, AccessType access, const string &name)

Constructor using file name and field name.

• ∼IOField ()

Destructor.

void setMeshFile (const string &file)

Set mesh file.

• void open ()

Open file.

• void open (const string &file, AccessType access)

Open file.

• void close ()

Close file.

void put (Mesh &ms)

Store mesh in file.

void put (const Vect< real\_t > &v)

Store Vect instance v in file.

• real\_t get (Vect< real\_t > &v)

Get Vect v instance from file.

• int get (Vect< real\_t > &v, const string &name)

Get Vect v instance from file if the field has the given name.

• int get (DMatrix < real\_t > &A, const string &name)

Get DMatrix A instance from file if the field has the given name.

• int get (DSMatrix< real\_t > &A, const string &name)

Get DSMatrix A instance from file if the field has the given name.

• int get (Vect< real\_t > &v, real\_t t)

Get Vect v instance from file corresponding to a specific time value.

void saveGMSH (string output\_file, string mesh\_file)

Save field vectors in a file using GMSH format.

• Tabulation ()

Default constructor.

• Tabulation (string file)

Constructor using file name.

• ∼Tabulation ()

Destructor.

• void setFile (string file)

Set file name.

• real\_t getValue (string funct, real\_t v)

Return the calculated value of the function.

• real\_t getDerivative (string funct, real\_t v)

Return the derivative of the function at a given point.

• real\_t getValue (string funct, real\_t v1, real\_t v2)

Return the calculated value of the function.

• real\_t getValue (string funct, real\_t v1, real\_t v2, real\_t v3)

Return the calculated value of the function.

• Point < double > CrossProduct (const Point < double > &lp, const Point < double > &rp)

Return Cross product of two vectors lp and rp

• SpMatrix ()

Default constructor.

• SpMatrix (size\_t nr, size\_t nc)

Constructor that initializes current instance as a dense matrix.

• SpMatrix (size\_t size, int is\_diagonal=false)

Constructor that initializes current instance as a dense matrix.

SpMatrix (Mesh &mesh, size\_t dof=0, int is\_diagonal=false)

Constructor using a Mesh instance.

• SpMatrix (const Vect< RC > &I, int opt=1)

Constructor for a square matrix using non zero row and column indices.

SpMatrix (const Vect< RC > &I, const Vect< T<sub>-</sub> > &a, int opt=1)

Constructor for a square matrix using non zero row and column indices.

SpMatrix (size\_t nr, size\_t nc, const vector < size\_t > &row\_ptr, const vector < size\_t > &col←
 \_ind)

Constructor for a rectangle matrix.

SpMatrix (size\_t nr, size\_t nc, const vector < size\_t > &row\_ptr, const vector < size\_t > &col←
 \_ind, const vector < T\_ > &a)

Constructor for a rectangle matrix.

• SpMatrix (const vector < size\_t > &row\_ptr, const vector < size\_t > &col\_ind)

Constructor for a rectangle matrix.

SpMatrix (const vector < size\_t > &row\_ptr, const vector < size\_t > &col\_ind, const vector < T\_ > &a)

Constructor for a rectangle matrix.

• SpMatrix (const SpMatrix &m)

Copy constructor.

~SpMatrix ()

Destructor.

• void Identity ()

Define matrix as identity.

• void Dense ()

Define matrix as a dense one.

• void Diagonal ()

Define matrix as a diagonal one.

• void Diagonal (const T<sub>-</sub> &a)

Define matrix as a diagonal one with diagonal entries equal to a

• void Laplace1D (size\_t n, real\_t h)

*Sets the matrix as the one for the Laplace equation in 1-D.* 

• void Laplace2D (size\_t nx, size\_t ny)

*Sets the matrix as the one for the Laplace equation in 2-D.* 

void setMesh (Mesh &mesh, size\_t dof=0)

Determine mesh graph and initialize matrix.

• void setOneDOF ()

Activate 1-DOF per node option.

• void setSides ()

Activate Sides option.

void setDiag ()

Store diagonal entries in a separate internal vector.

void DiagPrescribe (Mesh &mesh, Vect< T<sub>-</sub> > &b, const Vect< T<sub>-</sub> > &u)

Impose by a diagonal method an essential boundary condition.

• void DiagPrescribe (Vect< T $_->$  &b, const Vect< T $_->$  &u)

Impose by a diagonal method an essential boundary condition using the Mesh instance provided by the constructor.

• void setSize (size\_t size)

Set size of matrix (case where it's a square matrix).

• void setSize (size\_t nr, size\_t nc)

Set size (number of rows) of matrix.

• void setGraph (const Vect< RC > &I, int opt=1)

Set graph of matrix by giving a vector of its nonzero entries.

• Vect< T\_> getRow (size\_t i) const

Get i-th row vector.

• Vect< T\_> getColumn (size\_t j) const

Get j-th column vector.

• T<sub>-</sub> & operator() (size\_t i, size\_t j)

Operator () (Non constant version)

• T\_ operator() (size\_t i, size\_t j) const

Operator () (Constant version)

• T\_ operator() (size\_t i) const

Operator () with one argument (Constant version)

• T\_operator[] (size\_t i) const

Operator [] (Constant version).

• Vect $< T_- > operator*$  (const Vect $< T_- > &x$ ) const

*Operator* \* to multiply matrix by a vector.

• SpMatrix< T<sub>-</sub> > & operator\*= (const T<sub>-</sub> &a)

*Operator* \*= to premultiply matrix by a constant.

void getMesh (Mesh &mesh)

Get mesh instance whose reference will be stored in current instance of SpMatrix.

• void Mult (const Vect< T\_> &x, Vect< T\_> &y) const

Multiply matrix by vector and save in another one.

• void MultAdd (const Vect<  $T_-> &x$ , Vect<  $T_-> &y$ ) const

Multiply matrix by vector x and add to y.

void MultAdd (T<sub>-</sub> a, const Vect< T<sub>-</sub> > &x, Vect< T<sub>-</sub> > &y) const

Multiply matrix by vector  $\mathbf{a} * \mathbf{x}$  and add to  $\mathbf{y}$ .

• void TMult (const Vect<  $T_->$  &x, Vect<  $T_->$  &y) const

Multiply transpose of matrix by vector x and save in y.

• void Axpy (T<sub>-</sub> a, const SpMatrix< T<sub>-</sub> > &m)

Add to matrix the product of a matrix by a scalar.

• void Axpy  $(T_- a, const Matrix < T_- > *m)$ 

Add to matrix the product of a matrix by a scalar.

• void set (size\_t i, size\_t j, const T\_ &val)

Assign a value to an entry of the matrix.

void add (size\_t i, size\_t j, const T\_ &val)

Add a value to an entry of the matrix.

• void operator= (const T<sub>-</sub> &x)

Operator =.

• size\_t getColInd (size\_t i) const

Return storage information.

• size\_t getRowPtr (size\_t i) const

Return Row pointer at position i.

• int solve (const Vect<  $T_->$  &b, Vect<  $T_->$  &x, bool fact=false)

Solve the linear system of equations.

• void setSolver (Iteration solver=CG\_SOLVER, Preconditioner prec=DIAG\_PREC, int max ← \_it=1000, real\_t toler=1.e-8)

Choose solver and preconditioner for an iterative procedure.

• void clear ()

brief Set all matrix entries to zero

• T\_\* get () const

Return C-Array.

• T\_get (size\_t i, size\_t j) const

Return entry (i, j) of matrix if this one is stored, 0 otherwise.

• TrMatrix ()

Default constructor.

• TrMatrix (size\_t size)

Constructor for a tridiagonal matrix with size rows.

• TrMatrix (const TrMatrix &m)

Copy Constructor.

• ∼TrMatrix ()

Destructor.

• void Identity ()

Define matrix as identity matrix.

• void Diagonal ()

Define matrix as a diagonal one.

• void Diagonal (const T<sub>-</sub> &a)

Define matrix as a diagona one and assign value a to all diagonal entries.

• void Laplace1D (real\_t h)

Define matrix as the one of 3-point finite difference discretization of the second derivative.

void setSize (size\_t size)

Set size (number of rows) of matrix.

• void MultAdd (const Vect<  $T_->$  &x, Vect<  $T_->$  &y) const

Multiply matrix by vector x and add result to y.

void MultAdd (T<sub>-</sub> a, const Vect< T<sub>-</sub> > &x, Vect< T<sub>-</sub> > &y) const

Multiply matrix by vector a\*x and add result to y.

• void Mult (const Vect<  $T_->$  &x, Vect<  $T_->$  &y) const

Multiply matrix by vector **x** and save result in **y**.

• void TMult (const Vect $< T_- > &x$ , Vect $< T_- > &y$ ) const

Multiply transpose of matrix by vector x and save result in y.

• void Axpy (T<sub>-</sub> a, const TrMatrix< T<sub>-</sub> > &m)

Add to matrix the product of a matrix by a scalar.

• void Axpy (T<sub>-</sub> a, const Matrix< T<sub>-</sub> > \*m)

Add to matrix the product of a matrix by a scalar.

• void set (size\_t i, size\_t j, const T\_ &val)

Assign constant val to an entry (i, j) of the matrix.

• void add (size\_t i, size\_t j, const T\_ &val)

Add constant val value to an entry (i, j) of the matrix.

• T\_operator() (size\_t i, size\_t j) const

 $Operator\ ()\ (Constant\ version).$ 

• T<sub>-</sub> & operator() (size\_t i, size\_t j)

Operator () (Non constant version).

• TrMatrix< T $_->$  & operator= (const TrMatrix< T $_->$  &m)

Operator = .

• TrMatrix $< T_- > & operator = (const T_- & x)$ 

 $Operator = Assign \ matrix \ to \ identity \ times \ x.$ 

• TrMatrix< T<sub>−</sub> > & operator\*= (const T<sub>−</sub> &x)

Operator \*=.

• int solve (Vect< T\_> &b, bool fact=true)

Solve a linear system with current matrix (forward and back substitution).

• int solve (const Vect<  $T_->$  &b, Vect<  $T_->$  &x, bool fact=false)

Solve a linear system with current matrix (forward and back substitution).

• T\_\* get () const

Return C-Array.

• T<sub>\_</sub> get (size\_t i, size\_t j) const

Return entry (i, j) of matrix.

• Grid ()

Construct a default grid with 10 intervals in each direction.

• Grid (real\_t xm, real\_t xM, size\_t npx)

Construct a 1-D structured grid given its extremal coordinates and number of intervals.

Grid (real\_t xm, real\_t xM, real\_t ym, real\_t yM, size\_t npx, size\_t npy)

Construct a 2-D structured grid given its extremal coordinates and number of intervals.

Grid (Point < real\_t > m, Point < real\_t > M, size\_t npx, size\_t npy)

Construct a 2-D structured grid given its extremal coordinates and number of intervals.

Grid (real\_t xm, real\_t xM, real\_t ym, real\_t yM, real\_t zm, real\_t zM, size\_t npx, size\_t npx, size\_t npz)

Construct a 3-D structured grid given its extremal coordinates and number of intervals.

• Grid (Point< real\_t > m, Point< real\_t > M, size\_t npx, size\_t npy, size\_t npz)

Construct a 3-D structured grid given its extremal coordinates and number of intervals.

• void setXMin (const Point < real\_t > &x)

Set min. coordinates of the domain.

- void setXMax (const Point < real\_t > &x)
- void setDomain (real\_t xmin, real\_t xmax)

Set Dimensions of the domain: 1-D case.

void setDomain (real\_t xmin, real\_t xmax, real\_t ymin, real\_t ymax)

Set Dimensions of the domain: 2-D case.

• void setDomain (real\_t xmin, real\_t xmax, real\_t ymin, real\_t ymax, real\_t zmin, real\_t zmax)

Set Dimensions of the domain: 3-D case.

• void setDomain (Point< real\_t > xmin, Point< real\_t > xmax)

Set Dimensions of the domain: 3-D case.

• const Point< real\_t > & getXMin () const

Return min. Coordinates of the domain.

• const Point< real\_t > & getXMax () const

Return max. Coordinates of the domain.

• void setN (size\_t nx, size\_t ny=0, size\_t nz=0)

Set number of grid intervals in the x, y and z-directions.

• size\_t getNx () const

Return number of grid intervals in the x-direction.

• size\_t getNy () const

Return number of grid intervals in the y-direction.

• size\_t getNz () const

Return number of grid intervals in the z-direction.

• real\_t getHx () const

Return grid size in the x-direction.

• real\_t getHy () const

Return grid size in the y-direction.

• real\_t getHz () const

Return grid size in the z-direction.

• Point< real\_t > getCoord (size\_t i) const

Return coordinates a point with label i in a 1-D grid.

Point< real\_t > getCoord (size\_t i, size\_t j) const

Return coordinates a point with label (i, j) in a 2-D grid.

Point< real\_t > getCoord (size\_t i, size\_t j, size\_t k) const

Return coordinates a point with label (i, j, k) in a 3-D grid.

• real\_t getX (size\_t i) const

Return x-coordinate of point with index i

• real\_t getY (size\_t j) const

Return y-coordinate of point with index j

• real\_t getZ (size\_t k) const

Return z-coordinate of point with index k

• Point2D< real\_t > getXY (size\_t i, size\_t j) const

Return coordinates of point with indices (i, j)

• Point< real\_t > getXYZ (size\_t i, size\_t j, size\_t k) const

Return coordinates of point with indices (i, j, k)

• real\_t getCenter (size\_t i) const

Return coordinates of center of a 1-D cell with indices i, i+1

• Point< real\_t > getCenter (size\_t i, size\_t j) const

Return coordinates of center of a 2-D cell with indices (i,j), (i+1,j), (i+1,j+1), (i,j+1)

Point< real\_t > getCenter (size\_t i, size\_t j, size\_t k) const

Return coordinates of center of a 3-D cell with indices (i,j,k), (i+1,j,k), (i+1,j+1,k), (i,j+1,k), (i,j,k+1), (i+1,j,k+1), (i+1,j+1,k+1), (i,j+1,k+1)

• void setCode (string exp, int code)

Set a code for some grid points.

• void setCode (int side, int code)

Set a code for grid points on sides.

int getCode (int side) const

Return code for a side number.

• int getCode (size\_t i, size\_t j) const

Return code for a grid point.

• int getCode (size\_t i, size\_t j, size\_t k) const

Return code for a grid point.

• size\_t getDim () const

Return space dimension.

• void Deactivate (size\_t i)

Change state of a cell from active to inactive (1-D grid)

void Deactivate (size\_t i, size\_t j)

Change state of a cell from active to inactive (2-D grid)

• void Deactivate (size\_t i, size\_t j, size\_t k)

Change state of a cell from active to inactive (2-D grid)

• int isActive (size\_t i) const

Say if cell is active or not (1-D grid)

• int isActive (size\_t i, size\_t j) const

Say if cell is active or not (2-D grid)

• int isActive (size\_t i, size\_t j, size\_t k) const

Say if cell is active or not (3-D grid)

ostream & operator<< (ostream &s, const Grid &g)</li>

Output grid data.

• OFELIException (const std::string &s)

This form will be used most often in a throw.

• OFELIException ()

Throw with no error message.

• Iter ()

Default Constructor.

• Iter (int max\_it, real\_t toler)

Constructor with iteration parameters.

• bool check (Vect< T $_->$  &u, const Vect< T $_->$  &v, int opt=2)

Check convergence.

• bool check (T\_ &u, const T\_ &v)

Check convergence for a scalar case (one equation)

## 5.18.1 Detailed Description

## 5.18.2 Enumeration Type Documentation

#### enum PDE\_Terms

Enumerate variable that selects various terms in partial differential equations

#### Enumerator

CONSISTENT\_MASS Consistent mass term

LUMPED\_MASS Lumped mass term

MASS Consistent mass term

CAPACITY Consistent capacity term

CONSISTENT\_CAPACITY Consistent capacity term

LUMPED\_CAPACITY Lumped capacity term

VISCOSITY Viscosity term

STIFFNESS Stiffness term

**DIFFUSION** Diffusion term

MOBILITY Mobility term

**CONVECTION** Convection term

**DEVIATORIC** Deviatoric term

**DILATATION** Dilatational term

**ELECTRIC** Electric term

MAGNETIC Magnetic term

LOAD Body load term

**HEAT\_SOURCE** Body heat source term

BOUNDARY\_TRACTION Boundary traction (pressure) term

HEAT\_FLUX Boundary heat flux term

CONTACT Signorini contact

**BUOYANCY** Buoyancy force term

LORENTZ\_FORCE Lorentz force term

#### enum EqDataType

Enumerate variable that selects equation data type

#### Enumerator

INITIAL Initial condition

INITIAL\_FIELD Initial condition

**SOLUTION** Solution vector (same as Initial)

INITIAL\_AUX\_1 Initial auxiliary field

INITIAL\_AUX\_2 Initial auxiliary field

INITIAL\_AUX\_3 Initial auxiliary field

INITIAL\_AUX\_4 Initial auxiliary field

BOUNDARY\_CONDITION Boundary condition data

BODY\_FORCE Body force data

**SOURCE** Source data (same as Body force)

POINT\_FORCE Localized (at point) force

BOUNDARY\_FORCE Boundary force data

FLUX Flux data (same as Boundary force)

TRACTION Traction data (same as Boundary force)

AUX\_INPUT\_FIELD\_1 Auxiliary input field 1

AUX\_INPUT\_FIELD\_2 Auxiliary input field 2

AUX\_INPUT\_FIELD\_3 Auxiliary input field 3

AUX\_INPUT\_FIELD\_4 Auxiliary input field 4

DISPLACEMENT\_FIELD A displacement field

VELOCITY\_FIELD A velocity field

PRESSURE\_FIELD A pressure field

TEMPERATURE\_FIELD A temperature field

CONTACT\_DISTANCE Contact distance

#### enum Analysis

Selects Analysis type

Enumerator

STATIONARY Steady State analysis

STEADY\_STATE Steady state analysis

TRANSIENT Transient problem

TRANSIENT\_ONE\_STEP Transient problem, perform only one time step

**OPTIMIZATION** Optimization problem

EIGEN Eigenvalue problem

#### enum TimeScheme

Selects Time integration scheme

Enumerator

**NONE** No time integration scheme

FORWARD\_EULER Forward Euler scheme (Explicit)

BACKWARD\_EULER Backward Euler scheme (Implicit)

CRANK\_NICOLSON Crank-Nicolson scheme

**HEUN** Heun scheme

**NEWMARK** Newmark scheme

**LEAP\_FROG** Leap Frog scheme

ADAMS\_BASHFORTH Adams-Bashforth scheme (2nd Order)

AB2 Adams-Bashforth scheme (2nd Order)

RUNGE\_KUTTA 4-th Order Runge-Kutta scheme (4th Order)

RK4 4-th Order Runge-Kutta scheme

RK3\_TVD 3-rd Order Runge-Kutta TVD scheme

BDF2 Backward Difference Formula (2nd Order)

#### enum FEType

Choose Finite Element Type

#### Enumerator

FE\_2D\_3N 2-D elements, 3-Nodes (P1)

FE\_2D\_6N 2-D elements, 6-Nodes (P2)

FE\_2D\_4N 2-D elements, 4-Nodes (Q1)

FE\_3D\_AXI\_3N 3-D Axisymmetric elements, 3-Nodes (P1)

FE\_3D\_4N 3-D elements, 4-Nodes (P1)

FE\_3D\_8N 3-D elements, 8-Nodes (Q1)

### enum MatrixType

Choose matrix storage and type

#### Enumerator

**DENSE** Dense storage

SKYLINE Skyline storage

SPARSE Sparse storage

DIAGONAL Diagonal storage

TRIDIAGONAL Tridiagonal storage

**BAND** Band storage

SYMMETRIC Symmetric matrix

**UNSYMMETRIC** Unsymmetric matrix

IDENTITY Identity matrix

#### enum Iteration

Choose iterative solver for the linear system.

## Enumerator

DIRECT\_SOLVER Direct solver

CG\_SOLVER CG Method

CGS\_SOLVER CGS Metod

BICG\_SOLVER BiCG Method

BICG\_STAB\_SOLVER BiCGStab Method

GMRES\_SOLVER GMRes Method

#### enum Preconditioner

Choose preconditioner for the linear system.

#### Enumerator

IDENT\_PREC Identity (No preconditioning)

**DIAG\_PREC** Diagonal preconditioner

DILU\_PREC ILU (Incomplete factorization) preconditioner

ILU\_PREC DILU (Diagonal Incomplete factorization) preconditioner

SSOR\_PREC SSOR preconditioner

#### enum BCType

To select special boundary conditions.

#### Enumerator

**PERIODIC** A Periodic Boundary conditions (first side)

PERIODIC\_B Periodic Boundary conditions (second side)

CONTACT\_BC Contact Boundary conditions

SLIP Slip Boundary conditions

## enum IntegrationScheme

Choose numerical integration scheme

#### Enumerator

LEFT\_RECTANGLE Left rectangle integration formula

RIGHT\_RECTANGLE Right rectangle integration formula

MID\_RECTANGLE Midpoint (central) rectangle formula

TRAPEZOIDAL Trapezoidal rule

SIMPSON Simpson formula

GAUSS\_LEGENDRE Gauss-Legendre quadrature formulae

### 5.18.3 Function Documentation

#### T\_\* OFELI::A ( )

Return element matrix.

Matrix is returned as a C-array

## T\_\* OFELI::b ( )

Return element right-hand side.

Right-hand side is returned as a C-array

#### T\_\* OFELI::Prev ( )

Return element previous vector.

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

#### IOField ( const string & file, AccessType access, bool compact = true )

Constructor using file name.

#### **Parameters**

in	file	File name.	
in	access	Access code. This number is to be chosen among two enumerated values:	
		• IOField::IN to read the file	
		• IOField::OUT to write on it	
in	compact	Flag to choose a compact storage or not [Default: true]	

OFELI's Reference Guide

## IOField ( const string & $mesh\_file$ , const string & file, Mesh & ms, AccessType access, bool compact = true )

Constructor using file name, mesh file and mesh.

#### Parameters

in	mesh_file	File containing mesh	
in	file	File that contains field stored or to store	
in	ms	Mesh instance	
in	access	Access code. This number is to be chosen among two enumerated values:  • IOField::IN to read the file  • IOField::OUT to write on it	
in	compact	Flag to choose a compact storage or not [Default: true]	

## IOField ( const string & file, Mesh & ms, AccessType access, bool compact = true )

Constructor using file name and mesh.

#### Parameters

in	file	File that contains field stored or to store		
in	ms	Mesh instance		
in	access	Access code. This number is to be chosen among two enumerated values:  • IOField::IN to read the file  • IOField::OUT to write on it		
in	compact	Flag to choose a compact storage or not [Default: true]		

## IOField (const string & file, AccessType access, const string & name)

Constructor using file name and field name.

in	file	File that contains field stored or to store
in	access	Access code. This number is to be chosen among two enumerated values:
		• IOField::IN to read the file
		• IOField::OUT to write on it
in	name	Seek a specific field with given name

## void setMeshFile ( const string & file )

Set mesh file.

#### **Parameters**

in	file	Mesh file
----	------	-----------

## void open ( )

Open file.

Case where file name has been previously given (in the constructor).

## void open (const string & file, AccessType access)

Open file.

#### Parameters

in	file	File name.	
in	access	Access code. This number is to be chosen among two enumerated values:	
		• IOField::IN to read the file	
		• IOField::OUT to write on it	

## void put ( const Vect< real\_t > & v )

Store Vect instance v in file.

## Parameters

in	v	Vect instance to store

## $\operatorname{real\_t}$ get ( $\operatorname{Vect} < \operatorname{real\_t} > \&\ v$ )

Get Vect v instance from file.

First time step is read from the XML file.

## int get ( Vect< real\_t > & v, const string & name )

Get Vect v instance from file if the field has the given name.

First time step is read from the XML file.

in,out	v	Vect instance
in	name	Name to seek in the XML file

### int get ( DMatrix< real\_t> & A, const string & name )

Get DMatrix A instance from file if the field has the given name. First time step is read from the XML file.

#### **Parameters**

in,out	A	DMatrix instance
in	name	Name to seek in the XML file

## int get ( DSMatrix< real\_t > & A, const string & name )

Get DSMatrix A instance from file if the field has the given name. First time step is read from the XML file.

#### **Parameters**

in,out	A	DSMatrix instance
in	name	Name to seek in the XML file

#### int get ( Vect< real\_t > & v, real\_t t )

Get Vect v instance from file corresponding to a specific time value.

The sought vector corresponding to the time value is read from the XML file.

## Parameters

in,out	v	Vector instance
in	t	Time value

## void saveGMSH ( string output\_file, string mesh\_file )

Save field vectors in a file using **GMSH** format.

This member function enables avoiding the use of cfield. It must be used once all field vectors have been stored in output file. It closes this file and copies its contents to a **GMSH** file.

#### **Parameters**

in	output_file	Output file name where to store using <b>GMSH</b> format
in	mesh_file	File containing mesh data

## void setFile ( string file )

Set file name.

This function is to be used when the default constructor is invoked.

#### real\_t getValue ( string funct, real\_t v )

Return the calculated value of the function.

Case of a function of one variable

#### Parameters

in	funct	Name of the function to be evaluated, as read from input file
in	v	Value of the variable

#### Returns

Computed value of the function

## real\_t getDerivative ( string funct, real\_t v )

Return the derivative of the function at a given point.

Case of a function of one variable

#### Parameters

in	funct	Name of the function to be evaluated, as read from input file
in	v	Value of the variable

#### Returns

Derivative value

## real\_t getValue ( string funct, real\_t v1, real\_t v2 )

Return the calculated value of the function.

Case of a function of two variables

#### Parameters

in	funct	Name of the function to be evaluated, as read from input file
in	v1	Value of the first variable
in	<i>v</i> 2	Value of the second variable

#### Returns

Computed value of the function

## real\_t getValue ( string funct, real\_t v1, real\_t v2, real\_t v3 )

Return the calculated value of the function.

Case of a function of three variables

in	funct	Name of the funct to be evaluated, as read from input file

#### Parameters

in	v1	Value of the first variable
in	<i>v</i> 2	Value of the second variable
in	<i>v</i> 3	Value of the third variable

#### Returns

Computed value of the function

## SpMatrix ( )

Default constructor.

Initialize a zero-dimension matrix

## SpMatrix ( size\_t nr, size\_t nc )

Constructor that initializes current instance as a dense matrix. Normally, for a dense matrix this is not the right class.

#### **Parameters**

in	nr	Number of matrix rows.
in	пс	Number of matrix columns.

## SpMatrix ( size\_t size, int is\_diagonal = false )

Constructor that initializes current instance as a dense matrix. Normally, for a dense matrix this is not the right class.

#### **Parameters**

in	size	Number of matrix rows (and columns).
in	is_diagonal	Boolean argument to say is the matrix is actually a diagonal matrix or not.

## SpMatrix ( Mesh & mesh, size\_t dof = 0, int is\_diagonal = false )

Constructor using a Mesh instance.

in	mesh	Mesh instance from which matrix graph is extracted.
in	dof	Option parameter, with default value 0. dof=1 means that only one degree of freedom for each node (or element or side) is taken to determine matrix structure. The value dof=0 means that matrix structure is determined using all DOFs.
in	is_diagonal	Boolean argument to say is the matrix is actually a diagonal matrix or not.

## SpMatrix ( const Vect< RC > & I, int opt = 1 )

Constructor for a square matrix using non zero row and column indices.

#### **Parameters**

in	I	Vector containing pairs of row and column indices
in	opt	Flag indicating if vectors I is cleaned and ordered (opt=1) or not (opt=0). In the latter case, this vector can have the same contents more than once and are not necessarily ordered

## SpMatrix (const Vect< RC > & I, const Vect< $T_-$ > & a, int opt = 1)

Constructor for a square matrix using non zero row and column indices.

#### **Parameters**

in	I	Vector containing pairs of row and column indices
in	а	Vector containing matrix entries in the same order than the one given by I
in	opt	Flag indicating if vector I is cleaned and ordered (opt=1: default) or not (opt=0). In the latter case, this vector can have the same contents more than once and are not necessarily ordered

## SpMatrix ( size\_t nr, size\_t nc, const vector < size\_t > & row\_ptr, const vector < size\_t > & col\_ind )

Constructor for a rectangle matrix.

### Parameters

in	nr	Number of rows	
in	пс	Number of columns	
in	row_ptr	Vector of row pointers (See the above description of this class).	
in	col_ind	Vector of column indices (See the above description of this class).	

# SpMatrix ( size\_t nr, size\_t nc, const vector< size\_t > & row\_ptr, const vector< size\_t > & col\_ind, const vector< $T_-$ > & a )

Constructor for a rectangle matrix.

in	nr	Number of rows
in	пс	Number of columns
in	row_ptr	Vector of row pointers (See the above description of this class).
in	col_ind	Vector of column indices (See the above description of this class).

#### **Parameters**

in   a   vector instance containing matrix entries stored columnwise	
--	--

#### SpMatrix (const vector < size\_t > & row\_ptr, const vector < size\_t > & col\_ind)

Constructor for a rectangle matrix.

#### **Parameters**

in	row_ptr	Vector of row pointers (See the above description of this class).	]
in	col_ind	Vector of column indices (See the above description of this class).	]

## SpMatrix ( const vector< size\_t > & row\_ptr, const vector< size\_t > & col\_ind, const vector< $T_- > \& a$ )

Constructor for a rectangle matrix.

#### **Parameters**

in	row_ptr	Vector of row pointers (See the above description of this class).	
in	col_ind	Vector of column indices (See the above description of this class).	
in	а	vector instance that contain matrix entries stored row by row.  Number of rows is extracted from vector row_ptr.	

## void Laplace1D ( size\_t n, real\_t h )

Sets the matrix as the one for the Laplace equation in 1-D.

The matrix is initialized as the one resulting from  $P_1$  finite element discretization of the classical elliptic operator -u'' = f with homogeneous Dirichlet boundary conditions

## Remarks

This function is available for real valued matrices only.

#### **Parameters**

in	n	Size of matrix (Number of rows)
in	h	Mesh size (assumed constant)

#### void Laplace2D ( size\_t nx, size\_t ny )

Sets the matrix as the one for the Laplace equation in 2-D.

The matrix is initialized as the one resulting from  $P_1$  finite element discretization of the classical elliptic operator -Delta u = f with homogeneous Dirichlet boundary conditions

#### Remarks

This function is available for real valued matrices only.

#### **Parameters**

in	nx	Number of unknowns in the x-direction
in	ny	Number of unknowns in the y-direction

#### Remarks

The number of rows is equal to nx\*ny

#### void setMesh ( Mesh & mesh, size\_t dof = 0 )

Determine mesh graph and initialize matrix.

This member function is called by constructor with the same arguments

#### **Parameters**

in	mesh	Mesh instance for which matrix graph is determined.	
in	dof	Option parameter, with default value 0. dof=1 means that only one degree of freedom for each node (or element or side) is taken to determine matrix structure. The value dof=0 means that matrix structure is determined using all DOFs.	

## void DiagPrescribe (Mesh & mesh, Vect< T $_->$ & b, const Vect< T $_->$ & u)

Impose by a diagonal method an essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **set**← **Penal**(..).

#### **Parameters**

in	mesh	Mesh instance from which information is extracted.	
in,out	b	Vect instance that contains right-hand side.	
in	и	Vect instance that conatins imposed valued at DOFs where they are to be imposed.	

## void DiagPrescribe ( Vect< T $_->$ & b, const Vect< T $_->$ & u)

Impose by a diagonal method an essential boundary condition using the Mesh instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **set**←

## Penal(..).

#### Parameters

in,out	b	Vect instance that contains right-hand side.	
in	и	Vect instance that conatins imposed valued at DOFs where they are to be imposed.	

## void setSize ( size\_t size )

Set size of matrix (case where it's a square matrix).

#### Parameters

in	size	Number of rows and columns.
----	------	-----------------------------

## void setSize ( size\_t nr, size\_t nc )

Set size (number of rows) of matrix.

#### Parameters

in	nr	Number of rows
in	пс	Number of columns

## void setGraph ( const Vect < RC > & I, int opt = 1 )

Set graph of matrix by giving a vector of its nonzero entries.

#### Parameters

in	I	Vector containing pairs of row and column indices
in	opt	Flag indicating if vector I is cleaned and ordered (opt=1: default) or not (opt=0). In the latter case, this vector can have the same contents more than once and are not necessarily ordered

## T\_& operator() ( size\_t i, size\_t j ) [virtual]

Operator () (Non constant version)

## Parameters

in	i	Row index
in	j	Column index

Implements Matrix  $< T_- >$ .

## $T_{-}$ operator() ( size\_t i, size\_t j ) const [virtual]

Operator () (Constant version)

#### **Parameters**

in	i	Row index
in	j	Column index

Implements Matrix  $< T_- >$ .

## $T_{-}$ operator() ( size\_t i ) const

Operator () with one argument (Constant version)

Returns i-th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

## $T_{-}$ operator[] ( size\_t i ) const

Operator [] (Constant version).

Returns i-th position in the array storing matrix entries. The first entry is at location 0. Entries are stored row by row.

## Vect<T $_->$ operator\* ( const Vect< T $_->$ & x ) const

Operator \* to multiply matrix by a vector.

#### **Parameters**

	in	x	Vect instance to multiply by
--	----	---	------------------------------

## Returns

Vector product of matrix by x

## SpMatrix<T $_->$ & operator\*= ( const T $_-$ & a )

Operator \*= to premultiply matrix by a constant.

#### Parameters

in	а	Constant to multiply matrix by
----	---	--------------------------------

#### Returns

Resulting matrix

## void Mult ( const Vect< $T_- > \& x$ , Vect< $T_- > \& y$ ) const [virtual]

Multiply matrix by vector and save in another one.

#### Parameters

in	x	Vector to multiply by matrix
out	y	Vector that contains on output the result.

Implements Matrix  $< T_- >$ .

void MultAdd ( const Vect<  $T_-$  > & x, Vect<  $T_-$  > & y ) const [virtual]

Multiply matrix by vector x and add to y.

#### Parameters

in	x	Vector to multiply by matrix
out	y	Vector to add to the result. y contains on output the result.

Implements Matrix  $< T_- >$ .

void MultAdd (  $T_-a$ , const Vect<  $T_-> \& x$ , Vect<  $T_-> \& y$  ) const [virtual]

Multiply matrix by vector a\*x and add to y.

#### Parameters

in	а	Constant to multiply by matrix
in	x	Vector to multiply by matrix
out	y	Vector to add to the result. y contains on output the result.

Implements Matrix  $< T_- >$ .

void TMult ( const Vect<  $T_- > \& x$ , Vect<  $T_- > \& y$  ) const [virtual]

Multiply transpose of matrix by vector x and save in y.

## Parameters

in	x	Vector to multiply by matrix
out	y	Vector that contains on output the result.

Implements Matrix  $< T_- >$ .

void Axpy (  $T_-a$ , const SpMatrix<  $T_- > \& m$  )

Add to matrix the product of a matrix by a scalar.

in	а	Scalar to premultiply	
in	m	Matrix by which a is multiplied. The result is added to current instance	

#### void Axpy ( $T_-a$ , const Matrix< $T_- > * m$ ) [virtual]

Add to matrix the product of a matrix by a scalar.

#### Parameters

in	а	Scalar to premultiply	
in	m	Pointer to Matrix by which a is multiplied. The result is added to current instance	

Implements Matrix  $< T_- >$ .

## void set ( size\_t i, size\_t j, const T\_ & val ) [virtual]

Assign a value to an entry of the matrix.

#### **Parameters**

in	i	Row index
in	j	Column index
in	val	Value to assign to a(i,j)

Implements Matrix  $< T_- >$ .

## void add ( size\_t i, size\_t j, const T\_ & val ) [virtual]

Add a value to an entry of the matrix.

#### Parameters

in	i	Row index
in	j	Column index
in	val	Constant value to add to a(i,j)

Implements Matrix  $< T_- >$ .

### void operator= ( const $T_- \& x$ )

Operator =.

Assign constant value x to all matrix entries.

## size\_t getColInd ( size\_t i ) const [virtual]

Return storage information.

Returns

Column index of the i-th stored element in matrix

Reimplemented from Matrix $< T_- >$ .

int solve ( const Vect< T $_->$  & b, Vect< T $_->$  & x, bool fact = false ) [virtual]

Solve the linear system of equations.

The default parameters are:

- CG\_SOLVER for solver
- DIAG\_PREC for preconditioner
- Max. Number of iterations is 1000
- Tolerance is 1.e-8

To change these values, call function setSolver before this function

#### **Parameters**

in	b	Vector that contains right-hand side
out	x	Vector that contains the obtained solution
in	fact	Unused argument

#### Returns

Number of actual performed iterations

Implements Matrix $< T_- >$ .

void setSolver ( Iteration  $solver = CG\_SOLVER$ , Preconditioner  $prec = DIAG\_PREC$ , int  $max\_it = 1000$ , real\_t toler = 1.e-8)

Choose solver and preconditioner for an iterative procedure.

in	solver	Option to choose iterative solver in an enumerated variable	
		CG_SOLVER: Conjugate Gradient [default]	
		CGS_SOLVER: Squared conjugate gradient	
		BICG_SOLVER: Biconjugate gradient	
		BICG_STAB_SOLVER: Biconjugate gradient stabilized	
		GMRES_SOLVER: Generalized Minimal Residual	
		Default value is CG_SOLVER	
in	prec	Option to choose preconditioner in an enumerated variable	
		IDENT_PREC: Identity preconditioner (no preconditioning)	
		DIAG_PREC: Diagonal preconditioner [default]	
		SSOR_PREC: SSOR (Symmetric Successive Over Relaxation) preconditioner	
		DILU_PREC: ILU (Diagonal Incomplete factorization) preconditioner	
		<ul> <li>ILU_PREC: ILU (Incomplete factorization) preconditioner</li> </ul>	
		Default value is DIAG_PREC	
in	max↔ _it	Maximum number of allowed iterations. Default value is 1000.	

#### Parameters

in	toler	Tolerance for convergence. Default value is 1.e-8
----	-------	---

## $T_-*$ get ( ) const

#### Return C-Array.

Non zero terms of matrix is stored row by row.

## T\_get( size\_t i, size\_t j ) const [virtual]

Return entry (i,j) of matrix if this one is stored, 0 otherwise.

#### **Parameters**

in	i	Row index (Starting from 1)	
in	j	Column index (Starting from 1)	

Implements Matrix  $< T_- >$ .

## TrMatrix ( )

Default constructor.

Initialize a zero dimension tridiagonal matrix

## void Laplace1D ( real\_t h )

Define matrix as the one of 3-point finite difference discretization of the second derivative.

## Parameters

in	h	mesh size

## void setSize ( size\_t size )

Set size (number of rows) of matrix.

#### Parameters

in	size	Number of rows and columns.
----	------	-----------------------------

## void Axpy ( $T_-a$ , const TrMatrix< $T_- > \& m$ )

Add to matrix the product of a matrix by a scalar.

in	а	Scalar to premultiply	
in	m	Matrix by which a is multiplied. The result is added to current instance	

## void Axpy ( $T_-a_r$ const Matrix $< T_- > * m$ ) [virtual]

Add to matrix the product of a matrix by a scalar.

#### Parameters

in	а	Scalar to premultiply	
in	m	Matrix by which a is multiplied. The result is added to current instance	

Implements Matrix  $< T_- >$ .

## T\_operator() ( size\_t i, size\_t j ) const [virtual]

Operator () (Constant version).

#### Parameters

in	i	Row index
in	j	Column index

Implements Matrix  $< T_- >$ .

## T\_& operator() ( size\_t i, size\_t j ) [virtual]

Operator () (Non constant version).

### Parameters

in	i	Row index
in	j	Column index

Implements Matrix  $< T_- >$ .

## TrMatrix<T $_->$ & operator= ( const TrMatrix< T $_->$ & m )

Operator =.

Copy matrix m to current matrix instance.

## TrMatrix<T $_->$ & operator\*= ( const T $_-$ & x )

Operator \*=.

Premultiply matrix entries by constant value x.

## int solve ( Vect< T $_->$ & b, bool fact = true ) [virtual]

Solve a linear system with current matrix (forward and back substitution).

in,out	b	Vect instance that contains right-hand side on input and solution on output.
in	fact	Ununsed argument

#### Returns

- 0 if solution was normally performed,
- n if the n-th pivot is null.

Warning: Matrix is modified after this function.

Implements Matrix $< T_- >$ .

## int solve ( const Vect< T $_->$ & b, Vect< T $_->$ & x, bool fact = false ) [virtual]

Solve a linear system with current matrix (forward and back substitution).

#### Parameters

in	b	Vect instance that contains right-hand side.	
out	x	Vect instance that contains solution.	
in	fact	Unused argument	

#### Returns

- 0 if solution was normally performed,
- n if the n-th pivot is null.

Warning: Matrix is modified after this function.

Implements Matrix $< T_->$ .

## Grid ( real\_t xm, real\_t xM, size\_t npx )

Construct a 1-D structured grid given its extremal coordinates and number of intervals.

#### **Parameters**

in	xm	Minimal value for x
in	xM	Maximal value for x
in	прх	Number of grid intervals in the x-direction

## Grid ( real\_t xm, real\_t xM, real\_t ym, real\_t yM, size\_t npx, size\_t npy )

Construct a 2-D structured grid given its extremal coordinates and number of intervals.

in	xm	Minimal value for x
in	хM	Maximal value for x
in	ут	Minimal value for y
in	yМ	Maximal value for y
in	прх	Number of grid intervals in the x-direction
in	пру	Number of grid intervals in the y-direction

## Grid ( Point < real\_t > m, Point < real\_t > M, size\_t npx, size\_t npx,

Construct a 2-D structured grid given its extremal coordinates and number of intervals.

#### Parameters

in	m	Minimal coordinate value
in	M	Maximal coordinate value
in	прх	Number of grid intervals in the x-direction
in	пру	Number of grid intervals in the y-direction

## Grid ( real\_t xm, real\_t xm, real\_t ym, real\_t ym, real\_t zm, real\_t zm, size\_t npx, size\_t npx)

Construct a 3-D structured grid given its extremal coordinates and number of intervals.

#### Parameters

in	xm	Minimal value for x
in	хM	Maximal value for x
in	ym	Minimal value for y
in	уМ	Maximal value for y
in	zm	Minimal value for z
in	zM	Maximal value for z
in	прх	Number of grid intervals in the x-direction
in	пру	Number of grid intervals in the y-direction
in	npz	Number of grid intervals in the z-direction

## Grid ( Point < real\_t > m, Point < real\_t > M, size\_t npx, size\_t npy, size\_t npz)

Construct a 3-D structured grid given its extremal coordinates and number of intervals.

#### Parameters

in	m	Minimal coordinate value
in	M	Maximal coordinate value
in	прх	Number of grid intervals in the x-direction
in	пру	Number of grid intervals in the y-direction
in	npz	Number of grid intervals in the z-direction

## void setXMin ( const Point < real\_t > & x )

Set min. coordinates of the domain.

in	x	Minimal values of coordinates

#### void setXMax ( const Point < real\_t > & x )

Set max. coordinates of the domain.

#### Parameters

in x Maximal values of coordin
--------------------------------

## void setDomain ( real\_t xmin, real\_t xmax )

Set Dimensions of the domain: 1-D case.

#### **Parameters**

in	xmin	Minimal value of x-coordinate
in	xmax	Maximal value of x-coordinate

## void setDomain ( real\_t xmin, real\_t xmax, real\_t ymin, real\_t ymax )

Set Dimensions of the domain: 2-D case.

#### Parameters

in	xmin	Minimal value of x-coordinate
in	xmax	Maximal value of x-coordinate
in	ymin	Minimal value of y-coordinate
in	ymax	Maximal value of y-coordinate

## void setDomain ( real\_t xmin, real\_t xmax, real\_t ymin, real\_t ymin, real\_t zmin, real\_t zmax)

Set Dimensions of the domain: 3-D case.

#### Parameters

in	xmin	Minimal value of x-coordinate
in	xmax	Maximal value of x-coordinate
in	ymin	Minimal value of y-coordinate
in	ymax	Maximal value of y-coordinate
in	zmin	Minimal value of z-coordinate
in	zmax	Maximal value of z-coordinate

## void setDomain ( Point< real\_t > xmin, Point< real\_t > xmax )

Set Dimensions of the domain: 3-D case.

#### **Parameters**

in	xmin	Minimal coordinate value
in	xmax	Maximal coordinate value

## void setN ( size\_t nx, size\_t ny = 0, size\_t nz = 0 )

Set number of grid intervals in the x, y and z-directions. Number of points is the number of intervals plus one in each direction

#### Parameters

in	nx	Number of grid intervals in the x-direction
in	ny	Number of grid intervals in the y-direction (Default=0: 1-D grid)
in	nz	Number of grid intervals in the z-direction (Default=0: 1-D or 2-D grid)

#### Remarks

: The size of the grid (xmin and xmax) must have been defined before.

## size\_t getNy ( ) const

Return number of grid intervals in the y-direction. ny=0 for 1-D domains (segments)

## size\_t getNz ( ) const

Return number of grid intervals in the z-direction. nz=0 for 1-D (segments) and 2-D domains (rectangles)

## void setCode ( string exp, int code )

Set a code for some grid points.

#### **Parameters**

in	exp	Regular expression that determines the set of grid points on which the code is applied.	
in	code	Code to assign.	

## void setCode ( int side, int code )

Set a code for grid points on sides.

in	side	Side for which code is assigned. Possible values are: MIN_X, MAX_X, MIN_Y, MAX_Y, MIN_Z, MAX_Z
in	code	Code to assign.

## int getCode ( int side ) const

Return code for a side number.

#### **Parameters**

in	side	Side for which code is returned. Possible values are: MIN_X, MAX_X, MIN_Y, MAX_Y,	
		MIN_Z, MAX_Z	

## int getCode ( size\_t i, size\_t j ) const

Return code for a grid point.

#### Parameters

in	i	i-th index for node for which code is to be returned.
in	j	j-th index for node for which code is to be returned.

## int getCode ( size\_t i, size\_t j, size\_t k ) const

Return code for a grid point.

#### Parameters

in	i	i-th index for node for which code is to be returned.
in	j	j-th index for node for which code is to be returned.
in	k	k-th index for node for which code is to be returned.

## void Deactivate ( size\_t i )

Change state of a cell from active to inactive (1-D grid)

## Parameters

in	i	grid cell to remove
----	---	---------------------

## void Deactivate ( size\_t i, size\_t j )

Change state of a cell from active to inactive (2-D grid)

in	i	i-th index for grid cell to remove. If this value is 0, all cells (*, j) are deactivated
in	j	j-th index for grid cell to remove If this value is 0, all cells (i,*) are deactivated

## Remarks

if i and j have value 0 all grid cells are deactivated !!

## void Deactivate ( $size_t i$ , $size_t j$ , $size_t k$ )

Change state of a cell from active to inactive (2-D grid)

#### **Parameters**

in	i	i-th index for grid cell to remove. If this value is 0, all cells (*,j,k) are deactivated
in	j	j-th index for grid cell to remove If this value is 0, all cells (i,*,k) are deactivated
in	k	k-th index for grid cell to remove If this value is 0, all cells (i,j,*) are deactivated

## int is Active ( $size_t i$ ) const

Say if cell is active or not (1-D grid)

#### Parameters

in i	Index of cell
------	---------------

#### Returns

1 if cell is active, 0 if not

## int isActive ( size\_t i, size\_t j ) const

Say if cell is active or not (2-D grid)

#### Parameters

i	n	i	i-th index of cell	L
i	n	j	j-th index of cell	L

#### Returns

1 if cell is active, 0 if not

## int isActive ( size\_t i, size\_t j, size\_t k ) const

Say if cell is active or not (3-D grid)

in	i	i-th index of cell
in	j	j-th index of cell
in	k	k-th index of cell

#### Returns

1 if cell is active, 0 if not

#### Iter ( )

Default Constructor.

This constructor set default values: the maximal number of iterations is set to 100 and the tolerance to 1.e-8

## Iter ( int max\_it, real\_t toler )

Constructor with iteration parameters.

#### **Parameters**

in	max↔ _it	Maximum number of iterations
in	toler	Tolerance value for convergence

## bool check ( Vect< $T_-$ > & u, const Vect< $T_-$ > & v, int opt = 2)

Check convergence.

#### Parameters

in,out	и	Solution vector at previous iteration
in	v	Solution vector at current iteration
in	opt	Vector norm for convergence checking 1: 1-norm, 2: 2-norm, 0: Max. norm [Default: 2]

## Returns

true if convergence criterion is satisfied, false if not

After checking, this function copied v into u.

## bool check ( $T_- \& u$ , const $T_- \& v$ )

Check convergence for a scalar case (one equation)

#### **Parameters**

in,out	и	Solution at previous iteration
in	v	Solution at current iteration

## Returns

true if convergence criterion is satisfied, false if not

After checking, this function copied v into u.

## Chapter 6

# Namespace Documentation

## 6.1 OFELI Namespace Reference

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

#### Classes

class AbsEqua

Mother abstract class to describe equation.

• class Bar2DL2

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

class Beam3DL2

To build element equations for 3-D beam equations using 2-node lines.

• class BiotSavart

Class to compute the magnetic induction from the current density using the Biot-Savart formula.

class BMatrix

To handle band matrices.

• class Brick

To store and treat a brick (parallelepiped) figure.

• class Circle

To store and treat a circular figure.

class DC1DL2

 $Builds\ finite\ element\ arrays\ for\ thermal\ diffusion\ and\ convection\ in\ 1-D\ using\ 2-Node\ elements.$ 

• class DC2DT3

Builds finite element arrays for thermal diffusion and convection in 2-D domains using 3-Node triangles.

• class DC2DT6

Builds finite element arrays for thermal diffusion and convection in 2-D domains using 6-Node triangles.

• class DC3DAT3

Builds finite element arrays for thermal diffusion and convection in 3-D domains with axisymmetry using 3-Node triangles.

• class DC3DT4

Builds finite element arrays for thermal diffusion and convection in 3-D domains using 4-Node tetrahedra.

class DG

Enables preliminary operations and utilities for the Discontinous Galerkin method.

• class DMatrix

To handle dense matrices.

• class Domain

To store and treat finite element geometric information.

class DSMatrix

To handle symmetric dense matrices.

• class EC2D1T3

Eddy current problems in 2-D domains using solenoidal approximation.

• class EC2D2T3

Eddy current problems in 2-D domains using transversal approximation.

class Edge

To describe an edge.

class EdgeList

Class to construct a list of edges having some common properties.

• class EigenProblemSolver

Class to find eigenvalues and corresponding eigenvectors of a given matrix in a generalized eigenproblem, i.e. Find scalars l and non-null vectors v such that  $[K]\{v\} = l[M]\{v\}$  where [K] and [M] are symmetric matrices. The eigenproblem can be originated from a PDE. For this, we will refer to the matrices K and M as Stiffness and Mass matrices respectively.

• class Elas2DQ4

To build element equations for 2-D linearized elasticity using 4-node quadrilaterals.

• class Elas2DT3

To build element equations for 2-D linearized elasticity using 3-node triangles.

class Elas3DH8

To build element equations for 3-D linearized elasticity using 8-node hexahedra.

class Elas3DT4

To build element equations for 3-D linearized elasticity using 4-node tetrahedra.

class Element

To store and treat finite element geometric information.

class ElementList

Class to construct a list of elements having some common properties.

• class Ellipse

To store and treat an ellipsoidal figure.

class Equa\_Electromagnetics

Abstract class for Electromagnetics Equation classes.

• class Equa\_Fluid

Abstract class for Fluid Dynamics Equation classes.

class Equa\_Laplace

Abstract class for classes about the Laplace equation.

class Equa\_Porous

Abstract class for Porous Media Finite Element classes.

• class Equa\_Solid

Abstract class for Solid Mechanics Finite Element classes.

class Equa\_Therm

Abstract class for Heat transfer Finite Element classes.

• class Equation

Abstract class for all equation classes.

• class Estimator

To calculate an a posteriori estimator of the solution.

class FastMarching2D

To run a Fast Marching Method on 2-D structured uniform grids.

• class FEShape

Parent class from which inherit all finite element shape classes.

• class Figure

To store and treat a figure (or shape) information.

class FMM2D

class for the fast marching 2-D algorithm

• class FMM3D

class for the 3-D fast marching algorithm

• class FMMSolver

The Fast Marching Method solver.

class Funct

A simple class to parse real valued functions.

class Gauss

Calculate data for Gauss integration.

class Grid

To manipulate structured grids.

class HelmholtzBT3

Builds finite element arrays for Helmholtz equations in a bounded media using 3-Node triangles.

class Hexa8

Defines a three-dimensional 8-node hexahedral finite element using Q1-isoparametric interpolation.

• class ICPG1D

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 1-D.

class ICPG2DT

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 2-D.

class ICPG3DT

 ${\it Class\ to\ solve\ the\ Inviscid\ compressible\ fluid\ flows\ (Euler\ equations)\ for\ perfect\ gas\ in\ 3-D.}$ 

• class Integration

Class for numerical integration methods.

• class IOField

Enables working with files in the XML Format.

• class IPF

To read project parameters from a file in IPF format.

• class Iter

Class to drive an iterative process.

• class Laplace1DL2

To build element equation for a 1-D elliptic equation using the 2-Node line element  $(P_1)$ .

• class Laplace1DL3

To build element equation for the 1-D elliptic equation using the 3-Node line  $(P_2)$ .

• class Laplace2DT3

To build element equation for the Laplace equation using the 2-D triangle element  $(P_1)$ .

• class Laplace2DT6

To build element equation for the Laplace equation using the 2-D triangle element  $(P_2)$ .

• class LaplaceDG2DP1

To build and solve the linear system for the Poisson problem using the DG P<sub>1</sub> 2-D triangle element.

• class LCL1D

Class to solve the linear conservation law (Hyperbolic equation) in 1-D by a MUSCL Finite Volume scheme.

class LCL2DT

Class to solve the linear hyperbolic equation in 2-D by a MUSCL Finite Volume scheme on triangles.

class LCL3DT

Class to solve the linear conservation law equation in 3-D by a MUSCL Finite Volume scheme on tetrahedra.

class Line2

To describe a 2-Node planar line finite element.

• class Line3

*To describe a 3-Node quadratic planar line finite element.* 

• class LinearSolver

Class to solve systems of linear equations by iterative methods.

class LocalMatrix

Handles small size matrices like element matrices, with a priori known size.

class LocalVect

Handles small size vectors like element vectors.

class Material

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

class Matrix

Virtual class to handle matrices for all storage formats.

class Mesh

To store and manipulate finite element meshes.

class MeshAdapt

To adapt mesh in function of given solution.

• class Muscl

Parent class for hyperbolic solvers with Muscl scheme.

class Muscl1D

Class for 1-D hyperbolic solvers with Muscl scheme.

• class Muscl2DT

Class for 2-D hyperbolic solvers with Muscl scheme.

• class Muscl3DT

Class for 3-D hyperbolic solvers with Muscl scheme using tetrahedra.

class MyNLAS

Abstract class to define by user specified function.

class MyOpt

Abstract class to define by user specified optimization function.

class NLASSolver

To solve a system of nonlinear algebraic equations of the form f(u) = 0.

• class Node

To describe a node.

• class NodeList

Class to construct a list of nodes having some common properties.

• class NSP2DQ41

Builds finite element arrays for incompressible Navier-Stokes equations in 2-D domains using  $Q_1/P_0$  element and a penaly formulation for the incompressibility condition.

• class ODESolver

To solve a system of ordinary differential equations.

• class OFELIException

To handle exceptions in OFELI.

class OptSolver

To solve an optimization problem with bound constraints.

• class Partition

To partition a finite element mesh into balanced submeshes.

• class Penta6

Defines a 6-node pentahedral finite element using  $P_1$  interpolation in local coordinates (s.x, s.y) and  $Q_1$  isoparametric interpolation in local coordinates (s.x, s.z) and (s.y, s.z).

class PhaseChange

This class enables defining phase change laws for a given material.

• class Point

Defines a point with arbitrary type coordinates.

• class Point2D

Defines a 2-D point with arbitrary type coordinates.

class Polygon

To store and treat a polygonal figure.

• class Prec

To set a preconditioner.

• class Prescription

To prescribe various types of data by an algebraic expression. Data may consist in boundary conditions, forces, tractions, fluxes, initial condition. All these data types can be defined through an enumerated variable.

class Quad4

Defines a 4-node quadrilateral finite element using  $Q_1$  isoparametric interpolation.

• class Reconstruction

To perform various reconstruction operations.

• class Rectangle

To store and treat a rectangular figure.

• class Side

To store and treat finite element sides (edges in 2-D or faces in 3-D)

class SideList

Class to construct a list of sides having some common properties.

class SkMatrix

To handle square matrices in skyline storage format.

class SkSMatrix

To handle symmetric matrices in skyline storage format.

class Sphere

To store and treat a sphere.

• class SpMatrix

To handle matrices in sparse storage format.

• class SteklovPoincare2DBE

Solver of the Steklov Poincare problem in 2-D geometries using piecewie constant boundary elemen.

• class Tabulation

To read and manipulate tabulated functions.

#### • class Tetra4

Defines a three-dimensional 4-node tetrahedral finite element using  $P_1$  interpolation.

• class Timer

To handle elapsed time counting.

• class TimeStepping

To solve time stepping problems, i.e. systems of linear ordinary differential equations of the form  $[A2]\{y''\}+[A1]\{y'\}+[A0]\{y\}=\{b\}.$ 

• class TINS2DT3S

Builds finite element arrays for transient incompressible fluid flow using Navier-Stokes equations in 2-D domains. Numerical approximation uses stabilized 3-node triangle finite elements for velocity and pressure. 2nd-order projection scheme is used for time integration.

class TINS3DT4S

Builds finite element arrays for transient incompressible fluid flow using Navier-Stokes equations in 3-D domains. Numerical approximation uses stabilized 4-node tatrahedral finite elements for velocity and pressure. 2nd-order projection scheme is used for time integration.

• class Triang3

Defines a 3-Node (P1) triangle.

• class Triang6S

Defines a 6-Node straight triangular finite element using P2 interpolation.

class Triangle

To store and treat a triangle.

• class triangle

Defines a triangle. The reference element is the rectangle triangle with two unit edges.

• class TrMatrix

To handle tridiagonal matrices.

• class Vect

To handle general purpose vectors.

• class WaterPorous2D

To solve water flow equations in porous media (1-D)

### **Enumerations**

```
• enum PDE_Terms {
  CONSISTENT\_MASS = 0x00001000,
 LUMPED\_MASS = 0x00002000,
 MASS = 0x00002000,
 CAPACITY = 0x00004000,
 CONSISTENT_CAPACITY = 0x00004000,
 LUMPED_CAPACITY = 0x00008000,
  VISCOSITY = 0x00010000,
  STIFFNESS = 0x00020000,
 DIFFUSION = 0x00040000,
  MOBILITY = 0x00040000,
 CONVECTION = 0x00080000,
 DEVIATORIC = 0x00100000,
 DILATATION = 0x00200000,
 ELECTRIC = 0x00400000,
 MAGNETIC = 0x008000000,
 LOAD = 0x010000000
 HEAT\_SOURCE = 0x020000000,
  BOUNDARY_TRACTION = 0x04000000,
 HEAT_FLUX = 0x080000000,
 CONTACT = 0x100000000,
 BUOYANCY = 0x200000000,
 LORENTZ_FORCE = 0x40000000 }
enum EqDataType {
 INITIAL = 1,
 INITIAL_FIELD = 1,
 SOLUTION = 1,
 INITIAL_AUX_1 = 2,
 INITIAL_AUX_2 = 3,
 INITIAL_AUX_3 = 4,
 INITIAL_AUX_4 = 5,
  BOUNDARY_CONDITION = 6,
 BODY\_FORCE = 7,
 SOURCE = 7,
 POINT_FORCE = 8,
 BOUNDARY\_FORCE = 9,
 FLUX = 9,
 TRACTION = 9,
  AUX_INPUT_FIELD_1 = 10,
  AUX_INPUT_FIELD_2 = 11,
  AUX_INPUT_FIELD_3 = 11,
  AUX_INPUT_FIELD_4 = 12,
 DISPLACEMENT_FIELD = 13,
  VELOCITY_FIELD = 14,
 PRESSURE_FIELD = 15,
  TEMPERATURE_FIELD = 16,
 CONTACT_DISTANCE = 17 }
```

• enum Analysis {

```
STATIONARY = 0,
  STEADY\_STATE = 0,
 TRANSIENT = 1,
  TRANSIENT_ONE_STEP = 2,
  OPTIMIZATION = 3,
 EIGEN = 4
• enum TimeScheme {
 NONE = 0,
 FORWARD_EULER = 1,
 BACKWARD\_EULER = 2,
 CRANK_NICOLSON = 3,
 HEUN = 4,
 NEWMARK = 5,
 LEAP\_FROG = 6,
  ADAMS\_BASHFORTH = 7,
  AB2 = 7,
  RUNGE\_KUTTA = 8,
 RK4 = 8,
 RK3_TVD = 9
 BDF2 = 10 }
enum FEType {
 FE_2D_3N,
 FE_2D_6N,
 FE_2D_4N,
 FE_3D_AXI_3N,
 FE_3D_4N,
 FE_3D_8N }
enum MatrixType {
 DENSE = 1,
 SKYLINE = 2,
 SPARSE = 4,
 DIAGONAL = 8,
  TRIDIAGONAL = 16,
  BAND = 32
 SYMMETRIC = 64,
 UNSYMMETRIC = 128,
 IDENTITY = 256 }
enum Iteration {
 DIRECT\_SOLVER = 0,
 CG\_SOLVER = 1,
 CGS\_SOLVER = 2,
  BICG\_SOLVER = 3,
  BICG_STAB_SOLVER = 4,
 GMRES_SOLVER = 5 }
    Choose iterative solver for the linear system.
• enum Preconditioner {
 IDENT\_PREC = 0,
 DIAG_PREC = 1,
  DILU\_PREC = 2,
 ILU\_PREC = 3,
 SSOR\_PREC = 4
     Choose preconditioner for the linear system.
```

180

OFELI's Reference Guide

```
• enum NormType {
     NORM1,
     WNORM1,
     NORM2,
     WNORM2,
     NORM_MAX }

    enum {

     BY_VALUE,
     BY_POSITION,
     BY_FIELD }
   • enum BCType {
     PERIODIC_A = 9999,
     PERIODIC_B = -9999,
     CONTACT_BC = 9998,
     CONTACT_M = 9997,
     CONTACT_S = -9997,
     SLIP = 9996 }
   • enum IntegrationScheme {
     LEFT_RECTANGLE = 0,
     RIGHT_RECTANGLE = 1,
     MID_RECTANGLE = 2,
     TRAPEZOIDAL = 3,
     SIMPSON = 4,
     GAUSS_LEGENDRE = 5 }
Functions
   • ostream & operator << (ostream &s, const Muscl3DT &m)
        Output mesh data as calculated in class Muscl3DT.
   • T_*A()
        Return element matrix.
   • T<sub>-</sub> * b ()
        Return element right-hand side.
   • T_ * Prev ()
        Return element previous vector.
   • ostream & operator << (ostream &s, const complex_t &x)
        Output a complex number.
   • ostream & operator << (ostream &s, const std::string &c)
        Output a string.
   • template<class T_>
     ostream & operator << (ostream &s, const vector < T_- > &v)
        Output a vector instance.
   • template<class T_>
     ostream & operator << (ostream &s, const std::list < T_> &l)
        Output a vector instance.
   • template<class T_>
     ostream & operator << (ostream &s, const std::pair < T_-, T_- > &a)
        Output a pair instance.

    void saveField (Vect< real_t > &v, string output_file, int opt)
```

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

- void saveField (const Vect< real\_t > &v, const Mesh &mesh, string output\_file, int opt)
   Save a vector to an output file in a given file format.
- void saveField (Vect< real\_t > &v, const Grid &g, string output\_file, int opt)

  Save a vector to an output file in a given file format, for a structured grid data.
- void saveGnuplot (string input\_file, string output\_file, string mesh\_file, int f=1)

  Save a vector to an input Gnuplot file.
- void saveGnuplot (Mesh &mesh, string input\_file, string output\_file, int f=1) Save a vector to an input Gnuplot file.
- void saveTecplot (string input\_file, string output\_file, string mesh\_file, int f=1)

  Save a vector to an output file to an input Tecplot file.
- void saveTecplot (Mesh &mesh, string input\_file, string output\_file, int f=1)

  Save a vector to an output file to an input Tecplot file.
- void saveVTK (string input\_file, string output\_file, string mesh\_file, int f=1)
   Save a vector to an output VTK file.
- void saveVTK (Mesh &mesh, string input\_file, string output\_file, int f=1)

  Save a vector to an output VTK file.
- void saveGmsh (string input\_file, string output\_file, string mesh\_file, int f=1)

  Save a vector to an output Gmsh file.
- void saveGmsh (Mesh &mesh, string input\_file, string output\_file, int f=1) Save a vector to an output Gmsh file.
- ostream & operator<< (ostream &s, const Tabulation &t)</li>

Output Tabulated function data.

- template < class T\_, size\_t N\_, class E\_ >
   void element\_assembly (const E\_ &e, const LocalVect < T\_, N\_ > &be, Vect < T\_ > &b)
   Assemble local vector into global vector.
- template < class T\_, size\_t N\_, class E\_ > void element\_assembly (const E\_ &e, const LocalMatrix < T\_, N\_, N\_ > &ae, Vect < T\_ > &b)
   Assemble diagonal local vector into global vector.
- template < class  $T_-$ , size\_t  $N_-$ , class  $E_-$  > void element\_assembly (const  $E_-$  &e, const LocalMatrix <  $T_-$ ,  $N_-$ ,  $N_-$  > &ae, Matrix <  $T_-$  > \*A)

Assemble local matrix into global matrix.

template < class T\_, size\_t N\_, class E\_ > void element\_assembly (const E\_ &e, const LocalMatrix < T\_, N\_, N\_ > &ae, SkMatrix < T\_ > &A)

Assemble local matrix into global skyline matrix.

• template < class T\_ , size\_t N\_ , class E\_ > void element\_assembly (const E\_ &e, const LocalMatrix < T\_ , N\_ , N\_ > &ae, SkSMatrix < T\_ > &A)

Assemble local matrix into global symmetric skyline matrix.

• template < class T\_ , size\_t N\_ , class E\_ > void element\_assembly (const E\_ &e, const LocalMatrix < T\_ , N\_ , N\_ > &ae, SpMatrix < T\_ > &A)

Assemble local matrix into global sparse matrix.

• template < class T\_, size\_t N\_> void side\_assembly (const Element &e, const LocalMatrix < T\_, N\_, N\_ > &ae, SpMatrix < T\_ > &A)

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

• template < class T\_, size\_t N\_> void side\_assembly (const Element &e, const LocalMatrix < T\_, N\_, N\_ > &ae, SkSMatrix <  $T_{-} > &A$ Side assembly of local matrix into global matrix (as instance of class SkSMatrix). • template < class T\_, size\_t N\_> void side\_assembly (const Element &e, const LocalMatrix < T\_, N\_, N\_ > &ae, SkMatrix <  $T_{-} > \&A$ Side assembly of local matrix into global matrix (as instance of class SkMatrix). • template < class T\_, size\_t N\_> void side\_assembly (const Element &e, const LocalVect< T\_, N\_> &be, Vect< T\_> &b) Side assembly of local vector into global vector. • template<class T\_>  $Vect < T_- > operator* (const BMatrix < T_- > &A, const Vect < T_- > &b)$ Operator \* (Multiply vector by matrix and return resulting vector. • template<class T\_> BMatrix $< T_- > operator* (T_- a, const BMatrix<math>< T_- > &A)$ Operator \* (Premultiplication of matrix by constant) • template<class T\_> ostream & operator << (ostream &s, const BMatrix < T<sub>-</sub> > &a) Output matrix in output stream. • template<class T\_>  $Vect < T_- > operator* (const DMatrix < T_- > &A, const Vect < T_- > &b)$ Operator \* (Multiply vector by matrix and return resulting vector. • template<class T\_> ostream & operator << (ostream &s, const DMatrix < T<sub>-</sub> > &a) Output matrix in output stream. • template<class T\_>  $Vect < T_- > operator* (const DSMatrix < T_- > &A, const Vect < T_- > &b)$ Operator \* (Multiply vector by matrix and return resulting vector. template<class T<sub>−</sub>> ostream & operator << (ostream &s, const DSMatrix < T\_> &a) Output matrix in output stream. • template < class T\_, size\_t NR\_, size\_t NC\_> LocalMatrix < T., NR., NC. > operator\* (T. a, const LocalMatrix < T., NR., NC. > &x) Operator \* (Multiply matrix x by scalar a) • template < class T\_, size\_t NR\_, size\_t NC\_> LocalVect < T\_, NR\_ > operator\* (const LocalMatrix < T\_, NR\_, NC\_ > &A, const LocalVect <  $T_{-}, NC_{-} > \&x$ Operator \* (Multiply matrix A by vector x)• template < class T\_, size\_t NR\_, size\_t NC\_> LocalMatrix < T., NR., NC. > operator / (T. a, const LocalMatrix < T., NR., NC. > &x) Operator / (Divide matrix x by scalar a) • template<class T\_, size\_t NR\_, size\_t NC\_> LocalMatrix < T\_, NR\_, NC\_ > operator+ (const LocalMatrix < T\_, NR\_, NC\_ > &x, const LocalMatrix $< T_-, NR_-, NC_- > \&y$ ) Operator + (Add matrix x to y)• template < class T\_, size\_t NR\_, size\_t NC\_> LocalMatrix < T\_, NR\_, NC\_ > operator- (const LocalMatrix < T\_, NR\_, NC\_ > &x, const

LocalMatrix $< T_-, NR_-, NC_- > &y)$ 

```
Operator - (Subtract matrix y from x)
• template < class T_, size_t NR_, size_t NC_>
  ostream & operator << (ostream &s, const LocalMatrix < T_, NR_, NC_ > &A)
      Output vector in output stream.

    template < class T_, size_t N_>

  LocalVect< T_-, N_- > operator + (const LocalVect < T_-, N_- > &x, const LocalVect < T_-, N_- >
  &y)
      Operator + (Add two vectors)
• template < class T_, size_t N_>
  LocalVect< T<sub>-</sub>, N<sub>-</sub> > operator- (const LocalVect< T<sub>-</sub>, N<sub>-</sub> > &x, const LocalVect< T<sub>-</sub>, N<sub>-</sub> >
  &y)
      Operator - (Subtract two vectors)
• template < class T_, size_t N_>
  LocalVect< T_-, N_- > operator* (T_- a, const LocalVect<math>< T_-, N_- > \&x)
      Operator * (Premultiplication of vector by constant)
• template<class T_, size_t N_>
  LocalVect< T<sub>-</sub>, N<sub>-</sub>> operator/ (T<sub>-</sub> a, const LocalVect< T<sub>-</sub>, N<sub>-</sub>> &x)
      Operator / (Division of vector by constant)
• template < class T_, size_t N_>
  real_t Dot (const LocalVect< T_-, N_- > \&a, const LocalVect< T_-, N_- > \&b)
      Calculate dot product of 2 vectors (instances of class Local Vect)
• template<class T_, size_t N_>
  void Scale (T_- a, const LocalVect< T_-, N_- > &x, LocalVect< T_-, N_- > &y)
      Multiply vector x by constant a and store result in y.
• template < class T_, size_t N_>
  void Scale (T_- a, LocalVect< T_-, N_- > \&x)
      Multiply vector x by constant a and store result in x.
• template<class T_, size_t N_>
  void Axpy (T_a, const LocalVect< T_a, N_a > &x, LocalVect< T_a, N_a > &y)
      Add a*x to vector y.
• template<class T_, size_t N_>
  void Copy (const LocalVect< T<sub>-</sub>, N<sub>-</sub>> &x, LocalVect< T<sub>-</sub>, N<sub>-</sub>> &y)
      Copy vector x into vector y.
• template<class T_, size_t N_>
  ostream & operator << (ostream &s, const Local Vect < T_-, N_ > &v)
      Output vector in output stream.

    template<class T₋>

  bool operator== (const Point< T_- > &a, const Point< T_- > &b)
      Operator ==
• template<class T_>
  Point< T_- > operator + (const Point < T_- > &a, const Point < T_- > &b)
      Operator +
• template<class T_>
  Point< T_- > operator+ (const Point< T_- > &a, const T_- &x)
      Operator +
• template<class T_>
  Point< T_-> operator- (const Point< T_-> &a)
      Unary Operator -
```

```
    template<class T<sub>−</sub>>

  Point< T_- > operator- (const Point< T_- > \&a, const Point< T_- > \&b)
      Operator -
• template<class T_>
  Point< T_- > operator- (const Point< T_- > &a, const T_- &x)
      Operator -

    template<class T₋>

  Point< T_- > operator* (const T_- &a, const Point<math>< T_- > \&b)
      Operator *
• template<class T_>
  Point< T_- > operator* (const int &a, const Point< T_- > \&b)
      Operator *.
• template<class T_>
  Point< T_- > operator* (const Point<math>< T_- > \&b, const T_- \&a)
      Operator /
• template<class T_>
  Point< T_- > operator* (const Point<math>< T_- > \&b, const int &a)
      Operator *

    template<class T<sub>-</sub>>

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

    template<class T<sub>−</sub>>

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

    Point < double > CrossProduct (const Point < double > &lp, const Point < double > &rp)

      Return Cross product of two vectors lp and rp

    bool areClose (const Point < double > &a, const Point < double > &b, double toler=OFE←

  LI_TOLERANCE)
      Return true if both instances of class Point<double> are distant with less then toler

    double SqrDistance (const Point < double > &a, const Point < double > &b)

      Return squared euclidean distance between points a and b

    double Distance (const Point < double > &a, const Point < double > &b)

      Return euclidean distance between points a and b

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

      Comparison operator. Returns true if all components of first vector are lower than those of second one.

    template<class T₋>

  std::ostream & operator << (std::ostream &s, const Point < T_> &a)
      Output point coordinates.

    template<class T₋>

  bool operator== (const Point2D< T<sub>-</sub>> &a, const Point2D< T<sub>-</sub>> &b)
      Operator ==.
• template<class T₋>
  Point2D< T_-> operator+ (const Point2D< T_-> &a, const Point2D< T_-> &b)
      Operator +.

    template<class T_>

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

    template<class T<sub>-</sub>>

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

```
Unary Operator -

    template<class T<sub>−</sub>>

  Point2D< T_-> operator- (const Point2D< T_-> &a, const Point2D< T_-> &b)
      Operator -
• template<class T_>
  Point2D< T_-> operator- (const Point2D< T_-> &a, const T_- &x)
      Operator -
  template<class T_>
  Point2D< T_-> operator* (const T_- &a, const Point2D< T_-> &b)
      Operator *.

    template<class T_>

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

    template<class T₋>

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

    template<class T_>

  T_- operator* (const Point2D< T_- > &b, const Point2D< T_- > &a)
• template<class T_>
  Point2D< T_-> operator/ (const Point2D< T_-> &b, const T_- &a)
• bool areClose (const Point2D< real_t > &a, const Point2D< real_t > &b, real_t toler=OFE←
  LI_TOLERANCE)
      Return true if both instances of class Point2D<real_t> are distant with less then toler [Default: OFEL←
      I\_EPSMCH].

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

      Return squared euclidean distance between points a and b
• real_t Distance (const Point2D< real_t > &a, const Point2D< real_t > &b)
      Return euclidean distance between points a and b
• template<class T_>
  std::ostream & operator<< (std::ostream &s, const Point2D< T_> &a)
      Output point coordinates.
• template<class T_>
  Vect < T_- > operator* (const SkMatrix < T_- > &A, const Vect < T_- > &b)
      Operator * (Multiply vector by matrix and return resulting vector.
• template<class T_>
  ostream & operator << (ostream &s, const SkMatrix < T_ > &a)
      Output matrix in output stream.
• template<class T_>
  Vect < T_- > operator* (const SkSMatrix < T_- > &A, const Vect < T_- > &b)
      Operator * (Multiply vector by matrix and return resulting vector.

    template<class T<sub>−</sub>>

  ostream & operator << (ostream &s, const SkSMatrix < T<sub>-</sub> > &a)
      Output matrix in output stream.
• template<class T₋>
  Vect < T_- > operator* (const SpMatrix < T_- > &A, const Vect < T_- > &b)
```

```
Operator * (Multiply vector by matrix and return resulting vector.
• template<class T_>
  ostream & operator << (ostream &s, const SpMatrix < T<sub>-</sub> > &A)
      Output matrix in output stream.
• template<class T_>
  Vect < T_- > operator* (const TrMatrix < T_- > &A, const Vect < T_- > &b)
      Operator * (Multiply vector by matrix and return resulting vector.
• template<class T_>
  TrMatrix < T_- > operator* (T_- a, const TrMatrix < T_- > &A)
      Operator * (Premultiplication of matrix by constant)
• template<class T_>
  ostream & operator << (ostream &s, const TrMatrix < T_> &A)
      Output matrix in output stream.
• template<class T_>
  Vect< T_-> operator+ (const Vect< T_-> &x, const Vect< T_-> &y)
      Operator + (Addition of two instances of class Vect)

    template<class T₋>

  Vect< T_-> operator- (const Vect< T_-> &x, const Vect< T_-> &y)
      Operator - (Difference between two vectors of class Vect)
• template<class T_>
  Vect < T_- > operator* (const T_- &a, const Vect < T_- > &x)
      Operator * (Premultiplication of vector by constant)
• template<class T_>
  Vect < T_- > operator* (const Vect < T_- > &x, const T_- &a)
      Operator * (Postmultiplication of vector by constant)
• template<class T_>
  Vect < T_- > operator/ (const Vect < T_- > &x, const T_- &a)
      Operator / (Divide vector entries by constant)
• template<class T_>
  T_- Dot (const Vect< T_- > &x, const Vect< T_- > &y)
      Calculate dot product of two vectors.

    real_t Discrepancy (Vect< real_t > &x, const Vect< real_t > &y, int n, int type=1)

      Return discrepancy between 2 vectors x and y

    real_t Discrepancy (Vect< complex_t > &x, const Vect< complex_t > &y, int n, int type=1)

      Return discrepancy between 2 vectors x and y
• void Modulus (const Vect< complex_t > &x, Vect< real_t > &y)
      Calculate modulus of complex vector.
• void Real (const Vect< complex_t > &x, Vect< real_t > &y)
      Calculate real part of complex vector.
• void Imag (const Vect< complex_t > &x, Vect< real_t > &y)
      Calculate imaginary part of complex vector.
• template<class T_>
  istream & operator>> (istream &s, Vect < T_- > &v)
• template<class T_>
  ostream & operator << (ostream &s, const Vect < T<sub>-</sub> > &v)
      Output vector in output stream.

    ostream & operator<< (ostream &s, const Edge &ed)</li>

      Output edge data.
```

ostream & operator<< (ostream &s, const Element &el)</li>
 Output element data.

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

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

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

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

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

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

• void getMesh (string file, ExternalFileFormat form, Mesh &mesh, size\_t nb\_dof=1)

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

void getBamg (string file, Mesh &mesh, size\_t nb\_dof=1)

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

• void getEasymesh (string file, Mesh &mesh, size\_t nb\_dof=1)

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

• void getGambit (string file, Mesh &mesh, size\_t nb\_dof=1)

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

void getGmsh (string file, Mesh &mesh, size\_t nb\_dof=1)

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

void getMatlab (string file, Mesh &mesh, size\_t nb\_dof=1)

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

void getNetgen (string file, Mesh &mesh, size\_t nb\_dof=1)

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

• void getTetgen (string file, Mesh &mesh, size\_t nb\_dof=1)

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

• void getTriangle (string file, Mesh &mesh, size\_t nb\_dof=1)

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

• ostream & operator<< (ostream &s, const Grid &g)

Output grid data.

• ostream & operator << (ostream &s, const Material &m)

Output material data.

ostream & operator<< (ostream &s, const Mesh &ms)</li>

Output mesh data.

• ostream & operator << (ostream &s, const MeshAdapt &a)

Output MeshAdapt class data.

• ostream & operator<< (ostream &s, const NodeList &nl)

Output NodeList instance.

• ostream & operator<< (ostream &s, const ElementList &el)

Output ElementList instance.

• ostream & operator<< (ostream &s, const SideList &sl)

Output SideList instance.

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

Output EdgeList instance.

• size\_t Label (const Node &nd)

Return label of a given node.

• size\_t Label (const Element &el)

Return label of a given element.

• size\_t Label (const Side &sd)

Return label of a given side.

• size\_t Label (const Edge &ed)

Return label of a given edge.

• size\_t NodeLabel (const Element &el, size\_t n)

Return global label of node local label in element.

• size\_t NodeLabel (const Side &sd, size\_t n)

Return global label of node local label in side.

• Point< real\_t > Coord (const Node &nd)

Return coordinates of a given node.

• int Code (const Node &nd, size\_t i=1)

Return code of a given (degree of freedom of) node.

• int Code (const Element &el)

Return code of a given element.

• int Code (const Side &sd, size\_t i=1)

Return code of a given (degree of freedom of) side.

• bool operator== (const Element &el1, const Element &el2)

Check equality between 2 elements.

bool operator== (const Side &sd1, const Side &sd2)

Check equality between 2 sides.

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

Calculate deformed mesh using a displacement field.

void MeshToMesh (Mesh &m1, Mesh &m2, const Vect< real\_t > &u1, Vect< real\_t > &u2, size\_t nx, size\_t ny=0, size\_t dof=1)

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

void MeshToMesh (const Vect< real\_t > &u1, Vect< real\_t > &u2, size\_t nx, size\_t ny=0, size\_t nz=0, size\_t dof=1)

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

void MeshToMesh (Mesh &m1, Mesh &m2, const Vect< real\_t > &u1, Vect< real\_t > &u2, const Point< real\_t > &xmin, const Point< real\_t > &xmax, size\_t nx, size\_t ny, size\_t nz, size\_t dof=1)

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

• real\_t getMaxSize (const Mesh &m)

Return maximal size of element edges for given mesh.

real\_t getMinSize (const Mesh &m)

Return minimal size of element edges for given mesh.

real\_t getMinElementMeasure (const Mesh &m)

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

real\_t getMaxElementMeasure (const Mesh &m)

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

real\_t getMinSideMeasure (const Mesh &m)

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

real\_t getMaxSideMeasure (const Mesh &m)

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

• real\_t getMeanElementMeasure (const Mesh &m)

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

real\_t getMeanSideMeasure (const Mesh &m)

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

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

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

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

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

• void setSideCodes (Mesh &m, const string &exp, int code, size\_t dof=1)

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

• void setBoundarySideCodes (Mesh &m, const string &exp, int code, size\_t dof=1)

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

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

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

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

Say if a given node belongs to a given element.

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

Say if a given node belongs to a given side.

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

Say if a given side belongs to a given element.

ostream & operator<< (ostream &s, const Node &nd)</li>

Output node data.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

ostream & operator<< (ostream &s, const Side &sd)</li>

Output side data.

• ostream & operator << (ostream &s, const Estimator &r)

 $Output\ estimator\ vector\ in\ output\ stream.$ 

• template<class  $T_->$ 

int BiCG (const SpMatrix<  $T_-$  > &A, const Prec<  $T_-$  > &P, const Vect<  $T_-$  > &b, Vect<  $T_-$  > &x, int max\_it, real\_t &toler)

Biconjugate gradient solver function.

• template < class T\_>

int BiCG (const SpMatrix<  $T_->$  &A, int prec, const Vect<  $T_->$  &b, Vect<  $T_->$  &x, int max\_it, real\_t toler)

Biconjugate gradient solver function.

• template<class T\_>

int BiCGStab (const SpMatrix  $< T_- > &A$ , const Prec  $< T_- > &P$ , const Vect  $< T_- > &b$ , Vect  $< T_- > &x$ , int max\_it, real\_t toler)

Biconjugate gradient stabilized solver function.

• template<class T\_>

int BiCGStab (const SpMatrix< T $_-$  > &A, int prec, const Vect< T $_-$  > &b, Vect< T $_-$  > &x, int max\_it, real\_t toler)

Biconjugate gradient stabilized solver function.

void BSpline (size\_t n, size\_t t, Vect< Point< real\_t >> &control, Vect< Point< real\_t >> &output, size\_t num\_output)

Function to perform a B-spline interpolation.

• template<class T\_>

int CG (const SpMatrix< T $_-$  > &A, const Prec< T $_-$  > &P, const Vect< T $_-$  > &b, Vect< T $_-$  > &x, int max $_i$ t, real $_i$ t toler)

Conjugate gradient solver function.

• template<class T\_>

int CG (const SpMatrix  $< T_- > &A$ , int prec, const Vect  $< T_- > &b$ , Vect  $< T_- > &x$ , int max\_it, real\_t toler)

Conjugate gradient solver function.

• template<class T\_>

int CGS (const SpMatrix<  $T_->$  &A, const Prec<  $T_->$  &P, const Vect<  $T_->$  &b, Vect<  $T_->$  &x, int max\_it, real\_t toler)

Conjugate Gradient Squared solver function.

• template<class T\_>

int CGS (const SpMatrix $< T_- > &A$ , int prec, const Vect $< T_- > &b$ , Vect $< T_- > &x$ , int max\_it, real\_t toler)

Conjugate Gradient Squared solver function.

• ostream & operator<< (ostream &s, const EigenProblemSolver &es)

Output eigenproblem information.

• template<class T\_>

int GMRes (const SpMatrix< T $_-$  > &A, const Prec< T $_-$  > &P, const Vect< T $_-$  > &b, Vect< T $_-$  > &x, size $_+$ t m, int max $_+$ it, real $_+$ t toler)

GMRes solver function.

• template<class T\_>

int GMRes (const SpMatrix< T $_-$  > &A, int prec, const Vect< T $_-$  > &b, Vect< T $_-$  > &x, size $_-$ t m, int max $_-$ it, real $_-$ t toler)

GMRes solver function.

• template<class T\_>

int GS (const SpMatrix<  $T_->$  &A, const Vect<  $T_->$  &b, Vect<  $T_->$  &x, real\_t omega, int max\_it, real\_t toler)

Gauss-Seidel solver function.

• template<class T\_>

int Jacobi (const SpMatrix  $< T_- > &A$ , const Vect  $< T_- > &b$ , Vect  $< T_- > &x$ , real\_t omega, int max\_it, real\_t toler)

Jacobi solver function.

ostream & operator<< (ostream &s, const NLASSolver &nl)</li>

Output nonlinear system information.

• ostream & operator<< (ostream &s, const ODESolver &de)

Output differential system information.

ostream & operator<< (ostream &s, const OptSolver &os)</li>

Output differential system information.

```
• template<class T_, class M_>
  int Richardson (const M_ &A, const Vect < T_ > &b, Vect < T_ > &x, real_t omega, int max_it,
  real_t toler, int verbose)
      Richardson solver function.
• template<class T_>
  void Schur (SkMatrix < T_- > &A, SpMatrix < T_- > &U, SpMatrix < T_- > &L, SpMatrix < T_- > &L
  > &D, Vect < T_- > &b, Vect < T_- > &c)
      Solve a linear system of equations with a 2x2-block matrix.
• template < class T_, class M_>
  int SSOR (const M_- &A, const Vect < T_- > \&b, Vect < T_- > \&x, int max_it, real_t toler)
      SSOR solver function.
• ostream & operator << (ostream &s, TimeStepping &ts)
      Output differential system information.
• void banner (const string &prog="")
      Outputs a banner as header of any developed program.
• template<class T_>
  void QuickSort (std::vector < T_- > &a, int begin, int end)
      Function to sort a vector.
• template<class T_>
  void qksort (std::vector < T_- > &a, int begin, int end)
      Function to sort a vector.
• template < class T_, class C_>
  void qksort (std::vector< T_-> &a, int begin, int end, C_- compare)
      Function to sort a vector according to a key function.
• int Sgn (real_t a)
      Return sign of a: - 1 or 1.

    real_t Abs2 (complex_t a)

      Return square of modulus of complex number a
• real_t Abs2 (real_t a)
      Return square of real number a

    real_t Abs (real_t a)

      Return absolute value of a

    real_t Abs (complex_t a)

      Return modulus of complex number a

    real_t Abs (const Point < real_t > &p)

      Return Norm of vector a
real_t Conjg (real_t a)
      Return complex conjugate of real number a

    complex_t Conjg (complex_t a)

      Return complex conjugate of complex number a
• real_t Max (real_t a, real_t b, real_t c)
      Return maximum value of real numbers a, b and c
• int Max (int a, int b, int c)
      Return maximum value of integer numbers a, b and c
• real_t Min (real_t a, real_t b, real_t c)
```

Return minimum value of real numbers a, b and c

• int Min (int a, int b, int c)

```
Return minimum value of integer numbers a, b and c
• real_t Max (real_t a, real_t b, real_t c, real_t d)
      Return maximum value of integer numbers a, b, c and d
• int Max (int a, int b, int c, int d)
      Return maximum value of integer numbers a, b, c and d
• real_t Min (real_t a, real_t b, real_t c, real_t d)
      Return minimum value of real numbers a, b, c and d
• int Min (int a, int b, int c, int d)
      Return minimum value of integer numbers a, b, c and d
• real_t Arg (complex_t x)
      Return argument of complex number x

    complex_t Log (complex_t x)

      Return principal determination of logarithm of complex number x

    template<class T_>

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

    template<class T_>

  void Xpy (const vector < T_- > &x, vector < T_- > &y)
      Add vector x to y
• template<class T_>
  void Axpy (size_t n, T_- a, T_-*x, T_-*y)
      Multiply array x by a and add result to y
• template<class T_>
  void Axpy (T_- a, const vector < T_- > &x, vector < T_- > &y)
      Multiply vector x by a and add result to y
• template<class T_>
  void Axpy (T_a, const Vect< T_a > &x, Vect< T_a > &y)
      Multiply vector x by a and add result to y
• template<class T_>
  void Copy (size_t n, T_- *x, T_- *y)
      Copy array x to y n is the arrays size.
• real_t Error2 (const vector < real_t > &x, const vector < real_t > &y)
      Return absolute L2 error between vectors x and y

    real_t RError2 (const vector < real_t > &x, const vector < real_t > &y)

      Return absolute L^2 error between vectors x and y

    real_t ErrorMax (const vector < real_t > &x, const vector < real_t > &y)
```

Return absolute Max. error between vectors x and y real\_t RErrorMax (const vector < real\_t > &x, const vector < real\_t > &y) Return relative Max. error between vectors x and y template<class T₋>  $T_-$  Dot (size\_t n,  $T_- *x$ ,  $T_- *y$ ) Return dot product of arrays x and y real\_t Dot (const vector < real\_t > &x, const vector < real\_t > &y) Return dot product of vectors x and y. real\_t operator\* (const vector< real\_t > &x, const vector< real\_t > &y) *Operator* \* (*Dot product of 2 vector instances*) • template<class T\_>  $T_-$  Dot (const Point  $< T_- > &x$ , const Point  $< T_- > &y$ ) Return dot product of x and y• real\_t exprep (real\_t x) Compute the exponential function with avoiding over and underflows. template<class T\_> void Assign (vector  $< T_- > &v$ , const  $T_- &a$ ) Assign the value a to all entries of a vector v• template<class T\_> void clear (vector  $< T_- > &v$ ) Assign 0 to all entries of a vector. • template<class T\_> void clear (Vect<  $T_->$  &v) Assign 0 to all entries of a vector. • real\_t Nrm2 (size\_t n, real\_t \*x) Return 2-norm of array x • real\_t Nrm2 (const vector < real\_t > &x) Return 2-norm of vector x • template<class T\_> real\_t Nrm2 (const Point $< T_- > &a$ ) Return 2-norm of a bool Equal (real\_t x, real\_t y, real\_t toler=OFELI\_EPSMCH) Function to return true if numbers x and y are close up to a given tolerance toler • char itoc (int i) Function to convert an integer to a character. • std::string itos (int i) Function to convert an integer to a string. • std::string itos (size\_t i)

Function to convert an integer to a string.

• std::string dtos (real\_t d)

Function to convert a real to a string.

• template<class T\_>

T<sub>-</sub> stringTo (const std::string &s)

Function to convert a string to a template type parameter.

### **Variables**

• Node \* theNode

A pointer to Node.

• Element \* the Element

A pointer to Element.

• Side \* theSide

A pointer to Side.

• Edge \* theEdge

A pointer to Edge.

int Verbosity

Verbosity parameter.

• int theStep

Time step counter.

• int theIteration

Iteration counter.

• int NbTimeSteps

Number of time steps.

• int MaxNbIterations

Maximal number of iterations.

• real\_t theTimeStep

Time step label.

• real\_t theTime

Time value.

• real\_t theFinalTime

Final time value.

• real\_t theTolerance

Tolerance value for convergence.

real\_t theDiscrepancy

Value of discrepancy for an iterative procedure Its default value is 1.0.

bool Converged

Boolean variable to say if an iterative procedure has converged.

• bool InitPetsc

### 6.1.1 Detailed Description

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

Namespace OFELI groups all OFELI library classes, functions and global variables.

### **6.1.2** Enumeration Type Documentation

### enum NormType

Choose type of vector norm to compute

Enumerator

NORM1 1-norm

WNORM1 Weighted 1-norm (Discrete L1-Norm)

NORM2 2-norm

WNORM2 Weighted 2-norm (Discrete L2-Norm)

NORM\_MAX Max-norm (Infinity norm)

# Chapter 7

# **Class Documentation**

# 7.1 AbsEqua < T\_ > Class Template Reference

Mother abstract class to describe equation.

Inheritance diagram for AbsEqua < T\_>:



### **Public Member Functions**

• AbsEqua ()

Default constructor.

• virtual ~AbsEqua ()

Destructor.

• void setMesh (Mesh &m)

Define mesh and renumber DOFs after removing imposed ones.

• Mesh & getMesh () const

 $Return\ reference\ to\ {\it Mesh}\ instance.$ 

• LinearSolver < T\_> & getLinearSolver ()

Return reference to linear solver instance.

• Matrix < T\_> \* getMatrix () const

Return pointer to matrix.

• void setSolver (Iteration ls, Preconditioner pc=IDENT\_PREC)

Choose solver for the linear system.

• void setLinearSolver (Iteration ls, Preconditioner pc=IDENT\_PREC)

Choose solver for the linear system.

• void setMatrixType (int t)

Choose type of matrix.

• int solveLinearSystem (Matrix < T\_ > \*A, Vect < T\_ > &b, Vect < T\_ > &x)

Solve the linear system with given matrix and right-hand side.

• int solveLinearSystem (Vect< T\_> &b, Vect< T\_> &x)

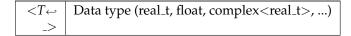
Solve the linear system with given right-hand side.

### 7.1.1 Detailed Description

 $template < class \ T_{-}> \\ class \ OFELI::AbsEqua < T_{-}>$ 

Mother abstract class to describe equation.

**Template Parameters** 



Author

Rachid Touzani

Copyright

GNU Lesser Public License

### 7.1.2 Member Function Documentation

Mesh& getMesh ( ) const

Return reference to Mesh instance.

Returns

Reference to Mesh instance

### void setSolver ( Iteration ls, Preconditioner $pc = IDENT\_PREC$ )

Choose solver for the linear system.

in	ls	Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER	
		<ul> <li>DIRECT_SOLVER, Use a facorization solver [default]</li> </ul>	
		CG_SOLVER, Conjugate Gradient iterative solver	
		CGS_SOLVER, Squared Conjugate Gradient iterative solver	
		BICG_SOLVER, BiConjugate Gradient iterative solver	
		BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver	
		GMRES_SOLVER, GMRES iterative solver	
		QMR_SOLVER, QMR iterative solver	

### 7.1. ABSEQUA< T $_->$ CLASS TEMPLATE REFERENCE APTER 7. CLASS DOCUMENTATION

### Parameters

in	рс	Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:
		IDENT_PREC, Identity preconditioner (no preconditioning [default])
		DIAG_PREC, Diagonal preconditioner
		ILU_PREC, Incomplete LU factorization preconditioner

## void setLinearSolver ( Iteration ls, Preconditioner $pc = IDENT\_PREC$ )

Choose solver for the linear system.

### Parameters

in	ls	Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER	
		DIRECT_SOLVER, Use a facorization solver [default]	
		CG_SOLVER, Conjugate Gradient iterative solver	
		CGS_SOLVER, Squared Conjugate Gradient iterative solver	
		BICG_SOLVER, BiConjugate Gradient iterative solver	
		BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver	
		GMRES_SOLVER, GMRES iterative solver	
		QMR_SOLVER, QMR iterative solver	
in	рс	Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:	
		IDENT_PREC, Identity preconditioner (no preconditioning [default])	
		DIAG_PREC, Diagonal preconditioner	
		ILU_PREC, Incomplete LU factorization preconditioner	

## void setMatrixType ( int t )

Choose type of matrix.

in	t	Type of the used matrix. To choose among the enumerated values: SKYLINE, SPARSE,
		DIAGONAL TRIDIAGONAL, SYMMETRIC, UNSYMMETRIC, IDENTITY

### int solveLinearSystem ( Matrix< T $_->*A$ , Vect< T $_->$ & b, Vect< T $_->$ & x )

Solve the linear system with given matrix and right-hand side.

#### **Parameters**

in	A	Pointer to matrix of the system	
in	b	Vector containing right-hand side	
in,out	x	Vector containing initial guess of solution on input, actual solution on outpu	

### int solveLinearSystem ( Vect< $T_-$ > & b, Vect< $T_-$ > & x)

Solve the linear system with given right-hand side.

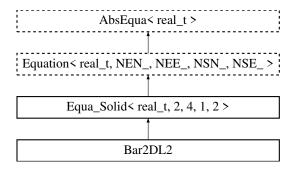
#### **Parameters**

in	b	Vector containing right-hand side	
in,out	x	Vector containing initial guess of solution on input, actual solution on output	

### 7.2 Bar2DL2 Class Reference

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

Inheritance diagram for Bar2DL2:



### **Public Member Functions**

• Bar2DL2 ()

Default Constructor.

• Bar2DL2 (Mesh &ms)

Constructor using a Mesh instance.

• Bar2DL2 (Mesh &ms, Vect< real\_t > &u)

Constructor using a Mesh instance and a solution vector instance.

• ~Bar2DL2 ()

Destructor.

void setSection (real\_t A)

Define bar section.

• void LMass (real\_t coef=1)

Add lumped mass matrix to element matrix after multiplying it by coefficient coef

• void Mass (real\_t coef=1)

Add consistent mass matrix to element matrix after multiplying it by coefficient coef

• void Stiffness (real\_t coef=1.)

Add element stiffness to left hand side.

• real\_t Stress () const

Return stresses in bar.

void getStresses (Vect< real\_t > &s)

Return stresses in the truss structure (elementwise)

• void build ()

Build the linear system of equations.

### **Additional Inherited Members**

### 7.2.1 Detailed Description

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

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

### 7.2.2 Constructor & Destructor Documentation

#### Bar2DL2()

Default Constructor.

Constructs an empty equation.

#### Bar2DL2 (Mesh & ms)

Constructor using a Mesh instance.

#### **Parameters**

in	ms	Reference Mesh instance

### Bar2DL2 ( Mesh & ms, Vect< real\_t > & u )

Constructor using a Mesh instance and a solution vector instance.

in	ms	Reference Mesh instance
in,out	и	Reference to solution vector

### 7.2.3 Member Function Documentation

void LMass ( real\_t coef = 1 ) [virtual]

Add lumped mass matrix to element matrix after multiplying it by coefficient coef

**Parameters** 

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

Reimplemented from Equa\_Solid < real\_t, 2, 4, 1, 2 >.

void Mass ( real\_t coef = 1 ) [virtual]

Add consistent mass matrix to element matrix after multiplying it by coefficient coef

**Parameters** 

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

Reimplemented from Equa\_Solid < real\_t, 2, 4, 1, 2 >.

void Stiffness ( real\_t coef = 1. ) [virtual]

Add element stiffness to left hand side.

**Parameters** 

in | coef | Coefficient to multuply by added term [Default: 1].

Reimplemented from Equa\_Solid < real\_t, 2, 4, 1, 2 >.

void getStresses ( Vect< real\_t > & s )

Return stresses in the truss structure (elementwise)

**Parameters** 

in s Vect instance containing axial stresses in elements

#### void build ( )

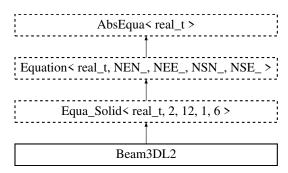
Build the linear system of equations.

Before using this function, one must have properly selected appropriate options for:

- The choice of a steady state or transient analysis
- In the case of transient analysis, the choice of a time integration scheme and a lumped or consistent mass matrix
- The choice of desired linear system solver

### 7.3 Beam3DL2 Class Reference

To build element equations for 3-D beam equations using 2-node lines. Inheritance diagram for Beam3DL2:



### **Public Member Functions**

• Beam3DL2()

Default Constructor.

• Beam3DL2 (Mesh &ms, real\_t A, real\_t I1, real\_t I2)

Constructor using mesh and constant beam properties.

• Beam3DL2 (Mesh &ms)

Constructor using a Mesh instance.

• Beam3DL2 (Mesh &ms, Vect< real\_t > &u)

Constructor using a Mesh instance and solution vector.

• ~Beam3DL2 ()

Destructor.

• void set (real\_t A, real\_t I1, real\_t I2)

Set constant beam properties.

• void set (const Vect< real\_t > &A, const Vect< real\_t > &I1, const Vect< real\_t > &I2)

Set nonconstant beam properties.

void getDisp (Vect< real\_t > &d)

Get vector of displacements at nodes.

• void LMass (real\_t coef=1.)

Add element lumped Mass contribution to element matrix after multiplication by coef

• void Mass (real\_t coef=1.)

Add element consistent Mass contribution to RHS after multiplication by coef (not implemented)

• void Stiffness (real\_t coef=1.)

Add element stiffness to element matrix.

• void Load (const Vect< real\_t > &f)

Add contributions for loads.

• void setBending ()

Set bending contribution to stiffness.

• void setAxial ()

Set axial contribution to stiffness.

• void setShear ()

Set shear contribution to stiffness.

• void setTorsion ()

Set torsion contribution to stiffness.

• void setNoBending ()

Set no bending contribution.

void setNoAxial ()

Set no axial contribution.

• void setNoShear ()

Set no shear contribution.

void setNoTorsion ()

Set no torsion contribution.

void setReducedIntegration ()

Set reduced integration.

• void AxialForce (Vect< real\_t > &f)

Return axial force in element.

• void ShearForce (Vect< real\_t > &sh)

Return shear force in element.

• void BendingMoment (Vect< real\_t > &m)

Return bending moment in element.

• void TwistingMoment (Vect< real\_t > &m)

Return twisting moments.

• void build ()

Build the linear system of equations.

void buildEigen (SkSMatrix< real\_t > &K, Vect< real\_t > &M)

Build global stiffness and mass matrices for the eigen system.

### **Additional Inherited Members**

### 7.3.1 Detailed Description

To build element equations for 3-D beam equations using 2-node lines.

This class enables building finite element arrays for 3-D beam elements using 6 degrees of freedom per node and 2-Node line elements.

### 7.3.2 Constructor & Destructor Documentation

### Beam3DL2 ( Mesh & ms, real\_t A, real\_t I1, real\_t I2 )

Constructor using mesh and constant beam properties.

in	ms	Mesh instance
in	A	Section area of the beam
in	I1	first (x) momentum of inertia
in	<i>I</i> 2	second (y) momentum of inertia

### Beam3DL2 ( Mesh & ms )

Constructor using a Mesh instance.

### Parameters

in	ms	Reference to Mesh instance
----	----	----------------------------

### Beam3DL2 ( Mesh & ms, Vect< real\_t > & u )

Constructor using a Mesh instance and solution vector.

#### **Parameters**

in	ms	Reference to Mesh instance
in,out	и	Solution vector

### 7.3.3 Member Function Documentation

void set ( real\_t A, real\_t I1, real\_t I2 )

Set constant beam properties.

### Parameters

in	Α	Section area of the beam
in	I1	first (x) momentum of inertia
in	<i>I</i> 2	second (y) momentum of inertia

void set ( const Vect< real\_t > & A, const Vect< real\_t > & I1, const Vect< real\_t > & I2 )

Set nonconstant beam properties.

#### **Parameters**

i	n	A	Vector containing section areas of the beam (for each element)
i	n	I1	Vector containing first (x) momentum of inertia (for each element)
i	n	I2	Vector containing second (y) momentum of inertia (for each element)

### void getDisp ( Vect< real\_t > & d )

Get vector of displacements at nodes.

out	d	Vector containing three components for each node that are x, y and z
		displacements.

### void AxialForce ( Vect< real\_t > & f )

Return axial force in element.

#### **Parameters**

### void ShearForce ( Vect< real\_t > & sh )

Return shear force in element.

#### Parameters

out	sh	Vector containing shear forces (2 components) in each element. This vector is
		resized in the function

### void BendingMoment ( Vect< real\_t > & m )

Return bending moment in element.

### Parameters

out	m	Vector containing bending moments (2 components) in each element. This vector
		is resized in the function

### void TwistingMoment ( Vect< real\_t > & m )

Return twisting moments.

### Parameters

out	m	Vector containing twisting moment in each element. This vector is resized in the
		function

### void buildEigen ( SkSMatrix< real\_t > & K, Vect< real\_t > & M)

Build global stiffness and mass matrices for the eigen system. Case where the mass matrix is lumped

in	K	Stiffness matrix
in	M	Vector containing diagonal mass matrix

### 7.4 BiotSavart Class Reference

Class to compute the magnetic induction from the current density using the Biot-Savart formula.

### **Public Member Functions**

• BiotSavart ()

Default constructor.

• BiotSavart (Mesh &ms)

Constructor using mesh data.

• BiotSavart (Mesh &ms, const Vect< real\_t > &J, Vect< real\_t > &B, int code=0)

Constructor using mesh and vector of real current density.

BiotSavart (Mesh &ms, const Vect < complex\_t > &J, Vect < complex\_t > &B, int code=0)

Constructor using mesh and vector of complex current density.

• ∼BiotSavart ()

Destructor.

• void setCurrentDensity (const Vect< real\_t > &J)

Set (real) current density given at elements.

void setCurrentDensity (const Vect< complex\_t > &J)

Set (real) current density given at elements.

void setMagneticInduction (Vect< real\_t > &B)

Transmit (real) magnetic induction vector given at nodes.

void setMagneticInduction (Vect< complex\_t > &B)

Transmit (complex) magnetic induction vector given at nodes.

• void selectCode (int code)

Choose code of faces or edges at which current density is given.

• void setPermeability (real\_t mu)

Set the magnetic permeability coefficient.

void setBoundary ()

Choose to compute the magnetic induction at boundary nodes only.

• Point < real\_t > getB3 (Point < real\_t > x)

Compute the real magnetic induction at a given point using the volume Biot-Savart formula.

• Point< real\_t > getB2 (Point< real\_t > x)

Compute the real magnetic induction at a given point using the surface Biot-Savart formula.

• Point< real\_t > getB1 (Point< real\_t > x)

Compute the real magnetic induction at a given point using the line Biot-Savart formula.

• Point< complex\_t > getBC3 (Point< real\_t > x)

Compute the complex magnetic induction at a given point using the volume Biot-Savart formula.

• Point< complex\_t > getBC2 (Point< real\_t > x)

Compute the complex magnetic induction at a given point using the surface Biot-Savart formula.

• Point< complex\_t > getBC1 (Point< real\_t > x)

Compute the complex magnetic induction at a given point using the line Biot-Savart formula.

• int run ()

Run the calculation by the Biot-Savart formula.

### 7.4.1 Detailed Description

Class to compute the magnetic induction from the current density using the Biot-Savart formula. Given a current density vector given at elements, a collection of sides of edges (piecewise constant), this class enables computing the magnetic induction vector (continuous and piecewise linear) using the Ampere equation. This magnetic induction is obtained by using the Biot-Savart formula which can be either a volume, surface or line formula depending on the nature of the current density vector.

#### Author

Rachid Touzani

### Copyright

**GNU Lesser Public License** 

### 7.4.2 Constructor & Destructor Documentation

#### BiotSavart ( Mesh & ms )

Constructor using mesh data.

#### Parameters

in	ms	Mesh instance
----	----	---------------

#### BiotSavart (Mesh & ms, const Vect < real\_t > & J, Vect < real\_t > & B, int code = 0)

Constructor using mesh and vector of real current density.

The current density is assumed piecewise constant

#### **Parameters**

in	ms	Mesh instance
in	J	Sidewise vector of current density (J is a real valued vector), in the case of a surface supported current
in	В	Nodewise vector that contains, once the member function run is used, the magnetic induction
in	code	Only sides with given <i>code</i> support current [Default: 0]

### BiotSavart ( Mesh & ms, const Vect < complex t > & J, Vect < complex t > & B, int code = 0)

Constructor using mesh and vector of complex current density.

The current density is assumed piecewise constant

in	ms	Mesh instance
in	J	Sidewise vector of current density (J is a complex valued vector), in the case of a surface supported current

#### **Parameters**

iı	В	Nodewise vector that contains, once the member function run is used, the magnetic induction
iı	code	Only sides with given code support current [Default: 0]

### 7.4.3 Member Function Documentation

### void setCurrentDensity ( const Vect< real\_t > & J )

Set (real) current density given at elements.

The current density is assumed piecewise constant and real valued. This function can be used in the case of the volume Biot-Savart formula.

#### **Parameters**

iı	ı J	-	Current density vector (Vect instance) and real entries	
----	-----	---	---	--

### void setCurrentDensity ( const Vect< complex\_t > & J )

Set (real) current density given at elements.

The current density is assumed piecewise constant and complex valued. This function can be used in the case of the volume Biot-Savart formula.

### Parameters

in	J	Current density vector (Vect instance) of complex entries	
----	---	---	--

### void setMagneticInduction ( Vect< real\_t > & B )

Transmit (real) magnetic induction vector given at nodes.

#### **Parameters**

out	В	Magnetic induction vector (Vect instance) and real entries
-----	---	--

### void setMagneticInduction ( Vect< complex\_t > & B )

Transmit (complex) magnetic induction vector given at nodes.

#### Parameters

out	В	Magnetic induction vector (Vect instance) and complex entries

#### void setPermeability ( real\_t mu )

Set the magnetic permeability coefficient.

#### **Parameters**

in	ти	Magnetic permeability
----	----	-----------------------

### void setBoundary ( )

Choose to compute the magnetic induction at boundary nodes only.

By default the magnetic induction is computed (using the function run) at all mesh nodes

#### Note

This function has no effect for surface of line Biot-Savart formula

### Point<real $_t>$ getB3 ( Point< real $_t>$ x )

Compute the real magnetic induction at a given point using the volume Biot-Savart formula.

This function computes a real valued magnetic induction for a real valued current density field

#### **Parameters**

in	x	Coordinates of point at which the magnetic induction is computed
----	---	--

#### Returns

Value of the magnetic induction at x

### Point<real\_t> getB2 ( Point< real\_t> x )

Compute the real magnetic induction at a given point using the surface Biot-Savart formula.

This function computes a real valued magnetic induction for a real valued current density field

#### **Parameters**

in	x	Coordinates of point at which the magnetic induction is computed
----	---	--

### Returns

Value of the magnetic induction at x

### Point<real\_t> getB1 ( Point< real\_t> x )

Compute the real magnetic induction at a given point using the line Biot-Savart formula.

This function computes a real valued magnetic induction for a real valued current density field

in	х	Coordinates of point at which the magnetic induction is computed	
----	---	--	--

#### Returns

Value of the magnetic induction at x

### Point<complex $_t>$ getBC3 ( Point< real $_t>x$ )

Compute the complex magnetic induction at a given point using the volume Biot-Savart formula. This function computes a complex valued magnetic induction for a complex valued current density field

#### **Parameters**

in	x	Coordinates of point at which the magnetic induction is computed
----	---	--

#### Returns

Value of the magnetic induction at x

### Point<complex\_t> getBC2 ( Point< real\_t> x )

Compute the complex magnetic induction at a given point using the surface Biot-Savart formula. This function computes a complex valued magnetic induction for a complex valued current density field

#### **Parameters**

in	х	Coordinates of point at which the magnetic induction is computed
----	---	--

#### Returns

Value of the magnetic induction at x

### Point<complex $_t>$ getBC1 ( Point< real $_t>x$ )

Compute the complex magnetic induction at a given point using the line Biot-Savart formula.

This function computes a complex valued magnetic induction for a complex valued current density field

### Parameters

in	x	Coordinates of point at which the magnetic induction is computed
----	---	--

#### Returns

Value of the magnetic induction at x

### int run ( )

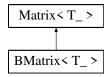
Run the calculation by the Biot-Savart formula.

This function computes the magnetic induction, which is stored in the vector B given in the constructor

# 7.5 BMatrix< T\_> Class Template Reference

To handle band matrices.

Inheritance diagram for BMatrix<  $T_->$ :



### **Public Member Functions**

• BMatrix ()

Default constructor.

• BMatrix (size\_t size, int ld, int ud)

Constructor that for a band matrix with given size and bandwidth.

• BMatrix (const BMatrix &m)

Copy Constructor.

• ∼BMatrix ()

Destructor.

• void setSize (size\_t size, int ld, int ud)

Set size (number of rows) and storage of matrix.

• void MultAdd (const Vect< T\_> &x, Vect< T\_> &y) const

Multiply matrix by vector x and add result to y

• void MultAdd (T\_a, const Vect< T\_> &x, Vect< T\_> &y) const

Multiply matrix by vector  $\mathbf{a} * \mathbf{x}$  and add result to  $\mathbf{y}$ 

• void Mult (const Vect< T $_->$  &x, Vect< T $_->$  &y) const

Multiply matrix by vector x and save result in y

• void TMult (const Vect<  $T_-$  > &x, Vect<  $T_-$  > &y) const

Multiply transpose of matrix by vector x and save result in y

• void Axpy (T<sub>-</sub> a, const BMatrix < T<sub>-</sub> > &x)

Add to matrix the product of a matrix by a scalar.

• void Axpy (T<sub>-</sub> a, const Matrix < T<sub>-</sub> > \*x)

Add to matrix the product of a matrix by a scalar.

void set (size\_t i, size\_t j, const T\_ &val)

Add constant val to an entry (i, j) of the matrix.

void add (size\_t i, size\_t j, const T\_ &val)

Add constant val value to an entry (i, j) of the matrix.

• T\_ operator() (size\_t i, size\_t j) const

Operator () (Constant version).

• T\_ & operator() (size\_t i, size\_t j)

Operator () (Non constant version).

• BMatrix< T<sub>-</sub> > & operator= (const BMatrix< T<sub>-</sub> > &m)

Operator =.

• BMatrix< T\_> & operator= (const T\_ &x)

 $Operator = Assign \ matrix \ to \ identity \ times \ x.$ 

• BMatrix< T<sub>−</sub> > & operator\*= (const T<sub>−</sub> &x)

Operator \*=.

• BMatrix < T\_ > & operator+= (const T\_ &x)

*Operator* +=.

• int setLU ()

Factorize the matrix (LU factorization)

• int solve (Vect< T<sub>-</sub> > &b, bool fact=false)

Solve linear system.

• int solve (const Vect< T\_> &b, Vect< T\_> &x, bool fact=false)

Solve linear system.

• T\_\* get () const

Return C-Array.

• T\_get (size\_t i, size\_t j) const

Return entry (i, j) of matrix.

## 7.5.1 Detailed Description

```
template<class T_> class OFELI::BMatrix< T_>
```

To handle band matrices.

This class enables storing and manipulating band matrices. The matrix can have different numbers of lower and upper co-diagonals

**Template Parameters** 

```
T \leftarrow  Data type (double, float, complex<double>, ...)
```

Author

Rachid Touzani

Copyright

GNU Lesser Public License

### 7.5.2 Constructor & Destructor Documentation

## BMatrix ( )

Default constructor.

Initialize a zero dimension band matrix

### BMatrix ( size\_t size, int ld, int ud )

Constructor that for a band matrix with given size and bandwidth. Assign 0 to all matrix entries.

**Parameters** 

in	size	Number of rows and columns

in	ld	Number of lower co-diagonals (must be $> 0$ )
in	ud	Number of upper co-diagonals (must be $> 0$ )

## 7.5.3 Member Function Documentation

void setSize ( size\_t size, int ld, int ud )

Set size (number of rows) and storage of matrix.

#### Parameters

in	size	Number of rows and columns
in	ld	Number of lower co-diagonals (must be $> 0$ )
in	ud	Number of upper co-diagonals (must be $> 0$ )

## void Axpy ( $T_- a$ , const BMatrix< $T_- > \& x$ )

Add to matrix the product of a matrix by a scalar.

#### Parameters

in a Scalar to premultiply		Scalar to premultiply
in	x	Matrix by which a is multiplied. The result is added to current instance

## void Axpy ( $T_-a$ , const Matrix< $T_- > *x$ ) [virtual]

Add to matrix the product of a matrix by a scalar.

### Parameters

in	а	Scalar to premultiply	
in	x	Matrix by which a is multiplied. The result is added to current instance	

Implements Matrix  $< T_- >$ .

## $T_{-}$ operator() ( size\_t i, size\_t j ) const [virtual]

Operator () (Constant version).

## Parameters

in	i	Row index
in	j	Column index

Implements Matrix  $< T_- >$ .

### T\_& operator() ( size\_t i, size\_t j ) [virtual]

Operator () (Non constant version).

#### **Parameters**

in	i	Row index
in	j	Column index

Implements Matrix  $< T_- >$ .

## BMatrix<T $_->$ & operator= ( const BMatrix< T $_->$ & m )

Operator =.

Copy matrix m to current matrix instance.

### BMatrix<T $_->$ & operator\*= ( const T $_-$ & x )

Operator \*=.

Premultiply matrix entries by constant value x

### BMatrix<T $_->$ & operator+= ( const T $_-$ & x )

Operator +=.

Add constant x to matrix entries.

#### int setLU ( )

Factorize the matrix (LU factorization)

LU factorization of the matrix is realized. Note that since this is an in place factorization, the contents of the matrix are modified.

#### Returns

- 0 if factorization was normally performed,
- n if the n-th pivot is null.

### Remarks

A flag in this class indicates after factorization that this one has been realized, so that, if the member function solve is called after this no further factorization is done.

### int solve ( Vect< T $_->$ & b, bool fact = false ) [virtual]

Solve linear system.

The linear system having the current instance as a matrix is solved by using the LU decomposition. Solution is thus realized after a factorization step and a forward/backward substitution step. The factorization step is realized only if this was not already done.

Note that this function modifies the matrix contents is a factorization is performed. Naturally, if the the matrix has been modified after using this function, the user has to refactorize it using the function setLU. This is because the class has no non-expensive way to detect if the matrix has been modified. The function setLU realizes the factorization step only.

in,out	b	Vect instance that contains right-hand side on input and solution on output	
in	fact	Unused argument	

#### Returns

- 0 if solution was normally performed,
- n if the n-th pivot is null.

Implements Matrix $< T_- >$ .

int solve ( const Vect< T $_->$  & b, Vect< T $_->$  & x, bool fact = false ) [virtual]

Solve linear system.

The linear system having the current instance as a matrix is solved by using the LU decomposition. Solution is thus realized after a factorization step and a forward/backward substitution step. The factorization step is realized only if this was not already done.

Note that this function modifies the matrix contents is a factorization is performed. Naturally, if the the matrix has been modified after using this function, the user has to refactorize it using the function setLU. This is because the class has no non-expensive way to detect if the matrix has been modified. The function setLU realizes the factorization step only.

#### **Parameters**

in	b	Vect instance that contains right-hand side.
out	x	Vect instance that contains solution
in	fact	Unused argument

## Returns

- 0 if solution was normally performed,
- n if the n-th pivot is null.

Implements Matrix $< T_- >$ .

## 7.6 Brick Class Reference

To store and treat a brick (parallelepiped) figure. Inheritance diagram for Brick:



## **Public Member Functions**

• Brick ()

Default constructor.

• Brick (const Point< real\_t > &bbm, const Point< real\_t > &bbM, int code=1)

Constructor.

- void setBoundingBox (const Point< real\_t > &bbm, const Point< real\_t > &bbM)

  Assign bounding box of the brick.
- Point< real\_t > getBoundingBox1 () const

Return first point of bounding box (xmin,ymin,zmin)

• Point< real\_t > getBoundingBox2 () const

Return second point of bounding box (xmax,ymax,zmax)

• real\_t getSignedDistance (const Point< real\_t > &p) const

Return signed distance of a given point from the current brick.

• Brick & operator+= (Point< real\_t > a)

Operator +=.

• Brick & operator+= (real\_t a)

Operator \*=.

## **Additional Inherited Members**

## 7.6.1 Detailed Description

To store and treat a brick (parallelepiped) figure.

### 7.6.2 Constructor & Destructor Documentation

Brick ( const Point< real\_t > & bbm, const Point< real\_t > & bbM, int code = 1 )

Constructor.

### **Parameters**

in	bbm	first point (xmin,ymin,zmin)
in	bbM	second point (xmax,ymax,zmax)
in	code	Code to assign to rectangle

### 7.6.3 Member Function Documentation

void setBoundingBox ( const Point < real\_t > & bbm, const Point < real\_t > & bbM)

Assign bounding box of the brick.

#### **Parameters**

in	bbm	first point (xmin,ymin,zmin)
in	bbM	second point (xmax,ymax,zmax)

## real\_t getSignedDistance ( const Point< real\_t > & p ) const [virtual]

Return signed distance of a given point from the current brick.

The computed distance is negative if p lies in the brick, negative if it is outside, and 0 on its boundary

#### **Parameters**

in	р	Point <double> instance</double>
----	---	----------------------------------

Reimplemented from Figure.

## Brick& operator+= ( Point< real\_t > a )

Operator +=.

Translate brick by a vector a

### Brick& operator+= $( real_t a )$

Operator \*=.

Scale brick by a factor a

## 7.7 Circle Class Reference

To store and treat a circular figure. Inheritance diagram for Circle:



### **Public Member Functions**

• Circle ()

Default construcor.

Circle (const Point < real\_t > &c, real\_t r, int code=1)

Constructor.

• void setRadius (real\_t r)

Assign radius of circle.

• real\_t getRadius () const

Return radius of circle.

• void setCenter (const Point < real\_t > &c)

Assign coordinates of center of circle.

• Point< real\_t > getCenter () const

Return coordinates of center of circle.

• real\_t getSignedDistance (const Point< real\_t > &p) const

Return signed distance of a given point from the current circle.

• Circle & operator+= (Point< real\_t > a)

Operator +=.

• Circle & operator+= (real\_t a)

Operator \*=.

## **Additional Inherited Members**

## 7.7.1 Detailed Description

To store and treat a circular figure.

### 7.7.2 Constructor & Destructor Documentation

Circle (const Point < real\_t > &  $c_r$  real\_t  $r_r$  int code = 1)

Constructor.

#### **Parameters**

in	С	Coordinates of center of circle
in	r	Radius
in	code	Code to assign to the generated domain [Default: 1]

### 7.7.3 Member Function Documentation

real\_t getSignedDistance ( const Point < real\_t > & p ) const [virtual]

Return signed distance of a given point from the current circle.

The computed distance is negative if p lies in the disk, positive if it is outside, and 0 on the circle

### Parameters

in	р	Point <double> instance</double>
----	---	----------------------------------

Reimplemented from Figure.

Circle& operator+= ( Point< real\_t > a )

Operator +=.

Translate circle by a vector a

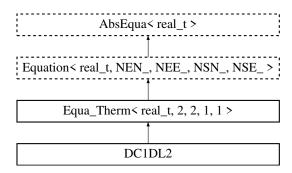
Circle& operator+=  $( real_t a )$ 

Operator \*=.

Scale circle by a factor a

## 7.8 DC1DL2 Class Reference

Builds finite element arrays for thermal diffusion and convection in 1-D using 2-Node elements. Inheritance diagram for DC1DL2:



### **Public Member Functions**

• DC1DL2()

Default Constructor.

- DC1DL2 (Mesh &ms)
- DC1DL2 (Mesh &ms, Vect< real\_t > &u)
- ~DC1DL2 ()

Destructor.

• void LCapacity (real\_t coef=1)

Add lumped capacity matrix to element matrix after multiplying it by coefficient coef

• void Capacity (real\_t coef=1)

Add Consistent capacity matrix to element matrix after multiplying it by coefficient coef.

• void Diffusion (real\_t coef=1)

Add diffusion matrix to element matrix after multiplying it by coefficient coef

• void Convection (const real\_t &v, real\_t coef=1)

Add convection matrix to element matrix after multiplying it by coefficient coef

void Convection (const Vect< real\_t > &v, real\_t coef=1)

Add convection matrix to element matrix after multiplying it by coefficient coef

• void Convection (real\_t coef=1)

Add convection matrix to element matrix after multiplying it by coefficient coef

• void BodyRHS (const Vect< real\_t > &f)

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

• real\_t Flux () const

Return (constant) heat flux in element.

void setInput (EqDataType opt, Vect< real\_t > &u)

Set equation input data.

### **Additional Inherited Members**

### 7.8.1 Detailed Description

Builds finite element arrays for thermal diffusion and convection in 1-D using 2-Node elements. Note that members calculating element arrays have as an argument a real coef that will be multiplied by the contribution of the current element. This makes possible testing different algorithms.

## 7.8.2 Constructor & Destructor Documentation

## DC1DL2()

Default Constructor.

Constructs an empty equation.

#### DC1DL2 (Mesh & ms)

Constructor using mesh instance

### Parameters

in ms	Mesh instance
-------	---------------

### DC1DL2 ( Mesh & ms, Vect< real\_t > & u )

Constructor using mesh instance and solution vector

#### **Parameters**

in	ms	Mesh instance
in,out	и	Vect instance containing solution vector

### 7.8.3 Member Function Documentation

void LCapacity ( real\_t coef = 1 ) [virtual]

Add lumped capacity matrix to element matrix after multiplying it by coefficient coef

## Parameters

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

Reimplemented from Equa\_Therm< real\_t, 2, 2, 1, 1 >.

## void Capacity ( real\_t coef = 1 ) [virtual]

Add Consistent capacity matrix to element matrix after multiplying it by coefficient coef.

### Parameters

i:	n <i>coef</i>	Coefficient to multiply by added term [default: 1]
----	---------------	--

Reimplemented from Equa\_Therm< real\_t, 2, 2, 1, 1 >.

## void Diffusion ( real\_t coef = 1 ) [virtual]

Add diffusion matrix to element matrix after multiplying it by coefficient coef

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

Reimplemented from Equa\_Therm< real\_t, 2, 2, 1, 1 >.

## void Convection ( const real\_t & v, real\_t coef = 1 )

Add convection matrix to element matrix after multiplying it by coefficient coef

#### **Parameters**

in	v	Constant velocity vector
in	coef	Coefficient to multiply by added term [default: 1]

## void Convection ( const Vect< real\_t > & v, real\_t coef = 1 )

Add convection matrix to element matrix after multiplying it by coefficient coef Case where velocity field is given by a vector v

#### **Parameters**

in	v	Velocity vector
in	coef	Coefficient to multiply by added term [default: 1]

## void Convection ( real\_t coef = 1 ) [virtual]

Add convection matrix to element matrix after multiplying it by coefficient coef Case where velocity field has been previouly defined

#### **Parameters**

|--|

Reimplemented from Equa\_Therm< real\_t, 2, 2, 1, 1 >.

### void BodyRHS ( const Vect < real t > & f ) [virtual]

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

#### **Parameters**

in $f$	Vector containing source at nodes.
--------	------------------------------------

Reimplemented from Equa\_Therm< real\_t, 2, 2, 1, 1 >.

### void setInput ( EqDataType opt, Vect < real\_t > & u )

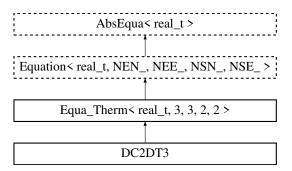
Set equation input data.

in	opt	Parameter that selects data type for input. This parameter is to be chosen in the enumerated variable EqDataType
		INITIAL_FIELD: Initial temperature
		BOUNDARY_CONDITION_DATA: Boundary condition (Dirichlet)
		SOURCE_DATA: Heat source
		FLUX_DATA: Heat flux (Neumann boundary condition)
		VELOCITY: Velocity vector (for the convection term)
in	и	Vector containing input data

## 7.9 DC2DT3 Class Reference

Builds finite element arrays for thermal diffusion and convection in 2-D domains using 3-Node triangles.

Inheritance diagram for DC2DT3:



## **Public Member Functions**

• DC2DT3 ()

Default Constructor. Constructs an empty equation.

• DC2DT3 (Mesh &ms)

Constructor using Mesh data.

• DC2DT3 (Mesh &ms, Vect< real\_t > &u)

Constructor using Mesh and initial condition.

• ~DC2DT3 ()

Destructor.

void LCapacity (real\_t coef=1)

Add lumped capacity matrix to element matrix after multiplying it by coefficient coef

• void Capacity (real\_t coef=1)

Add Consistent capacity matrix to element matrix after multiplying it by coefficient coef

• void Diffusion (real\_t coef=1)

Add diffusion matrix to element matrix after multiplying it by coefficient coef

void Diffusion (const LocalMatrix < real\_t, 2, 2 > &diff, real\_t coef=1)

Add diffusion matrix to element matrix after multiplying it by coefficient coef

void Convection (const Point < real\_t > &v, real\_t coef=1)

Add convection matrix to element matrix after multiplying it by coefficient coef

void Convection (const Vect< real\_t > &v, real\_t coef=1)

Add convection matrix to element matrix after multiplying it by coefficient coef

void Convection (real\_t coef=1)

Add convection matrix to element matrix after multiplying it by coefficient coef

• void LinearExchange (real\_t coef, real\_t T)

Add an edge linear exchange term to left and right-hand sides.

• void BodyRHS (const Vect< real\_t > &f)

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

void BodyRHS (real\_t f)

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

void BoundaryRHS (real\_t flux)

Add boundary right-hand side flux to right hand side.

void BoundaryRHS (const Vect< real\_t > &f)

Add boundary right-hand side term to right hand side after multiplying it by coefficient coef

• void Periodic (real\_t coef=1.e20)

Add contribution of periodic boundary condition (by a penalty technique).

• Point< real\_t > & Flux () const

Return (constant) heat flux in element.

• void Grad (Vect< Point< real\_t >> &g)

Compute gradient of solution.

• Point< real\_t > & Grad (const Vect< real\_t > &u) const

Return gradient of a vector in element.

• void setInput (EqDataType opt, Vect< real\_t > &u)

Set equation input data.

void JouleHeating (const Vect< real\_t > &sigma, const Vect< real\_t > &psi)

Set Joule heating term as source.

### **Additional Inherited Members**

## 7.9.1 Detailed Description

Builds finite element arrays for thermal diffusion and convection in 2-D domains using 3-Node triangles.

Note that members calculating element arrays have as an argument a real coef that will be multiplied by the contribution of the current element. This makes possible testing different algorithms.

### 7.9.2 Constructor & Destructor Documentation

## DC2DT3 ( Mesh & ms )

Constructor using Mesh data.

### Parameters

in	ms	Mesh instance

#### DC2DT3 ( Mesh & ms, Vect< real\_t > & u )

Constructor using Mesh and initial condition.

#### **Parameters**

in	ms	Mesh instance
in	и	Vect instance containing initial solution

## 7.9.3 Member Function Documentation

## void LCapacity ( real\_t coef = 1 ) [virtual]

Add lumped capacity matrix to element matrix after multiplying it by coefficient coef

### Parameters

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

Reimplemented from Equa\_Therm< real\_t, 3, 3, 2, 2 >.

### void Capacity ( real\_t coef = 1 ) [virtual]

Add Consistent capacity matrix to element matrix after multiplying it by coefficient coef

### Parameters

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

Reimplemented from Equa\_Therm< real\_t, 3, 3, 2, 2 >.

## void Diffusion ( real\_t coef = 1 ) [virtual]

Add diffusion matrix to element matrix after multiplying it by coefficient coef

## Parameters

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

Reimplemented from Equa\_Therm< real\_t, 3, 3, 2, 2 >.

## void Diffusion ( const LocalMatrix< real\_t, 2, 2 > & diff, real\_t coef = 1)

Add diffusion matrix to element matrix after multiplying it by coefficient coef Case where the diffusivity matrix is given as an argument.

#### Parameters

in	diff	Diffusion matrix (class LocalMatrix).	
in	coef	Coefficient to multiply by added term [Default: 1]	

### void Convection ( const Point < real\_t > & v, real\_t coef = 1 )

Add convection matrix to element matrix after multiplying it by coefficient coef

#### **Parameters**

in	v	Constant velocity vector	
in	coef	Coefficient to multiply by added term [Default: 1]	

### void Convection ( const Vect< real\_t > & v, real\_t coef = 1 )

Add convection matrix to element matrix after multiplying it by coefficient coef Case where velocity field is given by a vector v

#### **Parameters**

in	v	Velocity vector	
in	coef	Coefficient to multiply by added term (Default: 1]	

## void Convection ( real\_t coef = 1 ) [virtual]

Add convection matrix to element matrix after multiplying it by coefficient coef Case where velocity field has been previouly defined

#### **Parameters**

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

Reimplemented from Equa\_Therm< real\_t, 3, 3, 2, 2 >.

## void LinearExchange ( real\_t coef, real\_t T )

Add an edge linear exchange term to left and right-hand sides.

### Parameters

in	coef	Coefficient of exchange
in	T	External value for exchange

### Remarks

This assumes a constant value of T

### void BodyRHS ( const Vect< real\_t > & f ) [virtual]

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

	in $f$	Vector containing source at nodes.	
--	--------	------------------------------------	--

Reimplemented from Equa\_Therm< real\_t, 3, 3, 2, 2 >.

## void BodyRHS ( $real_t f$ )

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

Case where the body right-hand side is piecewise constant.

#### **Parameters**

in	f	Value of thermal source (Constant in element).
----	---	--

## void BoundaryRHS ( real\_t flux )

Add boundary right-hand side flux to right hand side.

#### **Parameters**

	in	flux	Vector containing source at side nodes.
--	----	------	---

### void BoundaryRHS ( const Vect < real\_t > & f ) [virtual]

Add boundary right-hand side term to right hand side after multiplying it by coefficient coef

### Parameters

in	f	Vector containing source at nodes
	)	0

Reimplemented from Equa\_Therm< real\_t, 3, 3, 2, 2 >.

## void Periodic ( real\_t coef = 1.e20 )

Add contribution of periodic boundary condition (by a penalty technique).

Boundary nodes where periodic boundary conditions are to be imposed must have codes equal to PERIODIC\_A on one side and PERIODIC\_B on the opposite side.

#### **Parameters**

	in	coef	Value of penalty parameter [Default: 1.e20]	
--	----	------	---	--

### void Grad ( Vect< Point< real\_t >> & g )

Compute gradient of solution.

ſ
---

## Point<real\_t>& Grad ( const Vect< real\_t>& u ) const

Return gradient of a vector in element.

#### **Parameters**

in	и	Global vector for which gradient is computed. Vector u has as size the total number
		of nodes

## void setInput ( EqDataType opt, Vect< real\_t > & u )

Set equation input data.

### Parameters

in	opt	Parameter to select type of input (enumerated values)
		INITIAL_FIELD: Initial temperature
		BOUNDARY_CONDITION_DATA: Boundary condition (Dirichlet)
		SOURCE_DATA: Heat source
		FLUX_DATA: Heat flux (Neumann boundary condition)
		VELOCITY_FIELD: Velocity vector (for the convection term)
in	и	Vector containing input data

## void JouleHeating ( const Vect< real\_t > & sigma, const Vect< real\_t > & psi)

Set Joule heating term as source.

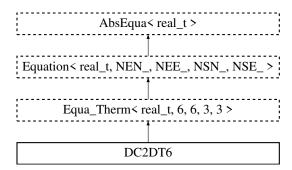
#### Parameters

in	sigma	Vect instance containing electric conductivity (elementwise)
in	psi	Vect instance containing electric potential (elementwise)

## 7.10 DC2DT6 Class Reference

Builds finite element arrays for thermal diffusion and convection in 2-D domains using 6-Node triangles.

Inheritance diagram for DC2DT6:



### **Public Member Functions**

• DC2DT6 ()

Default Constructor.

• DC2DT6 (Mesh &ms)

Constructor using Mesh data.

• DC2DT6 (Mesh &ms, Vect < real\_t > &u)

Constructor using Mesh data and solution vector.

• ~DC2DT6 ()

Destructor.

• void LCapacity (real\_t coef=1)

Add lumped capacity matrix to element matrix after multiplying it by coefficient coef.

• void Capacity (real\_t coef=1)

Add Consistent capacity matrix to element matrix after multiplying it by coefficient coef.

• void Diffusion (real\_t coef=1)

Add diffusion matrix to element matrix after multiplying it by coefficient coef

• void Convection (real\_t coef=1)

Add convection matrix to left-hand side after multiplying it by coefficient coef

• void Convection (Point< real\_t > &v, real\_t coef=1)

Add convection matrix to left hand side after multiplying it by coefficient coef

• void Convection (const Vect< real\_t > &v, real\_t coef=1)

Add convection matrix to left-hand side after multiplying it by coefficient coef

• void BodyRHS (const Vect< real\_t > &f)

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

void BoundaryRHS (const Vect< real\_t > &f)

Add boundary right-hand side term to right hand side after multiplying it by coefficient coef

### **Additional Inherited Members**

### 7.10.1 Detailed Description

Builds finite element arrays for thermal diffusion and convection in 2-D domains using 6-Node triangles.

Note that members calculating element arrays have as an argument a real coef that will be multiplied by the contribution of the current element. This makes possible testing different algorithms.

## 7.10.2 Constructor & Destructor Documentation

### DC2DT6()

Default Constructor.

Constructs an empty equation.

#### DC2DT6 ( Mesh & ms )

Constructor using Mesh data.

### Parameters

in ms	Mesh instance
-------	---------------

### DC2DT6 ( Mesh & ms, Vect< real\_t > & u )

Constructor using Mesh data and solution vector.

#### **Parameters**

j	in	ms	Mesh instance
j	in,out	и	Vect instance containing solution vector

### 7.10.3 Member Function Documentation

## void LCapacity ( real\_t coef = 1 ) [virtual]

Add lumped capacity matrix to element matrix after multiplying it by coefficient coef.

#### Parameters

1		2226	Coasti siant to mustingly buy add ad town (dataset walks 1)
	111	coej	Coefficient to multiply by added term (default value = 1).

Reimplemented from Equa\_Therm< real\_t, 6, 6, 3, 3 >.

## void Capacity ( real\_t coef = 1 ) [virtual]

Add Consistent capacity matrix to element matrix after multiplying it by coefficient coef.

### Parameters

in	coef	Coefficient to multiply by added term (default value = 1).

Reimplemented from Equa\_Therm< real\_t, 6, 6, 3, 3 >.

## void Diffusion ( real\_t coef = 1 ) [virtual]

Add diffusion matrix to element matrix after multiplying it by coefficient coef

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

Reimplemented from Equa\_Therm< real\_t, 6, 6, 3, 3 >.

## void Convection ( real\_t coef = 1 ) [virtual]

Add convection matrix to left-hand side after multiplying it by coefficient coef Case where velocity field has been previouly defined

#### **Parameters**

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

Reimplemented from Equa\_Therm< real\_t, 6, 6, 3, 3 >.

### void Convection ( Point< real\_t > & v, real\_t coef = 1 )

Add convection matrix to left hand side after multiplying it by coefficient coef

#### **Parameters**

in	v	Constant velocity vector.
in	coef	Coefficient to multiply by added term [Default: 1].

### void Convection ( const Vect< real\_t > & v, real\_t coef = 1 )

Add convection matrix to left-hand side after multiplying it by coefficient coef Case where velocity field is given by a vector v

### **Parameters**

in	v	Velocity vector.
in	coef	Coefficient to multiply by added term [Default: 1].

## ${f void\ BodyRHS}$ ( ${f const\ Vect}{<{\it real\_t}} > \& f$ ) [virtual]

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

#### **Parameters**

in	f	Local vector (of size 6) containing source at nodes
----	---	---

Reimplemented from Equa\_Therm< real\_t, 6, 6, 3, 3 >.

### void BoundaryRHS ( const Vect< real\_t > & f ) [virtual]

Add boundary right-hand side term to right hand side after multiplying it by coefficient coef

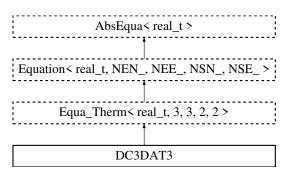
i	n f	Vector containing source at nodes
---	-----	-----------------------------------

Reimplemented from Equa\_Therm< real\_t, 6, 6, 3, 3 >.

## 7.11 DC3DAT3 Class Reference

Builds finite element arrays for thermal diffusion and convection in 3-D domains with axisymmetry using 3-Node triangles.

Inheritance diagram for DC3DAT3:



### **Public Member Functions**

• DC3DAT3 ()

Default Constructor.

• DC3DAT3 (Mesh &ms)

Constructor using Mesh data.

DC3DAT3 (Mesh &ms, Vect < real\_t > &u)

Constructor using Mesh data and solution vector.

• ~DC3DAT3 ()

Destructor.

• void LCapacity (real\_t coef=1)

Add lumped capacity matrix to element matrix after multiplying it by coefficient coef.

• void Capacity (real\_t coef=1)

Add Consistent capacity matrix to element matrix after multiplying it by coefficient coef

void Diffusion (real\_t coef=1)

Add diffusion matrix to left-hand side after multiplying it by coefficient coef

• void Diffusion (const LocalMatrix < real\_t, 2, 2 > &diff, real\_t coef=1)

Add diffusion matrix to left-hand side after multiplying it by coefficient coef

void BodyRHS (const Vect< real\_t > &f)

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

void BoundaryRHS (real\_t flux)

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

void BoundaryRHS (const Vect< real\_t > &f)

Add boundary right-hand side term to right hand side after multiplying it by coefficient coef

• Point< real\_t > & Grad (const Vect< real\_t > &u)

Return gradient of a vector in element.

## **Additional Inherited Members**

### 7.11.1 Detailed Description

Builds finite element arrays for thermal diffusion and convection in 3-D domains with axisymmetry using 3-Node triangles.

Note that members calculating element arrays have as an argument a real coef that will be multiplied by the contribution of the current element. This makes possible testing different algorithms.

### 7.11.2 Constructor & Destructor Documentation

### DC3DAT3()

Default Constructor.

Constructs an empty equation.

#### DC3DAT3 (Mesh & ms)

Constructor using Mesh data.

#### **Parameters**

in	ms	Mesh instance
----	----	---------------

## DC3DAT3 ( Mesh & ms, Vect< real\_t > & u )

Constructor using Mesh data and solution vector.

#### **Parameters**

in	ms	Mesh instance
in,out	и	Vect instance containing solution vector

## 7.11.3 Member Function Documentation

## void LCapacity ( real\_t coef = 1 ) [virtual]

Add lumped capacity matrix to element matrix after multiplying it by coefficient coef.

#### **Parameters**

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

Reimplemented from Equa\_Therm< real\_t, 3, 3, 2, 2 >.

### void Capacity ( real\_t coef = 1 ) [virtual]

Add Consistent capacity matrix to element matrix after multiplying it by coefficientcoef

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

Reimplemented from Equa\_Therm< real\_t, 3, 3, 2, 2 >.

## void Diffusion ( real\_t coef = 1 ) [virtual]

Add diffusion matrix to left-hand side after multiplying it by coefficient coef

#### **Parameters**

Reimplemented from Equa\_Therm< real\_t, 3, 3, 2, 2 >.

## void Diffusion ( const LocalMatrix< real\_t, 2, 2 > & diff, real\_t coef = 1)

Add diffusion matrix to left-hand side after multiplying it by coefficient coef Case where the diffusivity matrix is given as an argument

#### **Parameters**

in	diff	Instance of class DMatrix containing diffusivity matrix
in	coef	Coefficient to multiply by added term [Default: 1]

### void BodyRHS ( const Vect < real.t > & f ) [virtual]

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

#### Parameters

in	f	Local vector (of size 3) containing source at odes.
----	---	---

Reimplemented from Equa\_Therm< real\_t, 3, 3, 2, 2 >.

### void BoundaryRHS ( real\_t flux )

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

#### Parameters

	in	flux	Value of flux to impose on the side
--	----	------	-------------------------------------

## void BoundaryRHS ( const Vect< real\_t> & f ) [virtual]

Add boundary right-hand side term to right hand side after multiplying it by coefficient coef

	in f	Vector containing source at nodes	1
--	------	-----------------------------------	---

Reimplemented from Equa\_Therm< real\_t, 3, 3, 2, 2 >.

## Point<real $_{-}$ t>& Grad ( const Vect< real $_{-}$ t>& u )

Return gradient of a vector in element.

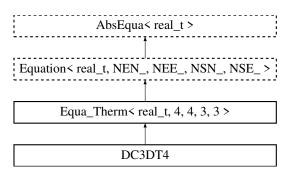
#### **Parameters**

in	и	Vector for which gradient is computed.
----	---	--

## 7.12 DC3DT4 Class Reference

Builds finite element arrays for thermal diffusion and convection in 3-D domains using 4-Node tetrahedra.

Inheritance diagram for DC3DT4:



### **Public Member Functions**

- DC3DT4()
  - Default Constructor.
- DC3DT4 (Mesh &ms)

Constructor using Mesh data.

• DC3DT4 (Mesh &ms, Vect< real\_t > &u)

Constructor using Mesh and initial condition.

- ~DC3DT4 ()
  - Destructor.
- void LCapacity (real\_t coef=1)

Add lumped capacity matrix to element matrix after multiplying it by coefficient coef

• void Capacity (real\_t coef=1)

Add consistent capacity matrix to element matrix after multiplying it by coefficient coef

• void Diffusion (real\_t coef=1)

Add diffusion matrix to element matrix after multiplying it by coefficient coef.

• void Diffusion (const DMatrix < real\_t > &diff, real\_t coef=1)

Add diffusion matrix to element matrix after multiplying it by coefficient coef

• void Convection (real\_t coef=1)

Add convection matrix to element matrix after multiplying it by coefficient coef

• void Convection (const Point< real\_t > &v, real\_t coef=1)

Add convection matrix to element matrix after multiplying it by coefficient coef

void Convection (const Vect< Point< real\_t >> &v, real\_t coef=1)

Add convection matrix to element matrix after multiplying it by coefficient coef

• void BodyRHS (const Vect< real\_t > &f)

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

void BoundaryRHS (const Vect< real\_t > &f)

Add boundary right-hand side term to right hand side after multiplying it by coefficient coef

void BoundaryRHS (real\_t flux)

Add boundary right-hand side flux to right hand side.

• Point< real\_t > Flux () const

Return (constant) heat flux in element.

void Grad (Vect< Point< real\_t >> &g)

Compute gradient of solution.

• void Periodic (real\_t coef=1.e20)

Add contribution of periodic boundary condition (by a penalty technique).

## **Additional Inherited Members**

## 7.12.1 Detailed Description

Builds finite element arrays for thermal diffusion and convection in 3-D domains using 4-Node tetrahedra.

Note that members calculating element arrays have as an argument a real coef that will be multiplied by the contribution of the current element. This makes possible testing different algorithms.

### 7.12.2 Constructor & Destructor Documentation

#### DC3DT4()

Default Constructor.

Constructs an empty equation.

#### DC3DT4 (Mesh & ms)

Constructor using Mesh data.

#### **Parameters**

in	ms	Mesh instance

### DC3DT4 ( Mesh & ms, Vect< real\_t > & u )

Constructor using Mesh and initial condition.

in	ms	Mesh instance
in	и	Vect instance containing initial solution

### 7.12.3 Member Function Documentation

### void LCapacity ( real\_t coef = 1 ) [virtual]

Add lumped capacity matrix to element matrix after multiplying it by coefficient coef

### Parameters

Reimplemented from Equa\_Therm< real\_t, 4, 4, 3, 3 >.

### void Capacity ( real\_t coef = 1 ) [virtual]

Add consistent capacity matrix to element matrix after multiplying it by coefficient coef

#### **Parameters**

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

Reimplemented from Equa\_Therm< real\_t, 4, 4, 3, 3 >.

## void Diffusion ( real\_t coef = 1 ) [virtual]

Add diffusion matrix to element matrix after multiplying it by coefficient coef.

### **Parameters**

in	coef	Coefficient to multiply by added term (default value = 1).
----	------	--

Reimplemented from Equa\_Therm< real\_t, 4, 4, 3, 3 >.

### void Diffusion ( const DMatrix< real\_t > & diff, real\_t coef = 1 )

Add diffusion matrix to element matrix after multiplying it by coefficient coef Case where the diffusivity matrix is given as an argument.

#### **Parameters**

in	diff	Diffusion matrix (class DMatrix).
in	coef	Coefficient to multiply by added term [Default: 1].

### void Convection ( real\_t coef = 1 ) [virtual]

Add convection matrix to element matrix after multiplying it by coefficient coef Case where velocity field has been previouly defined

### **Parameters**

|--|

Reimplemented from Equa\_Therm< real\_t, 4, 4, 3, 3 >.

### void Convection ( const Point < real t > & v, real t coef = 1 )

Add convection matrix to element matrix after multiplying it by coefficient coef

#### **Parameters**

i	n	v	Constant velocity vector
i	n	coef	Coefficient to multiply by added term [Default: 1].

### void Convection ( const Vect< Point< real\_t >> & v, real\_t coef = 1)

Add convection matrix to element matrix after multiplying it by coefficient coef Case where velocity field is given by a vector v.

#### **Parameters**

in	v	Velocity vector.
in	coef	Coefficient to multiply by added term [Default: 1].

## ${f void\ BodyRHS}$ ( ${f const\ Vect}<{f real\_t}>\&f$ ) [virtual]

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

### Parameters

in f Vector containing source	e at nodes.
-------------------------------	-------------

Reimplemented from Equa\_Therm< real\_t, 4, 4, 3, 3 >.

### void BoundaryRHS ( const Vect < real t > & f ) [virtual]

Add boundary right-hand side term to right hand side after multiplying it by coefficient coef Case where body source is given by a vector

#### **Parameters**

in	f	Vector containing source at nodes.
----	---	------------------------------------

Reimplemented from Equa\_Therm< real\_t, 4, 4, 3, 3 >.

## void BoundaryRHS ( real\_t flux )

Add boundary right-hand side flux to right hand side.

#### Parameters

	in	flux	Vector containing source at side nodes.	
--	----	------	---	--

## void Grad ( Vect< Point< real\_t >> & g )

Compute gradient of solution.

#### **Parameters**

in	8	Elementwise vector containing gradient of solution.
----	---	---

## void Periodic ( real\_t coef = 1.e20 )

Add contribution of periodic boundary condition (by a penalty technique).

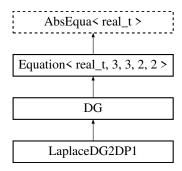
Boundary nodes where periodic boundary conditions are to be imposed must have codes equal to PERIODIC\_A on one side and PERIODIC\_B on the opposite side.

### Parameters

in	coef	Value of penalty parameter [Default: 1.e20].
----	------	--

## 7.13 DG Class Reference

Enables preliminary operations and utilities for the Discontinous Galerkin method. Inheritance diagram for DG:



## **Public Member Functions**

• DG (Mesh &ms, size\_t degree=1)

Constructor with mesh and degree of the method.

• ~DG ()

Destructor.

• int setGraph ()

Set matrix graph.

## 7.13.1 Detailed Description

Enables preliminary operations and utilities for the Discontinous Galerkin method.

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

## 7.13.2 Constructor & Destructor Documentation

DG ( Mesh & ms, size\_t degree = 1 )

Constructor with mesh and degree of the method.

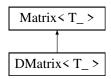
#### **Parameters**

in	ms	Mesh instance
in	degree	Polynomial degree of the DG method [Default: 1]

# 7.14 DMatrix $< T_- >$ Class Template Reference

To handle dense matrices.

Inheritance diagram for DMatrix $< T_- >$ :



## **Public Member Functions**

• DMatrix ()

Default constructor.

• DMatrix (size\_t nr)

Constructor for a matrix with nr rows and nr columns.

• DMatrix (size\_t nr, size\_t nc)

Constructor for a matrix with nr rows and nc columns.

• DMatrix (Vect< T<sub>-</sub> > &v)

Constructor that uses a Vect instance. The class uses the memory space occupied by this vector.

• DMatrix (const DMatrix < T<sub>−</sub> > &m)

Copy Constructor.

• DMatrix (Mesh &mesh, size\_t dof=0, int is\_diagonal=false)

Constructor using mesh to initialize structure of matrix.

• ∼DMatrix ()

Destructor.

void setDiag ()

Store diagonal entries in a separate internal vector.

void setDiag (const T<sub>-</sub> &a)

Set matrix as diagonal and assign its diagonal entries as a constant.

• void setDiag (const vector  $< T_- > &d$ )

Set matrix as diagonal and assign its diagonal entries.

• void setSize (size\_t size)

*Set size* (number of rows) of matrix.

• void setSize (size\_t nr, size\_t nc)

Set size (number of rows and columns) of matrix.

• void getColumn (size\_t j, Vect< T\_ > &v) const

Get j-th column vector.

Vect< T<sub>-</sub> > getColumn (size\_t j) const

Get j-th column vector.

• void getRow (size\_t i, Vect< T\_ > &v) const

Get i-th row vector.

• Vect< T\_> getRow (size\_t i) const

Get i-th row vector.

• void set (size\_t i, size\_t j, const T\_ &val)

Assign a constant value to an entry of the matrix.

• void reset ()

 $Set\ matrix\ to\ 0\ and\ reset\ factorization\ parameter.$ 

• void setRow (size\_t i, const Vect< T\_> &v)

Copy a given vector to a prescribed row in the matrix.

• void setColumn (size\_t j, const Vect< T\_ > &v)

Copy a given vector to a prescribed column in the matrix.

void MultAdd (T<sub>-</sub> a, const Vect< T<sub>-</sub> > &x, Vect< T<sub>-</sub> > &y) const

Multiply matrix by vector  $\mathbf{a} * \mathbf{x}$  and add result to  $\mathbf{y}$ .

• void MultAdd (const Vect< T<sub>-</sub> > &x, Vect< T<sub>-</sub> > &y) const

Multiply matrix by vector x and add result to y.

• void Mult (const Vect<  $T_->$  &x, Vect<  $T_->$  &y) const

Multiply matrix by vector x and save result in y.

• void TMult (const Vect<  $T_-> &x$ , Vect<  $T_-> &y$ ) const

Multiply transpose of matrix by vector x and add result in y.

• void add (size\_t i, size\_t j, const T\_ &val)

Add constant val to entry (i, j) of the matrix.

• void Axpy  $(T_- a, const DMatrix < T_- > &m)$ 

Add to matrix the product of a matrix by a scalar.

• void Axpy (T<sub>-</sub> a, const Matrix < T<sub>-</sub> > \*m)

Add to matrix the product of a matrix by a scalar.

```
• int setQR ()
      Construct a QR factorization of the matrix.
• int setTransQR ()
      Construct a QR factorization of the transpose of the matrix.
• int solveQR (const Vect< T_-> &b, Vect< T_-> &x)
      Solve a linear system by QR decomposition.
• int solveTransQR (const Vect< T_-> &b, Vect< T_-> &x)
      Solve a transpose linear system by QR decomposition.
• T_ operator() (size_t i, size_t j) const
      Operator () (Constant version). Return a(i, j)
• T_ & operator() (size_t i, size_t j)
      Operator () (Non constant version). Return a(i,j)
• int setLU ()
      Factorize the matrix (LU factorization)
• int setTransLU ()
      Factorize the transpose of the matrix (LU factorization)
• int solve (Vect< T_> &b, bool fact=true)
      Solve linear system.
• int solveTrans (Vect< T_> &b, bool fact=true)
      Solve the transpose linear system.
• int solve (const Vect< T<sub>-</sub> > &b, Vect< T<sub>-</sub> > &x, bool fact=true)
      Solve linear system.
• int solveTrans (const Vect< T_> &b, Vect< T_> &x, bool fact=true)
      Solve the transpose linear system.
• DMatrix & operator= (DMatrix < T_ > &m)
      Operator =
• DMatrix & operator+= (const DMatrix < T<sub>-</sub> > &m)
      Operator +=.
• DMatrix & operator-= (const DMatrix < T_ > &m)
      Operator -=.
• DMatrix & operator= (const T<sub>-</sub> &x)
      Operator =

    DMatrix & operator*= (const T<sub>-</sub> &x)

      Operator *=
• DMatrix & operator+= (const T<sub>-</sub> &x)
      Operator +=
• DMatrix & operator-= (const T_ &x)
      Operator -=
• T<sub>-</sub> * getArray () const
      Return matrix as C-Array.
• T_get (size_t i, size_t j) const
      Return entry (i, j) of matrix.
```

## 7.14.1 Detailed Description

template<class T\_> class OFELI::DMatrix< T\_>

To handle dense matrices.

This class enables storing and manipulating general dense matrices. Matrices can be square or rectangle ones.

**Template Parameters** 

$T \leftarrow$	Data type (double, float, complex <double>,)</double>
_←	

Author

Rachid Touzani

Copyright

GNU Lesser Public License

## 7.14.2 Constructor & Destructor Documentation

#### DMatrix ( )

Default constructor.

Initializes a zero-dimension matrix.

## DMatrix ( size\_t nr )

Constructor for a matrix with nr rows and nr columns.

Matrix entries are set to 0.

## DMatrix ( size\_t nr, size\_t nc )

Constructor for a matrix with nr rows and nc columns.

Matrix entries are set to 0.

## DMatrix ( Vect< T $_->$ & v )

Constructor that uses a Vect instance. The class uses the memory space occupied by this vector.

Parameters

in	v	Vector to copy

## DMatrix ( const DMatrix $< T_- > & m$ )

Copy Constructor.

in m	Matrix to copy
------	----------------

## DMatrix ( Mesh & mesh, size\_t dof = 0, int is\_diagonal = false )

Constructor using mesh to initialize structure of matrix.

#### **Parameters**

in	mesh	Mesh instance for which matrix graph is determined.
in	dof	Option parameter, with default value 0.
		dof=1 means that only one degree of freedom for each node (or element or
		side) is taken to determine matrix structure. The value dof=0 means that
		matrix structure is determined using all DOFs.
in	is_diagonal	Boolean argument to say is the matrix is actually a diagonal matrix or not.

## 7.14.3 Member Function Documentation

## void setDiag ( const T\_ & a )

Set matrix as diagonal and assign its diagonal entries as a constant.

### Parameters

:	in	а	Value to assign to all diagonal entries
---	----	---	---

## void setDiag ( const vector $< T_- > & d$ )

Set matrix as diagonal and assign its diagonal entries.

### Parameters

in	d	Vector entries to assign to matrix diagonal entries
----	---	---

## void setSize ( size\_t size )

Set size (number of rows) of matrix.

### Parameters

in	size	Number of rows and columns.
T11	5120	i validet of fows and columns.

### void setSize ( size\_t nr, size\_t nc )

Set size (number of rows and columns) of matrix.

## 7.14. DMATRIX < T\_ > CLASS TEMPLATE REFERENCE APTER 7. CLASS DOCUMENTATION

#### **Parameters**

in	nr	Number of rows.
in	пс	Number of columns.

## void getColumn ( size\_t j, Vect< $T_- > \& v$ ) const

Get j-th column vector.

#### **Parameters**

in	j	Index of column to extract	
out	v	Reference to Vect instance where the column is stored	

#### Remarks

Vector v does not need to be sized before. It is resized in the function

## $Vect < T_- > getColumn ( size_t j ) const$

Get j-th column vector.

#### **Parameters**

in	j	Index of column to extract
----	---	----------------------------

### Returns

Vect instance where the column is stored

### Remarks

Vector v does not need to be sized before. It is resized in the function

## void getRow ( size\_t i, Vect< T $_->$ & v ) const

Get i-th row vector.

#### **Parameters**

in	i	Index of row to extract	
out	v	Reference to Vect instance where the row is stored	

### Remarks

Vector v does not need to be sized before. It is resized in the function

## $Vect < T_{-} > getRow ( size_t i ) const$

Get i-th row vector.

## CHAPTER 7. CLASS DOCUMENTATION. DMATRIX $< T_- >$ CLASS TEMPLATE REFERENCE

#### Parameters

in i	Index of row to extract
------	-------------------------

#### Returns

Vect instance where the row is stored

#### Remarks

Vector v does not need to be sized before. It is resized in the function

## void set ( size\_t i, size\_t j, const T\_ & val ) [virtual]

Assign a constant value to an entry of the matrix.

### Parameters

in	i	row index of matrix
in	j	column index of matrix
in	val	Value to assign to a(i,j).

Implements Matrix  $< T_- >$ .

## void reset( ) [virtual]

Set matrix to 0 and reset factorization parameter.

### Warning

This function must be used if after a factorization, the matrix has modified

Reimplemented from Matrix $< T_- >$ .

## void setRow ( size\_t i, const Vect< $T_- > \& v$ )

Copy a given vector to a prescribed row in the matrix.

### Parameters

in	i	row index to be assigned	
in	v	Vect instance to copy	

## void setColumn ( size\_t j, const Vect< $T_- > \& v$ )

Copy a given vector to a prescribed column in the matrix.

### Parameters

in	j	column index to be assigned
in	v	Vect instance to copy

### 7.14. DMATRIX < T\_ > CLASS TEMPLATE REFERENCE APTER 7. CLASS DOCUMENTATION

## void MultAdd ( $T_-a$ , const Vect< $T_- > \& x$ , Vect< $T_- > \& y$ ) const [virtual]

Multiply matrix by vector a\*x and add result to y.

#### Parameters

in	а	constant to multiply by	
in	x	Vector to multiply by a	
in,out	y	on input, vector to add to. On output, result.	

Implements Matrix $< T_- >$ .

void MultAdd ( const Vect<  $T_-> \& x$ , Vect<  $T_-> \& y$  ) const [virtual]

Multiply matrix by vector x and add result to y.

#### Parameters

in	x	Vector to add to y	
in,out	y	on input, vector to add to. On output, result.	

Implements Matrix  $< T_- >$ .

void Mult ( const Vect<  $T_- > \& x$ , Vect<  $T_- > \& y$  ) const [virtual]

Multiply matrix by vector x and save result in y.

## Parameters

in	x	Vector to add to y
out	y	Result.

Implements Matrix $< T_->$ .

void TMult ( const Vect<  $T_- > & x$ , Vect<  $T_- > & y$  ) const [virtual]

Multiply transpose of matrix by vector x and add result in y.

### Parameters

in	x	Vector to add to y
in,out	у	on input, vector to add to. On output, result.

Implements Matrix  $< T_- >$ .

void add ( size\_t i, size\_t j, const T\_ & val ) [virtual]

Add constant val to entry (i,j) of the matrix.

in	i	row index
in	j	column index
in	val	Constant to add

Implements Matrix  $< T_- >$ .

# void Axpy ( $T_-a$ , const DMatrix< $T_- > \& m$ )

Add to matrix the product of a matrix by a scalar.

#### **Parameters**

in	а	Scalar to premultiply
in	m	Matrix by which a is multiplied. The result is added to current instance

## void Axpy ( $T_-a$ , const Matrix $< T_- > * m$ ) [virtual]

Add to matrix the product of a matrix by a scalar.

#### **Parameters**

in	а	Scalar to premultiply
in	m	Matrix by which a is multiplied. The result is added to current instance

Implements Matrix  $< T_- >$ .

## int setQR ( )

Construct a QR factorization of the matrix.

This function constructs the QR decomposition using the Householder method. The upper triangular matrix R is returned in the upper triangle of the current matrix, except for the diagonal elements of R which are stored in an internal vector. The orthogonal matrix Q is represented as a product of n-1 Householder matrices Q1 . . . Qn-1, where Qj = 1 - uj.uj/cj. The i-th component of uj is zero for i = 1, ..., j-1 while the nonzero components are returned in a[i][j] for i = j, ..., n.

#### Returns

0 if the decomposition was successful, k is the k-th row is singular

## Remarks

The matrix can be square or rectangle

#### int setTransQR ( )

Construct a QR factorization of the transpose of the matrix.

This function constructs the QR decomposition using the Householder method. The upper triangular matrix R is returned in the upper triangle of the current matrix, except for the diagonal elements of R which are stored in an internal vector. The orthogonal matrix Q is represented as a

product of n-1 Householder matrices Q1 . . . Qn-1, where Qj = 1 - uj.uj /cj . The i-th component of uj is zero for i = 1, ..., j-1 while the nonzero components are returned in a[i][j] for i = j, ..., n.

#### Returns

0 if the decomposition was successful, k is the k-th row is singular

#### Remarks

The matrix can be square or rectangle

# int solveQR ( const Vect< $T_-$ > & b, Vect< $T_-$ > & x)

Solve a linear system by QR decomposition.

This function constructs the QR decomposition, if this was not already done by using the member function QR and solves the linear system

#### **Parameters**

in	b	Right-hand side vector
out	x	Solution vector. Must have been sized before using this function.

#### Returns

The same value as returned by the function QR

## int solveTransQR ( const Vect< T $_->$ & b, Vect< T $_->$ & x )

Solve a transpose linear system by QR decomposition.

This function constructs the QR decomposition, if this was not already done by using the member function QR and solves the linear system

# Parameters

in	b	Right-hand side vector
out	x	Solution vector. Must have been sized before using this function.

#### Returns

The same value as returned by the function QR

#### T\_operator() ( size\_t i, size\_t j ) const [virtual]

Operator () (Constant version). Return a(i,j)

in	i	row index
in	j	column index

Implements Matrix  $< T_- >$ .

## T\_& operator() ( size\_t i, size\_t j ) [virtual]

Operator () (Non constant version). Return a(i,j)

#### **Parameters**

in	i	row index
in	j	column index

Implements Matrix  $< T_- >$ .

#### int setLU ( )

Factorize the matrix (LU factorization)

LU factorization of the matrix is realized. Note that since this is an in place factorization, the contents of the matrix are modified.

#### Returns

- 0 if factorization was normally performed,
- n if the n-th pivot is null.

#### Remarks

A flag in this class indicates after factorization that this one has been realized, so that, if the member function solve is called after this no further factorization is done.

## int setTransLU()

Factorize the transpose of the matrix (LU factorization)

LU factorization of the transpose of the matrix is realized. Note that since this is an in place factorization, the contents of the matrix are modified.

### Returns

- 0 if factorization was normally performed,
- n if the n-th pivot is null.

## Remarks

A flag in this class indicates after factorization that this one has been realized, so that, if the member function solve is called after this no further factorization is done.

## int solve ( Vect< T $_->$ & b, bool fact = true ) [virtual]

Solve linear system.

The linear system having the current instance as a matrix is solved by using the LU decomposition. Solution is thus realized after a factorization step and a forward/backward substitution step. The factorization step is realized only if this was not already done.

Note that this function modifies the matrix contents is a factorization is performed. Naturally, if the the matrix has been modified after using this function, the user has to refactorize it using the function setLU. This is because the class has no non-expensive way to detect if the matrix has been modified. The function setLU realizes the factorization step only.

in,out	b	Vect instance that contains right-hand side on input and solution on output.
in	fact	Set true if matrix is to be factorized (Default value), false if not

#### Returns

- 0 if solution was normally performed,
- n if the n-th pivot is null.

Implements Matrix $< T_- >$ .

#### int solveTrans ( Vect< T $_->$ & b, bool fact = true )

Solve the transpose linear system.

The linear system having the current instance as a transpose matrix is solved by using the LU decomposition. Solution is thus realized after a factorization step and a forward/backward substitution step. The factorization step is realized only if this was not already done.

Note that this function modifies the matrix contents is a factorization is performed. Naturally, if the the matrix has been modified after using this function, the user has to refactorize it using the function setLU. This is because the class has no non-expensive way to detect if the matrix has been modified. The function setLU realizes the factorization step only.

#### **Parameters**

in,out	b	Vect instance that contains right-hand side on input and solution on output.
in	fact	Set true if matrix is to be factorized (Default value), false if not

## Returns

- 0 if solution was normally performed,
- n if the n-th pivot is null.

#### int solve ( const Vect< T $_->$ & b, Vect< T $_->$ & x, bool fact = true ) [virtual]

Solve linear system.

The linear system having the current instance as a matrix is solved by using the LU decomposition. Solution is thus realized after a factorization step and a forward/backward substitution step. The factorization step is realized only if this was not already done.

Note that this function modifies the matrix contents is a factorization is performed. Naturally, if the the matrix has been modified after using this function, the user has to refactorize it using the function setLU. This is because the class has no non-expensive way to detect if the matrix has been modified. The function setLU realizes the factorization step only.

in	b	Vect instance that contains right-hand side.
out	x	Vect instance that contains solution
in	fact	Set true if matrix is to be factorized (Default value), false if not

#### Returns

- 0 if solution was normally performed,
- n if the n-th pivot is null.

Implements Matrix $< T_- >$ .

# int solveTrans ( const Vect< T $_->$ & b, Vect< T $_->$ & x, bool fact = true )

Solve the transpose linear system.

The linear system having the current instance as a transpose matrix is solved by using the LU decomposition. Solution is thus realized after a factorization step and a forward/backward substitution step. The factorization step is realized only if this was not already done.

Note that this function modifies the matrix contents is a factorization is performed. Naturally, if the the matrix has been modified after using this function, the user has to refactorize it using the function setLU. This is because the class has no non-expensive way to detect if the matrix has been modified. The function setLU realizes the factorization step only.

#### **Parameters**

in	b	Vect instance that contains right-hand side.
out	x	Vect instance that contains solution
in	fact	Set true if matrix is to be factorized (Default value), false if not

#### Returns

- 0 if solution was normally performed,
- n if the n-th pivot is null.

# DMatrix& operator= ( DMatrix $< T_- > & m$ )

Operator =

Copy matrix m to current matrix instance.

#### DMatrix& operator+= ( const DMatrix $< T_- > \& m$ )

Operator +=.

Add matrix m to current matrix instance.

## DMatrix& operator== ( const DMatrix $< T_- > \& m$ )

Operator -=.

Subtract matrix m from current matrix instance.

# DMatrix& operator= ( const $T_- \& x$ )

Operator =

Assign matrix to identity times x

# DMatrix& operator\*= ( const $T_- \& x$ )

Operator \*=

Premultiply matrix entries by constant value x.

### DMatrix& operator+= ( const $T_- \& x$ )

Operator +=

Add constant value x to matrix entries

## DMatrix& operator== ( const $T_- \& x$ )

Operator -=

Subtract constant value x from matrix entries.

# T\_\* getArray ( ) const

Return matrix as C-Array.

Matrix is stored row by row.

# 7.15 Domain Class Reference

To store and treat finite element geometric information.

## **Public Member Functions**

• Domain ()

Constructor of a null domain.

• Domain (const string &file)

Constructor with an input file.

• ~Domain ()

Destructor.

• void setFile (string file)

Set file containing Domain data.

void setDim (size\_t d)

 $Set\ space\ dimension.$ 

• size\_t getDim () const

Return space dimension.

void setNbDOF (size\_t n)

Set number of degrees of freedom.size\_t getNbDOF () const

Return number of degrees of freedom.

• size\_t getNbVertices () const

Return number of vertices.

• size\_t getNbLines () const

Return number of lines.

• size\_t getNbContours () const

Return number of contours.

• size\_t getNbHoles () const

Return number of holes.

• size\_t getNbSubDomains () const

Return number of sub-domains.

• int get ()

Read domain data interactively.

• void get (const string &file)

Read domain data from a data file.

• Mesh & getMesh () const

Return reference to generated Mesh instance.

void genGeo (string file)

Generate geometry file.

• void genMesh ()

Generate 2-D mesh.

void genMesh (const string &file)

Generate 2-D mesh and save in file (OFELI format)

• void genMesh (string geo\_file, string bamg\_file, string mesh\_file)

Generate 2-D mesh and save geo, bamg and mesh file (OFELI format)

• void generateMesh ()

Generate 2-D mesh using the BAMG mesh generator.

• Domain & operator\*= (real\_t a)

Operator \*=

• void insertVertex (real\_t x, real\_t y, real\_t h, int code)

Insert a vertex.

• void insertVertex (real\_t x, real\_t y, real\_t z, real\_t h, int code)

*Insert a vertex* (3-D case)

• void insertLine (size\_t n1, size\_t n2, int c)

Insert a straight line.

• void insertLine (size\_t n1, size\_t n2, int dc, int nc)

Insert a straight line.

• void insertCircle (size\_t n1, size\_t n2, size\_t n3, int c)

Insert a circluar arc.

• void insertCircle (size\_t n1, size\_t n2, size\_t n3, int dc, int nc)

Insert a circluar arc.

• void insertRequiredVertex (size\_t v)

*Insert a required (imposed) vertex.* 

• void insertRequiredEdge (size\_t e)

Insert a required (imposed) edge (or line)

• void insertSubDomain (size\_t n, int code)

Insert subdomain.

• void insertSubDomain (size\_t ln, int orient, int code)

Insert subdomain.

• void setNbDOF (int nb\_dof)

Set Number of degrees of freedom per node.

Point< real\_t > getMinCoord () const

Return minimum coordinates of vertices.

Point < real\_t > getMaxCoord () const
 Return maximum coordinates of vertices.

• real\_t getMinh () const

Return minimal value of mesh size.

• void setOutputFile (string file)

Define output mesh file.

# 7.15.1 Detailed Description

To store and treat finite element geometric information.

This class is essentially useful to construct data for mesh generators.

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

## 7.15.2 Constructor & Destructor Documentation

## Domain ( )

Constructor of a null domain.

This constructor assigns maximal values of parameters.

## Domain (const string & file)

Constructor with an input file.

**Parameters** 

in	file	Input file in the XML format defining the domain
----	------	--

## 7.15.3 Member Function Documentation

# void get ( const string & file )

Read domain data from a data file.

Parameters

in	file	Input file in Domain XML format	
----	------	---------------------------------	--

# void genMesh ( const string & file )

Generate 2-D mesh and save in file (OFELI format)

**Parameters** 

	in file	File where the generated mesh is saved	]
--	---------	--	---

# void genMesh ( string geo\_file, string bamg\_file, string mesh\_file )

Generate 2-D mesh and save geo, bamg and mesh file (OFELI format)

in geo_file C		Geo file	
in	bamg_file	Bamg file	
in	mesh_file	File where the generated mesh is saved	

# Domain& operator\*= ( real\_t a )

## Operator \*=

Rescale domain coordinates by myltiplying by a factor

#### Parameters

	in	а	Value to multiply by
--	----	---	----------------------

# void insertVertex ( real\_t x, real\_t y, real\_t h, int code )

Insert a vertex.

## Parameters

in	x	x-coordinate of vertex
in	y	y-coordinate of vertex
in	h	mesh size around vertex
in	code	code of coordinate

# void insertVertex ( real\_t x, real\_t y, real\_t z, real\_t h, int code )

Insert a vertex (3-D case)

## Parameters

in	х	x-coordinate of vertex	
in	y	y-coordinate of vertex	
in	Z	z-coordinate of vertex	
in	h	mesh size around vertex	
in	code	code of coordinate	

# void insertLine ( size\_t n1, size\_t n2, int c )

Insert a straight line.

in	n1	Label of the first vertex of line
in	n2	Label of the second vertex of line

in	С	Code to associate to created nodes (Dirichlet) or sides (Neumann) if $< 0$	
----	---	--	--

# void insertLine ( size\_t n1, size\_t n2, int dc, int nc )

Insert a straight line.

#### Parameters

in	n1	Label of the first vertex of line	
in	n2	Label of the second vertex of line	
in	dc	Code to associate to created nodes (Dirichlet)	
in	пс	Code to associate to created sides (Neumann)	

# void insertCircle ( size\_t n1, size\_t n2, size\_t n3, int c )

Insert a circluar arc.

# Parameters

in	n1	Label of vertex defining the first end of the arc	
in	n2	Label of vertex defining the second end of the arc	
in	п3	Label of vertex defining the center of the arc	
in	С	Code to associate to created nodes (Dirichlet) or sides (Neumann) if $< 0$	

# void insertCircle ( size\_t n1, size\_t n2, size\_t n3, int dc, int nc )

Insert a circluar arc.

## Parameters

in	n1	Label of vertex defining the first end of the arc	
in	n2	Label of vertex defining the second end of the arc	
in	п3	Label of vertex defining the center of the arc	
in	dc	Code to associate to created nodes (Dirichlet)	
in	пс	Code to associate to created sides (Neumann)	

# void insertRequiredVertex ( $size_t v$ )

Insert a required (imposed) vertex.

ſ	in	v	Label of vertex

# CHAPTER 7. CLASS DOCUMENTA $T_100$ N DSMATRIX $< T_- > CLASS$ TEMPLATE REFERENCE

# void insertRequiredEdge ( size\_t e )

Insert a required (imposed) edge (or line)

## Parameters

in	e	Label of line
----	---	---------------

# void insertSubDomain ( size\_t n, int code )

Insert subdomain.

#### **Parameters**

in	п	
in	code	

# void insertSubDomain ( size\_t ln, int orient, int code )

Insert subdomain.

## Parameters

in	ln	Line label
in	orient	Orientation (1 or -1)
in	code	Subdomain code or reference

# void setOutputFile ( string file )

Define output mesh file.

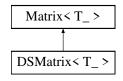
## Parameters

	in	file	String defining output mesh file
--	----	------	----------------------------------

# 7.16 DSMatrix $< T_- >$ Class Template Reference

To handle symmetric dense matrices.

Inheritance diagram for DSMatrix $< T_- >$ :



258

## **Public Member Functions**

• DSMatrix ()

Default constructor.

DSMatrix (size\_t dim)

Constructor that for a symmetric matrix with given number of rgows.

• DSMatrix (const DSMatrix < T\_> &m)

Copy Constructor.

• DSMatrix (Mesh &mesh, size\_t dof=0, int is\_diagonal=false)

Constructor using mesh to initialize matrix.

• ~DSMatrix ()

Destructor.

• void setDiag ()

Store diagonal entries in a separate internal vector.

• void setSize (size\_t dim)

Set size (number of rows) of matrix.

• void set (size\_t i, size\_t j, const T\_ &val)

Assign constant to entry (i, j) of the matrix.

• void getColumn (size\_t j, Vect< T\_ > &v) const

Get j-th column vector.

• Vect< T\_> getColumn (size\_t j) const

Get j-th column vector.

• void getRow (size\_t i, Vect< T\_ > &v) const

Get i-th row vector.

• Vect< T\_> getRow (size\_t i) const

Get i-th row vector.

• void setRow (size\_t i, const Vect< T\_> &v)

Copy a given vector to a prescribed row in the matrix.

• void setColumn (size\_t j, const Vect< T\_> &v)

Copy a given vector to a prescribed column in the matrix.

void setDiag (const T<sub>-</sub> &a)

Set matrix as diagonal and assign its diagonal entries as a constant.

void setDiag (const vector < T<sub>-</sub> > &d)

Set matrix as diagonal and assign its diagonal entries.

void add (size\_t i, size\_t j, const T\_ &val)

Add constant to an entry of the matrix.

• T\_ operator() (size\_t i, size\_t j) const

Operator () (Constant version).

• T\_ & operator() (size\_t i, size\_t j)

Operator () (Non constant version).

• DSMatrix< T<sub>-</sub> > & operator= (const DSMatrix< T<sub>-</sub> > &m)

Operator = Copy matrix m to current matrix instance.

• DSMatrix< T<sub>-</sub> > & operator= (const T<sub>-</sub> &x)

Operator = Assign matrix to identity times x.

• DSMatrix & operator+= (const T<sub>-</sub> &x)

Operator +=.

• DSMatrix & operator== (const T\_ &x)

*Operator* -=.

• int setLDLt ()

Factorize matrix  $(LDL^T)$ 

• void MultAdd (const Vect< T\_> &x, Vect< T\_> &y) const

Multiply matrix by vector a\*x and add result to y.

• void MultAdd (T<sub>-</sub> a, const Vect< T<sub>-</sub> > &x, Vect< T<sub>-</sub> > &y) const

Multiply matrix by vector  $\mathbf{a} * \mathbf{x}$  and add to  $\mathbf{y}$ .

• void Mult (const Vect< T $_->$  &x, Vect< T $_->$  &y) const

Multiply matrix by vector x and save result in y.

• void TMult (const Vect<  $T_->$  &x, Vect<  $T_->$  &y) const

Multiply transpose of matrix by vector x and add result in y.

• void Axpy (T<sub>-</sub> a, const DSMatrix < T<sub>-</sub> > &m)

Add to matrix the product of a matrix by a scalar.

• void Axpy (T<sub>-</sub> a, const Matrix < T<sub>-</sub> > \*m)

Add to matrix the product of a matrix by a scalar.

• int solve (Vect< T<sub>-</sub> > &b, bool fact=true)

Solve linear system.

• int solve (const Vect< T\_> &b, Vect< T\_> &x, bool fact=true)

Solve linear system.

• T<sub>-</sub> \* getArray () const

Return matrix as C-Array. Matrix is stored row by row. Only lower triangle is stored.

• T<sub>-</sub> get (size\_t i, size\_t j) const

Return entry (i, j) of matrix.

# 7.16.1 Detailed Description

template < class T\_> class OFELI::DSMatrix < T\_>

To handle symmetric dense matrices.

This class enables storing and manipulating symmetric dense matrices.

**Template Parameters** 

$T \leftarrow$	Data type (double, float, complex <double>,)</double>
_←	

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

## 7.16.2 Constructor & Destructor Documentation

DSMatrix ( size\_t dim )

Constructor that for a symmetric matrix with given number of rqows.

# 7.16. DSMATRIX< T $_->$ CLASS TEMPLATE REFERENCEPTER 7. CLASS DOCUMENTATION

#### Parameters

in dim Number of row	rs
----------------------	----

# DSMatrix (const DSMatrix $< T_- > & m$ )

Copy Constructor.

# Parameters

in	m	DSMatrix instance to copy
----	---	---------------------------

# DSMatrix ( Mesh & mesh, size\_t dof = 0, int is\_diagonal = false )

Constructor using mesh to initialize matrix.

# Parameters

in	mesh	Mesh instance for which matrix graph is determined.
in	dof	Option parameter, with default value 0. dof=1 means that only one degree of freedom for each node (or element or side) is taken to determine matrix structure. The value dof=0 means that matrix structure is determined using all DOFs.
in	is_diagonal	Boolean argument to say is the matrix is actually a diagonal matrix or not.

# 7.16.3 Member Function Documentation

void setSize ( size\_t dim )

Set size (number of rows) of matrix.

Parameters

in	dim	Number of rows and columns.
----	-----	-----------------------------

# void set ( size\_t i, size\_t j, const $T_- \& val$ ) [virtual]

Assign constant to entry (i,j) of the matrix.

## Parameters

in	i	row index
in	j	column index
in	val	value to assign to a(i,j)

Implements Matrix  $< T_- >$ .

# void getColumn ( size\_t j, Vect< $T_- > \& v$ ) const

Get j-th column vector.

#### Parameters

in	j	Index of column to extract
out	v	Reference to Vect instance where the column is stored

## Remarks

Vector v does not need to be sized before. It is resized in the function

# $Vect < T_- > getColumn ( size_t j ) const$

Get j-th column vector.

## Parameters

i:	n	j	Index of column to extract
----	---	---	----------------------------

#### Returns

Vect instance where the column is stored

## Remarks

Vector v does not need to be sized before. It is resized in the function

# void getRow ( size\_t $i_r$ Vect< T\_ > & v ) const

Get i-th row vector.

## Parameters

in	i	Index of row to extract
out	v	Reference to Vect instance where the row is stored

## Remarks

Vector v does not need to be sized before. It is resized in the function

# $Vect < T_{-} > getRow ( size_t i ) const$

Get i-th row vector.

in	i	Index of row to extract

# 7.16. DSMATRIX< T $_->$ CLASS TEMPLATE REFERENCEPTER 7. CLASS DOCUMENTATION

#### Returns

Vect instance where the row is stored

#### Remarks

Vector v does not need to be sized before. It is resized in the function

## void setRow ( size\_t i, const Vect< $T_- > \& v$ )

Copy a given vector to a prescribed row in the matrix.

#### Parameters

in	i	row index to be assigned
in	v	Vect instance to copy

# void setColumn ( size\_t j, const Vect< $T_- > \& v$ )

Copy a given vector to a prescribed column in the matrix.

#### Parameters

in	j	column index to be assigned
in	v	Vect instance to copy

# void setDiag ( const $T_- \& a$ )

Set matrix as diagonal and assign its diagonal entries as a constant.

# Parameters

in	а	Value to assign to all diagonal entries
----	---	---

# void setDiag ( const vector $< T_- > & d$ )

Set matrix as diagonal and assign its diagonal entries.

#### **Parameters**

in	d	Vector entries to assign to matrix diagonal entries	1
----	---	---	---

# void add ( size\_t i, size\_t j, const T\_ & val ) [virtual]

Add constant to an entry of the matrix.

in	i	row index
in	j	column index
in	val	value to add to a(i,j)

Implements Matrix  $< T_- >$ .

# $T_{-}$ operator() ( $size_{-}t i$ , $size_{-}t j$ ) const [virtual]

Operator () (Constant version).

#### Parameters

in	i	Row index
in	j	Column index

Implements Matrix  $< T_- >$ .

## T\_& operator() ( size\_t i, size\_t j ) [virtual]

Operator () (Non constant version).

### Parameters

in	i	Row index
in	j	Column index

## Warning

To modify a value of an entry of the matrix it is safer not to modify both lower and upper triangles. Otherwise, wrong values will be assigned. If not sure, use the member functions set or add.

Implements Matrix  $< T_- >$ .

## DSMatrix& operator+= ( const $T_- \& x$ )

Operator +=.

Add constant value x to all matrix entries.

# DSMatrix& operator== ( const $T_- & x$ )

Operator -=.

Subtract constant value x from to all matrix entries.

## int setLDLt ( )

Factorize matrix (LDL<sup>T</sup>)

# 7.16. DSMATRIX< T $_->$ CLASS TEMPLATE REFERENCEPTER 7. CLASS DOCUMENTATION

## Returns

- 0, if factorization was normally performed,
- n, if the n-th pivot is null.

# $void\ MultAdd\ (\ T_-\textit{a, const}\ Vect < T_->\&\textit{x, Vect} < T_->\&\textit{y}\ )\ const\ \ [\texttt{virtual}]$

Multiply matrix by vector a\*x and add to y.

#### **Parameters**

in	а	Constant to multiply by matrix	
in	x	ector to multiply by matrix	
in,out	у	Vector to add to the result. y contains on output the result.	

Implements Matrix  $< T_- >$ .

# void TMult ( const Vect< T $_->$ & x, Vect< T $_->$ & y ) const [virtual]

Multiply transpose of matrix by vector x and add result in y.

#### **Parameters**

in	х	Vector to add to y
in,out	y	on input, vector to add to. On output, result.

Implements Matrix  $< T_- >$ .

# void Axpy ( $T_- a$ , const DSMatrix< $T_- > \& m$ )

Add to matrix the product of a matrix by a scalar.

# Parameters

in	а	Scalar to premultiply
in	m	Matrix by which a is multiplied. The result is added to current instance

# void Axpy ( $T_-a$ , const Matrix< $T_- > * m$ ) [virtual]

Add to matrix the product of a matrix by a scalar.

#### **Parameters**

in	а	Scalar to premultiply	
in	m	Matrix by which a is multiplied. The result is added to current instance	

Implements Matrix  $< T_- >$ .

## int solve ( Vect< $T_- > & b$ , bool fact = true ) [virtual]

Solve linear system.

The matrix is factorized using the LDLt (Crout) decomposition. If this one is already factorized, no further factorization is performed. If the matrix has been modified the user has to refactorize it using the function setLDLt.

#### **Parameters**

in,out	b	Vect instance that contains right-hand side on input and solution on output.
in	fact	Set true if matrix is to be factorized (Default value), false if not

#### Returns

- 0 if solution was normally performed,
- n if the n-th pivot is null.

Implements Matrix  $< T_- >$ .

int solve ( const Vect< T $_->$  & b, Vect< T $_->$  & x, bool fact = true ) [virtual]

Solve linear system.

The matrix is factorized using the LDLt (Crout) decomposition. If this one is already factorized, no further factorization is performed. If the matrix has been modified the user has to refactorize it using the function setLDLt.

### **Parameters**

in	b	Vect instance that contains right-hand side.	
out	x	Vect instance that contains solution	
in	fact	Set true if matrix is to be factorized (Default value), false if	

#### Returns

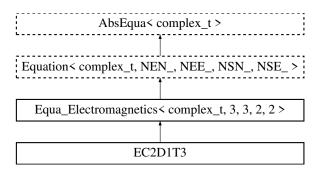
- 0 if solution was normally performed,
- n if the n-th pivot is null.

Implements Matrix  $< T_- >$ .

# 7.17 EC2D1T3 Class Reference

Eddy current problems in 2-D domains using solenoidal approximation.

Inheritance diagram for EC2D1T3:



## **Public Member Functions**

• EC2D1T3 ()

Default constructor.

• EC2D1T3 (Mesh &ms)

Constructor using mesh.

• EC2D1T3 (Mesh &ms, Vect< complex\_t > &u)

Constructor using mesh and solution vector.

• ~EC2D1T3 ()

Destructor.

• void setData (real\_t omega, real\_t volt)

Define data for equation.

• void build ()

Build the linear system of equations.

• void Magnetic (real\_t omega, real\_t coef=1.)

Add magnetic contribution to matrix.

• void Electric (real\_t coef=1.)

Add electric contribution to matrix.

real\_t Joule ()

Compute Joule density in element.

complex\_t IntegMF ()

Add element integral contribution.

• complex\_t IntegND (const Vect< complex\_t > &h)

Compute integral of normal derivative on edge.

• real\_t VacuumArea ()

Add contribution to vacuum area calculation.

## **Additional Inherited Members**

# 7.17.1 Detailed Description

Eddy current problems in 2-D domains using solenoidal approximation.

Builds finite element arrays for time harmonic eddy current problems in 2-D domains with solenoidal configurations (Magnetic field has only one nonzero component). Magnetic field is constant in the vacuum, and then zero in the outer vacuum.

Uses 3-Node triangles.

The unknown is the time-harmonic magnetic induction (complex valued).

#### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

## 7.17.2 Constructor & Destructor Documentation

#### EC2D1T3 (Mesh & ms)

Constructor using mesh.

#### Parameters

in ms Mesh instance
---------------------

# EC2D1T3 ( Mesh & ms, Vect< complex\_t > & u )

Constructor using mesh and solution vector.

#### **Parameters**

in	ms	Mesh instance	
in,out	и	Reference to solution vector instance	

# 7.17.3 Member Function Documentation

void setData ( real\_t omega, real\_t volt )

Define data for equation.

### Parameters

in	omega	Angular frequency
in	volt	Voltage

# void build ( )

Build the linear system of equations.

Before using this function, one must have properly selected appropriate options for:

- The choice of a steady state or transient analysis. By default, the analysis is stationary
- In the case of transient analysis, the choice of a time integration scheme and a lumped or consistent capacity matrix. If transient analysis is chosen, the lumped capacity matrix option is chosen by default, and the implicit Euler scheme is used by default for time integration.

## void Magnetic ( real\_t omega, real\_t coef = 1. )

Add magnetic contribution to matrix.

#### Parameters

in	omega	Angular frequency
in	coef	Coefficient to multiply by [Default: 1]

# void Electric ( real\_t coef = 1. )

Add electric contribution to matrix.

## **Parameters**

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

# complex\_t IntegND ( const Vect< complex\_t > & h )

Compute integral of normal derivative on edge.

#### **Parameters**

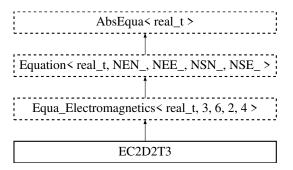
in	h	Vect instance containing magnetic field at nodes
----	---	--

## Note

This member function is to be called within each element, it detects boundary sides as the ones with nonzero code

# 7.18 EC2D2T3 Class Reference

Eddy current problems in 2-D domains using transversal approximation. Inheritance diagram for EC2D2T3:



# **Public Member Functions**

• EC2D2T3 ()

Default Constructor.

• EC2D2T3 (Mesh &ms)

Constructor using mesh.

• EC2D2T3 (Mesh &ms, Vect< real\_t > &u)

Constructor using mesh and solution vector.

• EC2D2T3 (const Side \*sd1, const Side \*sd2)

Constructor using two side data.

• ~EC2D2T3 ()

Destructor.

• void RHS (real\_t coef=1.)

Compute Contribution to Right-Hand Side.

void FEBlock (real\_t omega)

Compute Finite Element Diagonal Block.

void BEBlocks (size\_t n1, size\_t n2, SpMatrix< real\_t > &L, SpMatrix< real\_t > &U, Sp← Matrix< real\_t > &D)

Compute boundary element blocks.

• complex\_t Constant (real\_t omega, const Vect< real\_t > &u, complex\_t &I)

Compute constant to multiply by potential.

• real\_t MagneticPressure (const Vect< real\_t > &u)

Compute magnetic pressure in element.

#### **Additional Inherited Members**

## 7.18.1 Detailed Description

Eddy current problems in 2-D domains using transversal approximation.

Builds finite element arrays for time harmonic eddy current problems in 2-D domains with transversal configurations (Magnetic field has two nonzero components). Uses 3-Node triangles. The unknown is the time-harmonic scalar potential (complex valued).

Author

Rachid Touzani

Copyright

GNU Lesser Public License

## 7.18.2 Constructor & Destructor Documentation

EC2D2T3 ( Mesh & ms )

Constructor using mesh.

Parameters

in ms	Mesh instance
-------	---------------

## EC2D2T3 ( Mesh & ms, Vect< real\_t > & u )

Constructor using mesh and solution vector.

in	ms	Mesh instance	
in,out	и	Vect instance containing solution	

# 7.19 Edge Class Reference

To describe an edge.

## **Public Member Functions**

• Edge ()

Default Constructor.

• Edge (size\_t label)

Constructor with label.

• Edge (const Edge &ed)

Copy constructor.

• ~Edge ()

Destructor.

• void Add (Node \*node)

Insert a node at end of list of nodes of edge.

• void setLabel (size\_t i)

Assign label of edge.

void setFirstDOF (size\_t n)

Define First DOF.

void setNbDOF (size\_t nb\_dof)

Define number of DOF of edge.

• void DOF (size\_t i, size\_t dof)

Define label of DOF.

• void setDOF (size\_t &first\_dof, size\_t nb\_dof)

Define number of DOF.

• void setCode (size\_t dof, int code)

Assign code code to DOF number dof.

• void AddNeighbor (Side \*sd)

Add side pointed by sd to list of edge sides.

size\_t getLabel () const

Return label of edge.

• size\_t n () const

Return label of edge.

size\_t getNbEq () const

Return number of edge equations.

• size\_t getNbDOF () const

Return number of DOF.

• int getCode (size\_t dof=1) const

Return code for a given DOF of node.

• size\_t getDOF (size\_t i) const

Return label of i-th DOF.

• size\_t getFirstDOF () const

Return number of first dof of node.

• Node \* getPtrNode (size\_t i) const

List of element nodes.

Node \* operator() (size\_t i) const

Operator ().

• size\_t getNodeLabel (size\_t i) const

Return node label.

Side \* getNeighborSide (size\_t i) const

Return pointer to neighbor i-th side.

• int isOnBoundary () const

Say if current edge is a boundary edge or not.

• void setOnBoundary ()

Say that the edge is on the boundary.

Node \* operator() (size\_t i)

Operator ().

# 7.19.1 Detailed Description

To describe an edge.

Defines an edge of a 3-D finite element mesh. The edges are given in particular by a list of nodes. Each node can be accessed by the member function getPtrNode.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# 7.19.2 Constructor & Destructor Documentation

# Edge ( )

Default Constructor.

Initializes data to zero

Edge ( size\_t label )

Constructor with label.

Define an edge by giving its label

# 7.19.3 Member Function Documentation

void DOF ( size\_t i, size\_t dof )

Define label of DOF.

in	i	DOF index
in	dof	Its label

## void setDOF ( size\_t & first\_dof, size\_t nb\_dof )

Define number of DOF.

#### Parameters

in,out	first_dof	Label of the first DOF in input that is actualized
in	nb_dof	Number of DOF

# void setCode ( size\_t dof, int code )

Assign code code to DOF number dof.

# Parameters

in	dof	index of dof for assignment.
in	code	Value of code to assign.

# int getCode ( $size_t dof = 1$ ) const

Return code for a given DOF of node. Default value is 1

# Node\* operator() ( size\_t i ) const

Operator ().

Return pointer to node of local label i.

## size\_t getNodeLabel ( size\_t i ) const

Return node label.

## Parameters

in	i	Local label of node for which global label is returned

# int isOnBoundary ( ) const

Say if current edge is a boundary edge or not.

Note this information is available only if boundary edges were determined. See class Mesh

# Node\* operator() ( size\_t i )

Operator ().

Returns pointer to node of local label i

# 7.20 EdgeList Class Reference

Class to construct a list of edges having some common properties.

# **Public Member Functions**

• EdgeList (Mesh &ms)

Constructor using a Mesh instance.

• ∼EdgeList ()

Destructor.

• void selectCode (int code, int dof=1)

Select edges having a given code for a given degree of freedom.

• void unselectCode (int code, int dof=1)

Unselect edges having a given code for a given degree of freedom.

• size\_t getNbEdges () const

Return number of selected edges.

• void top ()

Reset list of edges at its top position (Non constant version)

• void top () const

Reset list of edges at its top position (Constant version)

• Edge \* get ()

Return pointer to current edge and move to next one (Non constant version)

• Edge \* get () const

Return pointer to current edge and move to next one (Constant version)

# 7.20.1 Detailed Description

Class to construct a list of edges having some common properties.

This class enables choosing multiple selection criteria by using function select... However, the intersection of these properties must be empty.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# 7.20.2 Member Function Documentation

void selectCode ( int code, int dof = 1 )

Select edges having a given code for a given degree of freedom.

## Parameters

in	code	Code that edges share
in	dof	Degree of Freedom label [Default: 1]

#### void unselectCode ( int code, int dof = 1 )

Unselect edges having a given code for a given degree of freedom.

in	code	Code of edges to exclude
in	dof	Degree of Freedom label [Default: 1]

# 7.21 EigenProblemSolver Class Reference

Class to find eigenvalues and corresponding eigenvectors of a given matrix in a generalized eigenproblem, *i.e.* Find scalars l and non-null vectors v such that  $[K]\{v\} = l[M]\{v\}$  where [K] and [M] are symmetric matrices. The eigenproblem can be originated from a PDE. For this, we will refer to the matrices K and M as *Stiffness* and *Mass* matrices respectively.

#### **Public Member Functions**

• EigenProblemSolver ()

Default constructor.

• EigenProblemSolver (DSMatrix < real\_t > &K, int n=0)

Constructor for a dense symmetric matrix that computes the eigenvalues.

EigenProblemSolver (SkSMatrix < real\_t > &K, SkSMatrix < real\_t > &M, int n=0)

Constructor for Symmetric Skyline Matrices.

• EigenProblemSolver (SkSMatrix < real\_t > &K, Vect < real\_t > &M, int n=0)

Constructor for Symmetric Skyline Matrices.

• EigenProblemSolver (DSMatrix< real\_t > &A, Vect< real\_t > &ev, int n=0)

Constructor for a dense matrix that compute the eigenvalues.

• EigenProblemSolver (AbsEqua < real\_t > &eq, bool lumped=true)

Consttuctor using partial differential equation.

• ~EigenProblemSolver ()

Destructor.

void setMatrix (SkSMatrix < real\_t > &K, SkSMatrix < real\_t > &M)

Set matrix instances (Symmetric matrices).

void setMatrix (SkSMatrix < real\_t > &K, Vect < real\_t > &M)

Set matrix instances (Symmetric matrices).

void setMatrix (DSMatrix < real\_t > &K)

Set matrix instance (Symmetric matrix).

void setPDE (AbsEqua < real\_t > &eq, bool lumped=true)

Define partial differential equation to solve.

• int run (int nb=0)

Run the eigenproblem solver.

void Assembly (const Element &el, real\_t \*eK, real\_t \*eM)

Assemble element arrays into global matrices.

void SAssembly (const Side &sd, real\_t \*sK)

Assemble side arrays into global matrix and right-hand side.

• int runSubSpace (size\_t nb\_eigv, size\_t ss\_dim=0)

Run the subspace iteration solver.

void setSubspaceDimension (int dim)

Define the subspace dimension.

void setMaxIter (int max\_it)

set maximal number of iterations.

• void setTolerance (real\_t eps)

set tolerance value

int checkSturm (int &nb\_found, int &nb\_lost)

Check how many eigenvalues have been found using Sturm sequence method.

• int getNbIter () const

Return actual number of performed iterations.

real\_t getEigenValue (int n) const

Return the n-th eigenvalue.

• void getEigenVector (int n, Vect< real\_t > &v) const

Return the n-th eigenvector.

# 7.21.1 Detailed Description

Class to find eigenvalues and corresponding eigenvectors of a given matrix in a generalized eigenproblem, *i.e.* Find scalars 1 and non-null vectors v such that  $[K]\{v\} = l[M]\{v\}$  where [K] and [M] are symmetric matrices. The eigenproblem can be originated from a PDE. For this, we will refer to the matrices K and M as *Stiffness* and *Mass* matrices respectively.

#### Author

Rachid Touzani

### Copyright

GNU Lesser Public License

## 7.21.2 Constructor & Destructor Documentation

EigenProblemSolver ( DSMatrix < real\_t > & K, int n = 0 )

Constructor for a dense symmetric matrix that computes the eigenvalues.

This constructor solves in place the eigenvalues problem and stores them in a vector (No need to use the function runSubSpace). The eigenvectors can be obtained by calling the member function getEigenVector.

#### **Parameters**

in	K	Matrix for which eigenmodes are sought.
in	n	Number of eigenvalues to extract. By default all eigenvalues are computed.

## EigenProblemSolver ( SkSMatrix < real\_t > & M, skSMatrix < real\_t > & M, int n = 0)

Constructor for Symmetric Skyline Matrices.

ir	K	"Stiffness" matrix
ir	M	"Mass" matrix
ir	n	Number of eigenvalues to extract. By default all eigenvalues are computed.

#### Note

The generalized eigenvalue problem is defined by Kx = aMx, where K and M are referred to as stiffness and mass matrix.

## EigenProblemSolver (SkSMatrix< real\_t > & K, Vect< real\_t > & M, int n = 0)

Constructor for Symmetric Skyline Matrices.

#### **Parameters**

in	K	"Stiffness" matrix	
in	M	Diagonal "Mass" matrix stored as a Vect instance	
in	n	Number of eigenvalues to extract. By default all eigenvalues are computed.	

#### Note

The generalized eigenvalue problem is defined by Kx = aMx, where K and M are referred to as stiffness and mass matrix.

# EigenProblemSolver ( DSMatrix< real\_t > & A, Vect< real\_t > & ev, int n = 0 )

Constructor for a dense matrix that compute the eigenvalues.

This constructor solves in place the eigenvalues problem and stores them in a vector (No need to use the function runSubSpace). The eigenvectors can be obtained by calling the member function getEigenVector.

### **Parameters**

in	Α	Matrix for which eigenmodes are sought.
in	ev	Vector containing all computed eigenvalues sorted increasingly.
in	n	Number of eigenvalues to extract. By default all eigenvalues are computed.

#### Remarks

The vector ev does not need to be sized before.

## EigenProblemSolver ( AbsEqua < real\_t > & eq, bool lumped = true )

Consttuctor using partial differential equation.

The used equation class must have been constructed using the Mesh instance

in	eq	Reference to equation instance
in	lumped	Mass matrix is lumped (true) or not (false) [Default: true]

# 7.21.3 Member Function Documentation

## void setMatrix ( SkSMatrix < real\_t > & K, SkSMatrix < real\_t > & M )

Set matrix instances (Symmetric matrices).

This function is to be used when the default constructor is applied. Case where the mass matrix is consistent.

#### **Parameters**

in	K	Stiffness matrix instance
in	M	Mass matrix instance

## void setMatrix ( SkSMatrix < real\_t > & K, Vect < real\_t > & M )

Set matrix instances (Symmetric matrices).

This function is to be used when the default constructor is applied. Case where the mass matrix is (lumped) diagonal and stored in a vector.

#### **Parameters**

ir	. K	Stiffness matrix instance	
ir	. M	Mass matrix instance where diagonal terms are stored as a vector.	

## void setMatrix ( DSMatrix < real\_t > & K )

Set matrix instance (Symmetric matrix).

This function is to be used when the default constructor is applied. Case of a standard (not generalized) eigen problem is to be solved

## Parameters

in	K	Stiffness matrix instance

# void setPDE ( AbsEqua< real\_t > & eq, bool lumped = true )

Define partial differential equation to solve.

The used equation class must have been constructed using the Mesh instance

#### **Parameters**

i	in	eq	Reference to equation instance	
i	in	lumped	Mass matrix is lumped (true) or not (false) [Default: true]	

## int run ( int nb = 0 )

Run the eigenproblem solver.

## 7.21. EIGENPROBLEMSOLVER CLASS REFERENCE HAPTER 7. CLASS DOCUMENTATION

#### **Parameters**

in	nb	Number of eigenvalues to be computed. By default, all eigenvalues are computed.	
----	----	---	--

# void Assembly ( const Element & el, real\_t \* eK, real\_t \* eM )

Assemble element arrays into global matrices.

This member function is to be called from finite element equation classes

#### **Parameters**

in	el	Reference to Element class	
in	еK	Pointer to element stiffness (or assimilated) matrix	
in	eМ	Pointer to element mass (or assimilated) matrix	

# void SAssembly ( const Side & sd, real\_t \* sK )

Assemble side arrays into global matrix and right-hand side.

This member function is to be called from finite element equation classes

#### **Parameters**

in	sd	Reference to Side class
in	sK	Pointer to side stiffness

# int runSubSpace ( size\_t nb\_eigv, size\_t ss\_dim = 0 )

Run the subspace iteration solver.

This function rune the Bathe subspace iteration method.

# Parameters

in	nb_eigv	Number of eigenvalues to be extracted	
in	ss_dim	Subspace dimension. Must be at least equal to the number eigenvalues to seek. [Default: nb_eigv]	

# Returns

1: Normal execution. Convergence has been achieved. 2: Convergence for eigenvalues has not been attained.

# void setSubspaceDimension (int dim)

Define the subspace dimension.

in	dim	Subspace dimension. Must be larger or equal to the number of wanted
		eigenvalues. By default this value will be set to the number of wanted
		eigenvalues

# void setTolerance ( real\_t eps )

set tolerance value

#### **Parameters**

i	.n	eps	Convergence tolerance for eigenvalues [Default: 1.e-8]	
---	----	-----	--	--

# int checkSturm ( int & nb\_found, int & nb\_lost )

Check how many eigenvalues have been found using Sturm sequence method.

#### **Parameters**

out	nb_found	number of eigenvalues actually found	
out	nb_lost	number of eigenvalues missing	

## Returns

- 0, Successful completion of subroutine.
  - 1, No convergent eigenvalues found.

# void getEigenVector ( int n, Vect< real\_t > & v ) const

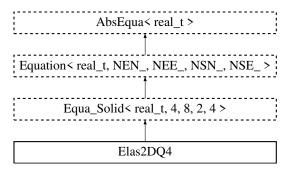
Return the n-th eigenvector.

# Parameters

in	n	Label of eigenvector (They are stored in ascending order of eigenvalues)
in,out	v	Vect instance where the eigenvector is stored.

# 7.22 Elas2DQ4 Class Reference

To build element equations for 2-D linearized elasticity using 4-node quadrilaterals. Inheritance diagram for Elas2DQ4:



### **Public Member Functions**

• Elas2DQ4 ()

Default Constructor.

Elas2DQ4 (Mesh &ms)

Constructor using Mesh instance.

• Elas2DQ4 (Mesh &ms, Vect< real\_t > &u)

Constructor using Mesh instance and solution vector.

• ~Elas2DQ4 ()

Destructor.

• void PlaneStrain ()

Set plane strain hypothesis.

• void PlaneStrain (real\_t E, real\_t nu)

Set plane strain hypothesis by giving values of Young's modulus and Poisson ratio.

• void PlaneStress ()

Set plane stress hypothesis.

• void PlaneStress (real\_t E, real\_t nu)

Set plane stress hypothesis by giving values of Young's modulus and Poisson ratio.

• void LMass (real\_t coef=1.)

Add element lumped mass contribution to element matrix after multiplication by coef [Default: 1].

• void Mass (real\_t coef=1.)

Add element consistent mass contribution to matrix and right-hand side after multiplication by coef [Default: 1].

• void Deviator (real\_t coef=1.)

Add element deviatoric matrix to element matrix after multiplication by coef [Default: 1].

• void Dilatation (real\_t coef=1.)

Add element dilatational contribution to element matrix after multiplication by coef [Default: 1].

void BodyRHS (const Vect< real\_t > &f)

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

void BoundaryRHS (const Vect< real\_t > &f)

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

void Strain (Vect< real\_t > &eps)

Calculate strains at element barycenters.

• void Stress (Vect< real\_t > &st, Vect< real\_t > &vm)

Calculate principal stresses and Von-Mises stress at element barycenter.

void Stress (Vect< real\_t > &sigma, Vect< real\_t > &s, Vect< real\_t > &st)

Calculate principal stresses and Von-Mises stress at element barycenter.

# **Additional Inherited Members**

# 7.22.1 Detailed Description

To build element equations for 2-D linearized elasticity using 4-node quadrilaterals.

This class enables building finite element arrays for linearized isotropic elasticity problem in 2-D domains using 4-Node quadrilaterals.

Unilateral contact is handled using a penalty function. Note that members calculating element arrays have as an argument a real coef that is multiplied by the contribution of the current element. This makes possible testing different algorithms.

# 7.22.2 Constructor & Destructor Documentation

# Elas2DQ4()

Default Constructor.

Constructs an empty equation.

# Elas2DQ4 ( Mesh & ms )

Constructor using Mesh instance.

#### **Parameters**

in   ms   Reference to Mesh instance	in ms	Reference to Mesh instance
--------------------------------------	-------	----------------------------

## Elas2DQ4 ( Mesh & ms, Vect< real\_t > & u )

Constructor using Mesh instance and solution vector.

## Parameters

in	ms	Reference to Mesh instance
in,out	и	Solution vector

#### 7.22.3 Member Function Documentation

## void PlaneStrain ( real\_t E, real\_t nu )

Set plane strain hypothesis by giving values of Young's modulus and Poisson ratio.

## Parameters

in	Ε	Young's modulus
in	пи	Poisson ratio

## void PlaneStress ( real\_t E, real\_t nu )

Set plane stress hypothesis by giving values of Young's modulus and Poisson ratio.

in	Ε	Young's modulus
in	пи	Poisson ratio

# void BodyRHS ( const Vect< real\_t > & f )

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

## Parameters

in	f	Vector containing source at nodes (DOF by DOF).
----	---	---

# void BoundaryRHS ( const Vect< real\_t > & f )

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

#### Parameters

in	f	Vector containing source at nodes (DOF by DOF).
----	---	---

# void Strain ( Vect< real\_t > & eps )

Calculate strains at element barycenters.

# Parameters

out	eps	Vector containing strains in elements

## Remarks

The instance of Elas2DQ4 must have been constructed using the constructor with Mesh instance and solution vector

# void Stress ( Vect< real\_t > & st, Vect< real\_t > & vm)

Calculate principal stresses and Von-Mises stress at element barycenter.

## Parameters

out	st	Vector containing principal stresses in elements
out	vm	Vector containing Von-Mises stresses in elements

## Remarks

The instance of Elas2DQ4 must have been constructed using the constructor with Mesh instance and solution vector

## void Stress ( Vect< real\_t > & sigma, Vect< real\_t > & s, Vect< real\_t > & st )

Calculate principal stresses and Von-Mises stress at element barycenter.

#### **Parameters**

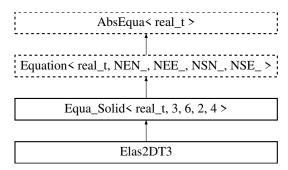
out	sigma	Vector containing principal stresses in elements
out	S	Vector containing principal stresses in elements
out	st	Value of Von-Mises stress in elements

#### Remarks

The instance of Elas2DQ4 must have been constructed using the constructor with Mesh instance and solution vector

# 7.23 Elas2DT3 Class Reference

To build element equations for 2-D linearized elasticity using 3-node triangles. Inheritance diagram for Elas2DT3:



# **Public Member Functions**

• Elas2DT3 ()

Default Constructor.

• Elas2DT3 (Mesh &ms)

Constructor using Mesh data.

• Elas2DT3 (Mesh &ms, Vect< real\_t > &u)

Constructor using Mesh data and solution vector.

• ~Elas2DT3 ()

Destructor.

• void Media (real\_t E, real\_t nu, real\_t rho)

Set media properties.

• void PlaneStrain ()

Set plane strain hypothesis.

• void PlaneStrain (real\_t E, real\_t nu)

Set plane strain hypothesis by giving values of Young's modulus E and Poisson ratio nu

• void PlaneStress ()

Set plane stress hypothesis.

• void PlaneStress (real\_t E, real\_t nu)

Set plane stress hypothesis by giving values of Young's modulus E and Poisson ratio nu

• void LMass (real\_t coef=1.)

Add element lumped mass contribution to element matrix after multiplication by coef

• void Mass (real\_t coef=1.)

Add element consistent mass contribution to element matrix after multiplication by coef

• void Deviator (real\_t coef=1.)

Add element deviatoric matrix to element matrix after multiplication by coef

• void Dilatation (real\_t coef=1.)

Add element dilatational contribution to element matrix after multiplication by coef

• void BodyRHS (const Vect< real\_t > &f)

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

void BoundaryRHS (const Vect< real\_t > &f)

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

• int Contact (real\_t coef=1.e07)

Penalty Signorini contact side contribution to matrix and right-hand side.

void Reaction (Vect< real\_t > &r)

Calculate reactions.

• void ContactPressure (const Vect< real\_t > &f, real\_t penal, Point< real\_t > &p)

Calculate contact pressure.

void Strain (Vect< real\_t > &eps)

Calculate strains in element.

void Stress (Vect< real\_t > &s, Vect< real\_t > &vm)

Calculate principal stresses and Von-Mises stress in element.

• void Periodic (real\_t coef=1.e20)

Add contribution of periodic boundary condition (by a penalty technique).

## **Additional Inherited Members**

# 7.23.1 Detailed Description

To build element equations for 2-D linearized elasticity using 3-node triangles.

This class enables building finite element arrays for linearized isotropic elasticity problem in 2-D domains using 3-Node triangles.

Unilateral contact is handled using a penalty function. Note that members calculating element arrays have as an argument a real coef that is multiplied by the contribution of the current element. This makes possible testing different algorithms.

## 7.23.2 Constructor & Destructor Documentation

#### Elas2DT3()

Default Constructor.

Constructs an empty equation.

## Elas2DT3 (Mesh & ms)

Constructor using Mesh data.

#### Parameters

in <i>ms</i> Mesh instar
--------------------------

#### Elas2DT3 ( Mesh & ms, Vect< real\_t > & u )

Constructor using Mesh data and solution vector.

#### **Parameters**

in ms		Mesh instance
in,out	и	Reference to solution vector

## 7.23.3 Member Function Documentation

void Media ( real\_t E, real\_t nu, real\_t rho )

Set media properties.

Useful to override material properties deduced from mesh file.

void LMass ( real\_t coef = 1. ) [virtual]

Add element lumped mass contribution to element matrix after multiplication by coef

**Parameters** 

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

Reimplemented from Equa\_Solid < real\_t, 3, 6, 2, 4 >.

void Mass ( real\_t coef = 1. ) [virtual]

Add element consistent mass contribution to element matrix after multiplication by coef

**Parameters** 

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

Reimplemented from Equa\_Solid < real\_t, 3, 6, 2, 4 >.

void Deviator ( real\_t coef = 1. ) [virtual]

Add element deviatoric matrix to element matrix after multiplication by coef

Parameters

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

Reimplemented from Equa\_Solid < real\_t, 3, 6, 2, 4 >.

# void Dilatation ( real\_t coef = 1. ) [virtual]

Add element dilatational contribution to element matrix after multiplication by coef

#### **Parameters**

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

Reimplemented from Equa\_Solid < real\_t, 3, 6, 2, 4 >.

# void BodyRHS ( const Vect< real\_t > & f )

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

#### **Parameters**

	in	f	Vector containing source at nodes (DOF by DOF)
--	----	---	--

# void BoundaryRHS ( const Vect< real\_t > & f )

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

#### Parameters

in	f	Vect instance that contains constant traction to impose to side.
----	---	--

# int Contact ( real\_t coef = 1.e07 )

Penalty Signorini contact side contribution to matrix and right-hand side.

#### **Parameters**

in	coef	Penalty value by which the added term is multiplied [Default: 1.e07]
----	------	--

## Returns

= 0 if no contact is achieved on this side, 1 otherwise

## void Reaction ( Vect< real\_t > & r )

Calculate reactions.

This function can be invoked in postprocessing

#### Parameters

in	r	Reaction on the side
----	---	----------------------

## void ContactPressure ( const Vect< real\_t > & f, real\_t penal, Point< real\_t > & p )

Calculate contact pressure.

This function can be invoked in postprocessing

#### **Parameters**

in	f	
in	penal	Penalty parameter that was used to impose contact condition
out	р	Contact pressure

# void Strain ( Vect< real\_t > & eps )

Calculate strains in element.

This function can be invoked in postprocessing.

#### **Parameters**

out eps	vector of strains in elements
---------	-------------------------------

# void Stress ( Vect< real\_t > & s, Vect< real\_t > & vm )

Calculate principal stresses and Von-Mises stress in element.

#### **Parameters**

out	S	vector of principal stresses in elements	
out	vm	Von-Mises stresses in elements This function can be invoked in postprocessing.	

# void Periodic ( real\_t coef = 1.e20 )

Add contribution of periodic boundary condition (by a penalty technique).

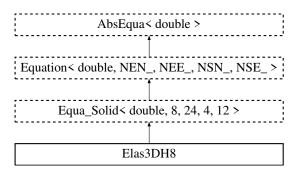
Boundary nodes where periodic boundary conditions are to be imposed must have codes equal to PERIODIC\_A on one side and PERIODIC\_B on the opposite side.

#### Parameters

in	coef	Value of penalty parameter [Default: 1.e20]	
----	------	---	--

# 7.24 Elas3DH8 Class Reference

To build element equations for 3-D linearized elasticity using 8-node hexahedra. Inheritance diagram for Elas3DH8:



#### **Public Member Functions**

• Elas3DH8 ()

Default Constructor.

• Elas3DH8 (Mesh &ms)

Constructor using Mesh instance.

• ~Elas3DH8 ()

Destructor.

• void LMass (real\_t coef=1.)

Add element lumped mass contribution to element matrix after multiplication by coef.

• void Mass (real\_t coef=1.)

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

• void Deviator (real\_t coef=1.)

Add element deviatoric matrix to element matrix after multiplication by coef.

• void Dilatation (real\_t coef=1.)

Add element dilatational contribution to element matrix after multiplication by coef.

• void BoundaryRHS (const Vect< real\_t > &f)

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

void BodyRHS (const Vect< real\_t > &f)

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

# **Additional Inherited Members**

# 7.24.1 Detailed Description

To build element equations for 3-D linearized elasticity using 8-node hexahedra.

This class enables building finite element arrays for linearized isotropic elasticity problem in 3-D domains using 8-Node hexahedra.

Note that members calculating element arrays have as an argument a double coef that is multiplied by the contribution of the current element. This makes possible testing different algorithms.

## 7.24.2 Constructor & Destructor Documentation

#### Elas3DH8()

Default Constructor.

Constructs an empty equation.

## Elas3DH8 (Mesh & ms)

Constructor using Mesh instance.

#### **Parameters**

in	ms	Reference to Mesh instance
----	----	----------------------------

# 7.24.3 Member Function Documentation

# void BoundaryRHS ( const Vect< real\_t > & f )

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

#### **Parameters**

# void BodyRHS ( const Vect< real\_t > & f )

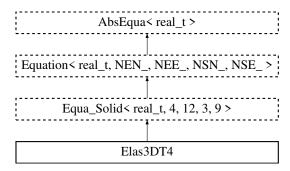
Add body right-hand side term to right hand side.

#### **Parameters**

in	f	Vector containing source at nodes (DOF by DOF).
----	---	---

# 7.25 Elas3DT4 Class Reference

To build element equations for 3-D linearized elasticity using 4-node tetrahedra. Inheritance diagram for Elas3DT4:



# **Public Member Functions**

- Elas3DT4 ()
  - Default Constructor.
- Elas3DT4 (Mesh &ms)

Constructor using a Mesh instance.

• Elas3DT4 (Mesh &ms, Vect< real\_t > &u)

Constructor using a Mesh instance and solution vector.

• ~Elas3DT4 ()

Destructor.

• void Media (real\_t E, real\_t nu, real\_t rho)

Set Media properties.

• void LMass (real\_t coef=1)

Add element lumped mass contribution to element matrix after multiplication by coef.

• void Deviator (real\_t coef=1.)

Add element deviatoric matrix to element matrix after multiplication by coef.

• void Dilatation (real\_t coef=1.)

Add element dilatational contribution to left-hand side after multiplication by coef.

void BodyRHS (const Vect< real\_t > &f)

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

void BoundaryRHS (const Vect< real\_t > &f)

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

# **Additional Inherited Members**

# 7.25.1 Detailed Description

To build element equations for 3-D linearized elasticity using 4-node tetrahedra.

This class enables building finite element arrays for linearized isotropic elasticity problem in 3-D domains using 4-Node tetrahedra.

## 7.25.2 Constructor & Destructor Documentation

Elas3DT4 (Mesh & ms)

Constructor using a Mesh instance.

Parameters

_			
:	in	ms	Reference to Mesh instance

# Elas3DT4 ( Mesh & ms, Vect< real\_t > & u )

Constructor using a Mesh instance and solution vector.

#### Parameters

in	ms	Reference to Mesh instance
in,out	и	Reference to solution vector

#### 7.25.3 Member Function Documentation

void Media ( real\_t E, real\_t nu, real\_t rho )

Set Media properties.

#### **Parameters**

in	Е	Young's modulus
in	пи	Poisson ratio
in	rho	Density

# void BodyRHS ( const Vect< real\_t > & f )

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

## Parameters

in	f	Vect instance containing source at nodes (DOF by DOF).
----	---	--

# void BoundaryRHS ( const Vect< real\_t > & f )

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

#### **Parameters**

# 7.26 Element Class Reference

To store and treat finite element geometric information.

# **Public Member Functions**

• Element ()

Default constructor.

• Element (size\_t label, const string &shape)

Constructor initializing label, shape of element.

• Element (size\_t label, int shape)

Constructor initializing label, shape of element.

• Element (size\_t label, const string &shape, int c)

Constructor initializing label, shape and code of element.

• Element (size\_t label, int shape, int c)

Constructor initializing label, shape and code of element.

• Element (const Element &el)

 $Copy\ constructor.$ 

• ∼Element ()

Destructor.

void setLabel (size\_t i)

Define label of element.

• void setCode (int c)

Define code of element.

• void setCode (const string &exp, int code)

Define code by a boolean algebraic expression invoking coordinates of element nodes.

• void Add (Node \*node)

Insert a node at end of list of nodes of element.

• void Add (Node \*node, int n)

Insert a node and set its local node number.

• void Replace (size\_t label, Node \*node)

Replace a node at a given local label.

• void Replace (size\_t label, Side \*side)

Replace a side at a given local label.

void Add (Side \*sd)

Assign Side to Element.

void Add (Side \*sd, int k)

Assign Side to Element with assigned local label.

void Add (Element \*el)

Add a neighbor element.

• void set (Element \*el, int n)

Add a neighbor element and set its label.

• void setDOF (size\_t i, size\_t dof)

Define label of DOF.

• void setCode (size\_t dof, int code)

Assign code to a DOF.

• void setNode (size\_t i, Node \*node)

Assign a node given by its pointer as the i-th node of element.

void setNbDOF (size\_t i)

Set number of degrees of freedom of element.

• void setFirstDOF (size\_t i)

 $Set\ label\ of\ first\ DOF\ in\ element.$ 

• int getShape () const

Return element shape.

size\_t getLabel () const

Return label of element.

• size\_t n () const

Return label of element.

• int getCode () const

Return code of element.

• size\_t getNbNodes () const

Return number of element nodes.

• size\_t getNbVertices () const

Return number of element vertices.

• size\_t getNbSides () const

Return number of element sides (Constant version)

• size\_t getNbEq () const

Return number of element equations.

• size\_t getNbDOF () const

return element nb of DOF

• size\_t getDOF (size\_t i=1) const

Return element DOF label.

• size\_t getFirstDOF () const

Return element first DOF label.

• size\_t getNodeLabel (size\_t n) const

Return global label of node of local label i.

size\_t getSideLabel (size\_t n) const

Return global label of side of local label i.

• Node \* getPtrNode (size\_t i) const

Return pointer to node of label i (Local labelling).

Node \* operator() (size\_t i) const

Operator ().

Side \* getPtrSide (size\_t i) const

*Return pointer to side of label i (Local labelling).* 

• int Contains (const Node \*nd) const

Say if element contains given node.

• int Contains (const Node &nd) const

Say if element contains given node.

• int Contains (const Side \*sd) const

Say if element contains given side.

• int Contains (const Side &sd) const

Say if element contains given side.

Element \* getNeighborElement (size\_t i) const

Return pointer to element Neighboring element.

• size\_t getNbNeigElements () const

Return number of neighboring elements.

• real\_t getMeasure () const

Return measure of element.

• Point< real\_t > getUnitNormal (size\_t i) const

Return outward unit normal to i-th side of element.

• bool isOnBoundary () const

Say if current element is a boundary element or not.

• Node \* operator() (size\_t i)

Operator ().

• int setSide (size\_t n, size\_t \*nd)

Initialize information on element sides.

• bool isActive () const

Return true or false whether element is active or not.

• int getLevel () const

Return element level *Element* level decreases when element is refined (starting from 0). If the level is 0, then the element has no father.

void setChild (Element \*el)

Assign element as child of current one and assign current element as father This function is principally used when refining is invoked (e.g. for mesh adaption)

• Element \* getChild (size\_t i) const

Return pointer to i-th child element Return null pointer is no childs.

size\_t getNbChilds () const

Return number of children of element.

• Element \* getParent () const

Return pointer to parent element Return null if no parent.

• size\_t IsIn (const Node \*nd)

Check if a given node belongs to current element.

# 7.26.1 Detailed Description

To store and treat finite element geometric information.

Class Element enables defining an element of a finite element mesh. The element is given in particular by its shape and a list of nodes. Each node can be accessed by the member function getPtrNode. Moreover, class Mesh can generate for each element its list of sides. The string that defines the element shape must be chosen according to the following list:

#### Remarks

Once a Mesh instance is constructed, one has access for each Element of the mesh to pointers to element sides provided the member function getAllSides of Mesh has been invoked. With this, an element can be tested to see if it is on the boundary, i.e. if it has at least one side on the boundary

#### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

#### 7.26.2 Constructor & Destructor Documentation

Element ( size\_t label, const string & shape )

Constructor initializing label, shape of element.

#### **Parameters**

in	label	Label to assign to element.
in	shape	Shape of element (See class description).

## Element ( size\_t label, int shape )

Constructor initializing label, shape of element.

## Parameters

in	label	Label to assign to element.
in	shape	Shape of element (See enum ElementShape in Mesh)

# Element ( size\_t label, const string & shape, int c )

Constructor initializing label, shape and code of element.

## Parameters

in	label	Label to assign to element.	
in	shape	Shape of element (See class description).	
in	С	Code to assign to element (useful for media properties).	

# Element ( size\_t label, int shape, int c )

Constructor initializing label, shape and code of element.

#### **Parameters**

in	label	Label to assign to element.
in	shape	Shape of element (See enum ElementShape in Mesh).
in	С	Code to assign to element (useful for media properties).

# 7.26.3 Member Function Documentation

# void setLabel ( size\_t i )

Define label of element.

## Parameters

		T 1 1,
in	1	Label to assign to element

# void setCode ( int c )

Define code of element.

#### **Parameters**

in	С	Code to assign to element.
----	---	----------------------------

# void setCode ( const string & exp, int code )

Define code by a boolean algebraic expression invoking coordinates of element nodes.

## Parameters

in	ехр	Boolean algebraic expression as required by fparser
in	code	Code to assign to node if the algebraic expression is true

## void Add ( Node \* node )

Insert a node at end of list of nodes of element.

#### **Parameters**

in <i>node</i> Pointer to Node instar	nce.
---------------------------------------	------

# void Add ( Node \* node, int n )

Insert a node and set its local node number.

## Parameters

	node	[in] Pointer to Node instance
in	n	Element node number to assign

# void Replace ( size\_t label, Node \* node )

Replace a node at a given local label.

#### **Parameters**

in	label	Node to replace.
in	node	Pointer to Node instance to copy to current instance.

# void Replace ( size\_t label, Side \* side )

Replace a side at a given local label.

#### Parameters

in	label	Side to replace.	
in	side	Pointer to Side instance to copy to current instance.	

# void Add ( Side \* sd )

Assign Side to Element.

#### Parameters

in	sd	Pointer to Side instance.
----	----	---------------------------

# void Add ( Side \*sd, int k )

Assign Side to Element with assigned local label.

## Parameters

in	sd	Pointer to Side instance.
in	k	Local label.

# void Add ( Element \* el )

Add a neighbor element.

# Parameters

in el Poi	nter to Element instance
-----------	--------------------------

# void set ( Element \*el, int n )

Add a neighbor element and set its label.

## Parameters

in	el	Pointer to Element instance
in	n	Neighbor element number to assign

# void setDOF ( size\_t i, size\_t dof )

Define label of DOF.

## Parameters

in	i	Index of DOF.
in	dof	Label of DOF to assign.

# void setCode ( size\_t dof, int code )

Assign code to a DOF.

# Parameters

in	dof	Index of dof for assignment.
in	code	Code to assign.

# Node\* operator() ( size\_t i ) const

Operator ().

Return pointer to node of local label i.

#### int Contains (const Node \* nd) const

Say if element contains given node.

This function tests if the element contains a node with the same pointer at the sought one

#### Parameters

in	nd	Pointer to Node instance

#### Returns

Local node label in element. If 0, the element does not contain this node

#### int Contains (const Node & nd) const

Say if element contains given node.

This function tests if the element contains a node with the same label at the sought one

#### **Parameters**

	in	nd	Reference to Node instance	1
--	----	----	----------------------------	---

#### Returns

Local node label in element. If 0, the element does not contain this node

## int Contains (const Side \* sd) const

Say if element contains given side.

This function tests if the element contains a side with the same pointer at the sought one

## Parameters

in	sd	Pointer to Side instance

## Returns

Local side label in element. If 0, the element does not contain this side

## int Contains (const Side & sd) const

Say if element contains given side.

This function tests if the element contains a side with the same label at the sought one

## Parameters

in	sd	Reference to Side instance
----	----	----------------------------

#### Returns

Local side label in element. If 0, the element does not contain this side

## Element\* getNeighborElement ( size\_t i ) const

Return pointer to element Neighboring element.

#### **Parameters**

in i Index of element to look fe
----------------------------------

#### Note

This method returns valid information only if the Mesh member function Mesh::getElement ← NeighborElements() has been called before.

## size\_t getNbNeigElements ( ) const

Return number of neigboring elements.

Note

This method returns valid information only if the Mesh member function Mesh::getElement ← NeighborElements() has been called before.

## real\_t getMeasure ( ) const

Return measure of element.

This member function returns length, area or volume of element. In case of quadrilaterals and hexahedrals it returns determinant of Jacobian of mapping between reference and actual element

## Point<real\_t> getUnitNormal ( size\_t i ) const

Return outward unit normal to i-th side of element. Sides are ordered [node\_1,node\_2], [node\_2,node\_3], ...

# bool isOnBoundary ( ) const

Say if current element is a boundary element or not.

Note

this information is available only if boundary elements were determined i.e. if member function Mesh::getBoundarySides or Mesh::getAllSides has been invoked before.

## Node\* operator() ( size\_t i )

Operator ().

Return pointer to node of local label i.

# int setSide ( size\_t n, size\_t \* nd )

Initialize information on element sides.

This function is to be used to initialize loops over sides.

#### **Parameters**

in	n	Label of side.
in	nd	Array of pointers to nodes of the side (nd[0], nd[1], point to first, second nodes,

## void setChild ( Element \* el )

Assign element as child of current one and assign current element as father This function is principally used when refining is invoked (e.g. for mesh adaption)

#### **Parameters**

in		Pointer to element to assign	
----	--	------------------------------	--

#### size\_t IsIn ( const Node \* nd )

Check if a given node belongs to current element.

## Parameters

in <i>nd</i> Pointer to n	ode to locate
---------------------------	---------------

## Returns

local label of node if this one is found, 0 otherwise

# 7.27 ElementList Class Reference

Class to construct a list of elements having some common properties.

## **Public Member Functions**

• ElementList (Mesh &ms)

Constructor using a Mesh instance.

• ~ElementList ()

Destructor.

• void selectCode (int code)

Select elements having a given code.

• void unselectCode (int code)

Unselect elements having a given code.

• void selectLevel (int level)

Select elements having a given level.

• size\_t getNbElements () const

Return number of selected elements.

• void top ()

Reset list of elements at its top position (Non constant version)

• void top () const

Reset list of elements at its top position (Constant version)

• Element \* get ()

Return pointer to current element and move to next one (Non constant version)

• Element \* get () const

Return pointer to current element and move to next one (Constant version)

# 7.27.1 Detailed Description

Class to construct a list of elements having some common properties.

This class enables choosing multiple selection criteria by using function select... However, the intersection of these properties must be empty.

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

#### 7.27.2 Member Function Documentation

void unselectCode ( int code )

Unselect elements having a given code.

**Parameters** 

in   code   Code of elements to exclu
---------------------------------------

## void selectLevel ( int level )

Select elements having a given level.

Parameters

in	level	Level of elements to select

Elements having a given level (for mesh adaption) are selected in a list

# 7.28 Ellipse Class Reference

To store and treat an ellipsoidal figure. Inheritance diagram for Ellipse:



# **Public Member Functions**

• Ellipse ()

Default constructor.

• Ellipse (Point< real\_t > c, real\_t a, real\_t b, int code=1)

Constructor with given ellipse data.

• real\_t getSignedDistance (const Point< real\_t > &p) const

Return signed distance of a given point from the current ellipse.

• Ellipse & operator+= (Point< real\_t > a)

Operator +=

• Ellipse & operator+= (real\_t a)

Operator \*=

## **Additional Inherited Members**

# 7.28.1 Detailed Description

To store and treat an ellipsoidal figure.

## 7.28.2 Constructor & Destructor Documentation

## Ellipse ( )

Default constructor.

Constructs an ellipse with semimajor axis = 1, and semiminor axis = 1

## Ellipse ( Point< real\_t > c, real\_t a, real\_t b, int code = 1 )

Constructor with given ellipse data.

#### Parameters

in	С	Coordinates of center
in	а	Semimajor axis
in	b	Semiminor axis
in	code	Code to assign to the generated figure [Default: 1]

# 7.28.3 Member Function Documentation

## real\_t getSignedDistance ( const Point < real\_t > & p ) const [virtual]

Return signed distance of a given point from the current ellipse.

The computed distance is negative if p lies in the ellipse, positive if it is outside, and 0 on its boundary

#### **Parameters**

in	р	Point <double> instance</double>
----	---	----------------------------------

Reimplemented from Figure.

# Ellipse& operator+= ( Point< real\_t > a )

Operator +=

Translate ellipse by a vector a

#### Parameters

in	а	Vector defining the translation
----	---	---------------------------------

## Ellipse& operator+= ( real\_t a )

Operator \*=

Scale ellipse by a factor a

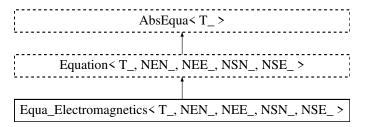
#### **Parameters**

in	а	Scaling value
----	---	---------------

# 7.29 Equa\_Electromagnetics< T\_, NEN\_, NEE\_, NSN\_, NSE\_ > Class Template Reference

Abstract class for Electromagnetics **Equation** classes.

Inheritance diagram for Equa\_Electromagnetics < T\_, NEN\_, NEE\_, NSN\_, NSE\_ >:



# **Protected Member Functions**

- void MagneticPermeability (const real\_t &mu)
  - Set (constant) magnetic permeability.
- void MagneticPermeability (const string &exp)
  - Set magnetic permeability given by an algebraic expression.
- void ElectricConductivity (const real\_t &sigma)
  - Set (constant) electric conductivity.
- void ElectricConductivity (const string &exp)
  - set electric conductivity given by an algebraic expression
- void ElectricResistivity (const real\_t &rho)
  - Set (constant) electric resistivity.
- void ElectricResistivity (const string &exp)
  - Set electric resistivity given by an algebraic expression.
- void setMaterial ()
  - Set material properties.

# **Additional Inherited Members**

# 7.29.1 Detailed Description

template<class T\_, size\_t NEN\_, size\_t NEE\_, size\_t NSN\_, size\_t NSE\_> class OFELI::Equa\_Electromagnetics< T\_, NEN\_, NEE\_, NSN\_, NSE\_>

Abstract class for Electromagnetics Equation classes.

**Template Parameters** 

<t_></t_>	data type (double, float,)
<nen></nen>	Number of element nodes
<nee←< th=""><th>Number of element equations</th></nee←<>	Number of element equations
_>	
<nsn↔< th=""><th>Number of side nodes</th></nsn↔<>	Number of side nodes
_>	
< <i>NSE</i> _>	Number of side equations

Author

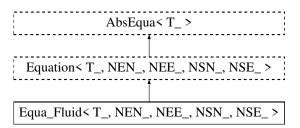
Rachid Touzani

Copyright

GNU Lesser Public License

# 7.30 Equa\_Fluid< T\_, NEN\_, NEE\_, NSN\_, NSE\_ > Class Template Reference

Abstract class for Fluid Dynamics Equation classes. Inheritance diagram for Equa\_Fluid < T\_-, NEN\_-, NEE\_-, NSN\_-, NSE\_- >:



# **Public Member Functions**

• Equa\_Fluid ()

Default constructor.

• virtual ~Equa\_Fluid ()

Destructor.

• void Reynolds (const real\_t &Re)

Set Reynolds number.

• void Viscosity (const real\_t &visc)

Set (constant) Viscosity.

• void Viscosity (const string &exp)

Set viscosity given by an algebraic expression.

• void Density (const real\_t &dens)

Set (constant) Viscosity.

• void Density (const string &exp)

Set Density given by an algebraic expression.

• void ThermalExpansion (const real\_t \*e)

Set (constant) thermal expansion coefficient.

void ThermalExpansion (const string &exp)

Set thermal expansion coefficient given by an algebraic expression.

void setMaterial ()

Set material properties.

# 7.30.1 Detailed Description

template<class T\_, size\_t NEN\_, size\_t NEE\_, size\_t NSN\_, size\_t NSE\_> class OFELI::Equa\_Fluid< T\_, NEN\_, NEE\_, NSN\_, NSE\_>

Abstract class for Fluid Dynamics Equation classes.

## **Template Parameters**

<t_></t_>	data type (double, float,)
<nen></nen>	Number of element nodes
<nee↔< th=""><th>Number of element equations</th></nee↔<>	Number of element equations
_>	
<nsn←< th=""><th>Number of side nodes</th></nsn←<>	Number of side nodes
_>	
< <i>NSE_&gt;</i>	Number of side equations

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

# 7.30.2 Constructor & Destructor Documentation

Equa\_Fluid ( )

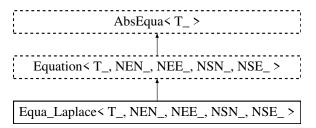
Default constructor.

Constructs an empty equation.

# 7.31 Equa\_Laplace< T\_, NEN\_, NEE\_, NSN\_, NSE\_ > Class Template Reference

Abstract class for classes about the Laplace equation.

Inheritance diagram for Equa\_Laplace < T\_, NEN\_, NEE\_, NSN\_, NSE\_ >:



# **Public Member Functions**

• Equa\_Laplace ()

Default constructor.

• virtual ~Equa\_Laplace ()

Destructor.

• virtual void LHS ()

Add finite element matrix to left-hand side.

• virtual void BodyRHS (const Vect< real\_t > &f)

Add body source term to right-hand side.

• virtual void BoundaryRHS (const Vect< real\_t > &h)

Add boundary source term to right-hand side.

• void build ()

Build global matrix and right-hand side.

• virtual void buildEigen (int opt=0)

Build matrices for an eigenvalue problem.

• void build (EigenProblemSolver &e)

Build the linear system for an eigenvalue problem.

# 7.31.1 Detailed Description

template<class T\_, size\_t NEN\_, size\_t NEE\_, size\_t NSN\_, size\_t NSE\_> class OFELI::Equa\_Laplace< T\_, NEN\_, NEE\_, NSN\_, NSE\_>

Abstract class for classes about the Laplace equation.

**Template Parameters** 

$T_{-}$	Data type (real_t, float, complex <real_t>,)</real_t>
NE↔	Number of element nodes
$N_{-}$	
NE⊷	Number of element equations
$E_{-}$	-
NS⊷	Number of side nodes
$N_{-}$	
NS⊷	Number of side equations
$E_{-}$	-

## 7.31. EQUA\_LAPLACE< T\_, NEN\_, NEE\_, NSN\_, NSEHAPTASS.TEMASSADEREHIRENIATION

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

## 7.31.2 Constructor & Destructor Documentation

## Equa\_Laplace ( )

Default constructor.

Constructs an empty equation.

## 7.31.3 Member Function Documentation

virtual void BodyRHS ( const Vect< real.t> & f ) [virtual]

Add body source term to right-hand side.

**Parameters** 

in	f	Vector containing the source given function at mesh nodes	
----	---	---	--

Reimplemented in Laplace2DT3, Laplace1DL2, Laplace1DL3, and Laplace2DT6.

## virtual void BoundaryRHS ( const Vect< real\_t > & h ) [virtual]

Add boundary source term to right-hand side.

Parameters

in	h	Vector containing the source given function at mesh nodes

Reimplemented in Laplace2DT3, Laplace1DL2, Laplace1DL3, and Laplace2DT6.

# void build ( )

Build global matrix and right-hand side.

The problem matrix and right-hand side are the ones used in the constructor. They are updated in this member function.

## void build ( EigenProblemSolver & e )

Build the linear system for an eigenvalue problem.

**Parameters** 

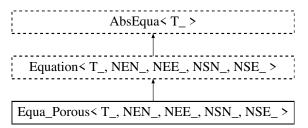
i	n	e	Reference to used EigenProblemSolver instance
---	---	---	---

OFELI's Reference Guide

# 7.32 Equa\_Porous<br/>< T\_, NEN\_, NEE\_, NSN\_, NSE\_ > Class Template Reference

Abstract class for Porous Media Finite Element classes.

Inheritance diagram for Equa\_Porous< T\_, NEN\_, NEE\_, NSN\_, NSE\_>:



## **Public Member Functions**

• Equa\_Porous ()

Default constructor.

• virtual ~Equa\_Porous ()

Destructor.

• virtual void Mobility ()

Add mobility term to the 0-th order element matrix.

• virtual void Mass ()

Add porosity term to the 1-st order element matrix.

virtual void BodyRHS (const Vect< real\_t > &bf)

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

virtual void BoundaryRHS (const Vect< real\_t > &sf)

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

• void build ()

Build the linear system of equations.

void build (TimeStepping &s)

Build the linear system of equations.

• void build (EigenProblemSolver &e)

Build the linear system for an eigenvalue problem.

• int run ()

Run the equation.

• void Mu (const string &exp)

Set viscosity given by an algebraic expression.

#### **Protected Member Functions**

• void setMaterial ()

Set material properties.

# 7.32.1 Detailed Description

template<class T\_, size\_t NEN\_, size\_t NEE\_, size\_t NSN\_, size\_t NSE\_> class OFELI::Equa\_Porous< T\_, NEN\_, NEE\_, NSN\_, NSE\_>

Abstract class for Porous Media Finite Element classes.

#### **Template Parameters**

<t_></t_>	data type (real_t, float,)	
<nen></nen>	Number of element nodes	
<nee↔< th=""><th>Number of element equations</th></nee↔<>	Number of element equations	
_>		
<nsn←< th=""><th>Number of side nodes</th></nsn←<>	Number of side nodes	
_>		
< <i>NSE_&gt;</i>	Number of side equations	

## 7.32.2 Constructor & Destructor Documentation

#### Equa\_Porous ( )

Default constructor.

Constructs an empty equation.

#### 7.32.3 Member Function Documentation

virtual void BodyRHS ( const Vect< real\_t> & bf ) [virtual]

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

#### **Parameters**

	in bf	Vector containing source at nodes.
--	-------	------------------------------------

Reimplemented in WaterPorous2D.

## virtual void BoundaryRHS ( const Vect< real\_t > & sf ) [virtual]

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

#### Parameters

i	n	sf	Vector containing source at nodes.	]
---	---	----	------------------------------------	---

Reimplemented in WaterPorous2D.

## void build ( )

Build the linear system of equations.

Before using this function, one must have properly selected appropriate options for:

- The choice of a steady state or transient analysis. By default, the analysis is stationary
- In the case of transient analysis, the choice of a time integration scheme and a lumped or consistent capacity matrix. If transient analysis is chosen, the lumped capacity matrix option is chosen by default, and the implicit Euler scheme is used by default for time integration.

## void build ( TimeStepping & s )

Build the linear system of equations.

Before using this function, one must have properly selected appropriate options for:

- The choice of a steady state or transient analysis. By default, the analysis is stationary
- In the case of transient analysis, the choice of a time integration scheme. If transient analysis is chosen, the implicit Euler scheme is used by default for time integration.

#### **Parameters**

	in	S	Reference to used TimeStepping instance	
--	----	---	---	--

# void build ( EigenProblemSolver & e )

Build the linear system for an eigenvalue problem.

#### **Parameters**

in	e	Reference to used EigenProblemSolver instance
----	---	---

#### int run ( )

Run the equation.

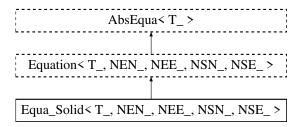
If the analysis (see function setAnalysis) is STEADY\_STATE, then the function solves the stationary equation.

If the analysis is TRANSIENT, then the function performs time stepping until the final time is reached.

# 7.33 Equa\_Solid< T\_, NEN\_, NEE\_, NSN\_, NSE\_ > Class Template Reference

Abstract class for Solid Mechanics Finite Element classes.

Inheritance diagram for Equa\_Solid < T\_, NEN\_, NEE\_, NSN\_, NSE\_ >:



## **Public Member Functions**

• Equa\_Solid ()

Default constructor.

• virtual ~Equa\_Solid ()

Destructor.

virtual void LMass (real\_t coef=1)

Add lumped mass contribution to left-hand side.

virtual void Mass (real\_t coef=1)

Add consistent mass contribution to left-hand side.

• virtual void Deviator (real\_t coef=1)

Add deviator matrix to left-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].

• virtual void Dilatation (real\_t coef=1)

Add dilatation matrix to left-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].

• virtual void Stiffness (real\_t coef=1)

Add stiffness matrix to left-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].

• void setInput (EqDataType opt, Vect< real\_t > &u)

Set specific input data to solid mechanics.

# **Protected Member Functions**

• void Young (const real\_t &E)

Set (constant) Young modulus.

• void Poisson (const real\_t &nu)

Set (constant) Poisson ratio.

• void Density (const real\_t &rho)

Set (constant) density.

• void Young (const string &exp)

Set Young modulus given by an algebraic expression.

• void Poisson (const string &exp)

Set Poisson ratio given by an algebraic expression.

• void Density (const string &exp)

Set density given by an algebraic expression.

void setMaterial ()

Set material properties.

## 7.33.1 Detailed Description

template<class T\_, size\_t NEN\_, size\_t NEE\_, size\_t NSN\_, size\_t NSE\_> class OFELI::Equa\_Solid< T\_, NEN\_, NEE\_, NSN\_, NSE\_>

Abstract class for Solid Mechanics Finite Element classes.

# Template Parameters

<t_></t_>	data type (double, float,)	
<nen></nen>	Number of element nodes	
<nee←< th=""><th>Number of element equations</th></nee←<>	Number of element equations	
_>		
<nsn↔< th=""><th>Number of side nodes</th></nsn↔<>	Number of side nodes	
_>		
< <u>NSE</u> ->	Number of side equations	

# 7.33.2 Constructor & Destructor Documentation

## Equa\_Solid ( )

Default constructor.

Constructs an empty equation.

## 7.33.3 Member Function Documentation

virtual void LMass ( real\_t coef = 1 ) [virtual]

Add lumped mass contribution to left-hand side.

#### **Parameters**

i	coef	coefficient to multiply by the matrix before adding [Default: 1]
---	------	--

Reimplemented in Beam3DL2, Elas2DT3, Elas2DQ4, Elas3DT4, Bar2DL2, and Elas3DH8.

# virtual void Mass ( real\_t coef = 1 ) [virtual]

Add consistent mass contribution to left-hand side.

#### **Parameters**

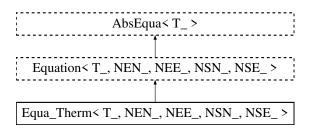
in	coef	coefficient to multiply by the matrix before adding [Default: 1]
----	------	--

Reimplemented in Beam3DL2, Elas2DT3, Elas2DQ4, Bar2DL2, and Elas3DH8.

# 7.34 Equa\_Therm< T\_, NEN\_, NEE\_, NSN\_, NSE\_ > Class Template Reference

Abstract class for Heat transfer Finite Element classes.

Inheritance diagram for Equa\_Therm< T\_, NEN\_, NEE\_, NSN\_, NSE\_>:



## **Public Member Functions**

• Equa\_Therm ()

Default constructor.

• virtual ~Equa\_Therm ()

Destructor.

• virtual void setStab ()

Set stabilized formulation.

• virtual void LCapacity (real\_t coef=1)

Add lumped capacity contribution to element matrix.

virtual void Capacity (real\_t coef=1)

Add consistent capacity contribution to left-hand side.

• virtual void Diffusion (real\_t coef=1.)

Add diffusion term to element matrix.

• virtual void Convection (real\_t coef=1.)

Add convection term to element matrix.

virtual void BodyRHS (const Vect< real\_t > &f)

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

• virtual void BoundaryRHS (const Vect< real\_t > &f)

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

• void build ()

Build the linear system of equations.

void build (TimeStepping &s)

Build the linear system of equations.

• void build (EigenProblemSolver &e)

Build the linear system for an eigenvalue problem.

void setRhoCp (const real\_t &rhocp)

Set product of Density by Specific heat (constants)

void setConductivity (const real\_t &diff)

Set (constant) thermal conductivity.

• void RhoCp (const string &exp)

Set product of Density by Specific heat given by an algebraic expression.

• void Conduc (const string &exp)

Set thermal conductivity given by an algebraic expression.

## **Protected Member Functions**

void setMaterial ()

Set material properties.

# 7.34.1 Detailed Description

template<class T\_, size\_t NEN\_, size\_t NEE\_, size\_t NSN\_, size\_t NSE\_> class OFELI::Equa\_Therm< T\_, NEN\_, NEE\_, NSN\_, NSE\_>

Abstract class for Heat transfer Finite Element classes.

**Template Parameters** 

<t_></t_>	data type (real_t, float,)	
<nen></nen>	Number of element nodes	
<nee↔< th=""><th>Number of element equations</th></nee↔<>	Number of element equations	
_>		
<nsn↔< th=""><th>Number of side nodes</th></nsn↔<>	Number of side nodes	
_>		
< <i>NSE</i> _>	Number of side equations	

OFELI's Reference Guide

# 7.34.2 Constructor & Destructor Documentation

#### Equa\_Therm ( )

Default constructor.

Constructs an empty equation.

# 7.34.3 Member Function Documentation

virtual void setStab ( ) [virtual]

Set stabilized formulation.

Stabilized variational formulations are to be used when the Pclet number is large. By default, no stabilization is used.

## virtual void LCapacity ( real\_t coef = 1 ) [virtual]

Add lumped capacity contribution to element matrix.

#### **Parameters**

in	coef	coefficient to multiply by the matrix before adding [Default: 1]
----	------	--

Reimplemented in DC2DT3, DC1DL2, DC3DT4, DC3DAT3, and DC2DT6.

# virtual void Capacity ( real\_t coef = 1 ) [virtual]

Add consistent capacity contribution to left-hand side.

#### Parameters

in	coef	coefficient to multiply by the matrix before adding [Default: 1]
----	------	--

Reimplemented in DC2DT3, DC1DL2, DC3DT4, DC3DAT3, and DC2DT6.

## virtual void BodyRHS ( const Vect< real.t > & f ) [virtual]

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

## Parameters

in I Vactor containing a	uuuaa at madaa
in   f   Vector containing so	ource at nodes.
The state of the s	our co dit riconos.

Reimplemented in DC2DT3, DC3DT4, DC1DL2, DC2DT6, and DC3DAT3.

# $virtual\ void\ BoundaryRHS\ (\ const\ Vect < real.t > \&f\ )\ [virtual]$

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

# Parameters

in	f	Vector containing source at nodes.

Reimplemented in DC2DT3, DC3DT4, DC2DT6, and DC3DAT3.

#### void build ( )

Build the linear system of equations.

Before using this function, one must have properly selected appropriate options for:

- The choice of a steady state or transient analysis. By default, the analysis is stationary
- In the case of transient analysis, the choice of a time integration scheme and a lumped or consistent capacity matrix. If transient analysis is chosen, the lumped capacity matrix option is chosen by default, and the implicit Euler scheme is used by default for time integration.

# void build ( TimeStepping & s )

Build the linear system of equations.

Before using this function, one must have properly selected appropriate options for:

- The choice of a steady state or transient analysis. By default, the analysis is stationary
- In the case of transient analysis, the choice of a time integration scheme and a lumped or consistent capacity matrix. If transient analysis is chosen, the lumped capacity matrix option is chosen by default, and the implicit Euler scheme is used by default for time integration.

#### **Parameters**

in	S	Reference to used TimeStepping instance
----	---	---

## void build ( EigenProblemSolver & e )

Build the linear system for an eigenvalue problem.

#### **Parameters**

in e Reference to used EigenProblemSolver ins	tance
---	-------

# 7.35 Equation < T\_, NEN\_, NEE\_, NSN\_, NSE\_ > Class Template Reference

Abstract class for all equation classes.

Inheritance diagram for Equation < T\_, NEN\_, NEE\_, NSN\_, NSE\_ >:



## **Public Member Functions**

- Equation ()
- Equation (Mesh &mesh)

Constructor with mesh instance.

• Equation (Mesh &mesh, Vect< T\_> &u)

Constructor with mesh instance and solution vector.

Equation (Mesh &mesh, Vect< T<sub>-</sub> > &u, real\_t &init\_time, real\_t &final\_time, real\_t &time←
 \_step)

Constructor with mesh instance, matrix and right-hand side.

• virtual ~ Equation ()

Destructor.

• void updateBC (const Element &el, const Vect< T\_> &bc)

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

• void DiagBC (int dof\_type=NODE\_DOF, int dof=0)

*Update element matrix to impose bc by diagonalization technique.* 

• void LocalNodeVector (Vect< T<sub>-</sub> > &b)

Localize Element Vector from a Vect instance.

• void ElementNodeVector (const Vect< T\_> &b, LocalVect< T\_, NEE\_> &be)

Localize Element Vector from a Vect instance.

void SideNodeVector (const Vect< T<sub>-</sub>> &b, LocalVect< T<sub>-</sub>, NSE<sub>-</sub>> &bs)

Localize Side Vector from a Vect instance.

• void ElementNodeVectorSingleDOF (const Vect< T\_> &b, LocalVect< T\_, NEN\_> &be)

Localize Element Vector from a Vect instance.

• void ElementNodeVector (const Vect< T\_> &b, LocalVect< T\_, NEN\_> &be, int dof)

Localize Element Vector from a Vect instance.

void ElementSideVector (const Vect< T<sub>-</sub>> &b, LocalVect< T<sub>-</sub>, NSE<sub>-</sub>> &be)

Localize Element Vector from a Vect instance.

• void ElementVector (const Vect< T\_> &b, int dof\_type=NODE\_FIELD, int flag=0)

Localize *Element* Vector.

• void SideVector (const Vect< T\_> &b)

Localize Side Vector.

void ElementNodeCoordinates ()

Localize coordinates of element nodes.

• void SideNodeCoordinates ()

Localize coordinates of side nodes.

• void ElementAssembly (Matrix< T\_> \*A)

Assemble element matrix into global one.

void ElementAssembly (BMatrix< T<sub>-</sub> > &A)

Assemble element matrix into global one.

void ElementAssembly (SkSMatrix< T<sub>-</sub> > &A)

Assemble element matrix into global one.

void ElementAssembly (SkMatrix< T<sub>-</sub>> &A)

Assemble element matrix into global one.

void ElementAssembly (SpMatrix< T<sub>-</sub> > &A)

Assemble element matrix into global one.

• void ElementAssembly (TrMatrix< T<sub>-</sub> > &A)

Assemble element matrix into global one.

• void DGElementAssembly (Matrix < T\_ > \*A)

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

void DGElementAssembly (SkSMatrix< T<sub>-</sub>> &A)

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

void DGElementAssembly (SkMatrix < T<sub>-</sub> > &A)

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

• void DGElementAssembly (SpMatrix< T\_> &A)

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

void DGElementAssembly (TrMatrix< T<sub>-</sub> > &A)

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

void SideAssembly (Matrix< T<sub>-</sub>>\*A)

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

void SideAssembly (SkSMatrix < T<sub>-</sub> > &A)

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

void SideAssembly (SkMatrix< T<sub>-</sub> > &A)

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

• void SideAssembly (SpMatrix < T\_ > &A)

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

• void ElementAssembly (Vect< T\_> &v)

Assemble element vector into global one.

• void SideAssembly (Vect< T\_> &v)

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

• void AxbAssembly (const Element &el, const Vect< T\_> &x, Vect< T\_> &b)

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

void AxbAssembly (const Side &sd, const Vect< T<sub>-</sub> > &x, Vect< T<sub>-</sub> > &b)

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

• size\_t getNbNodes () const

Return number of element nodes.

size\_t getNbEq () const

Return number of element equations.

real\_t setMaterialProperty (const string &exp, const string &prop)

Define a material property by an algebraic expression.

#### 7.35.1 Detailed Description

template<class T\_, size\_t NEN\_, size\_t NEE\_, size\_t NSN\_, size\_t NSE\_> class OFELI::Equation< T\_, NEN\_, NEE\_, NSN\_, NSE\_>

Abstract class for all equation classes.

**Template Arguments:** 

- T<sub>-</sub>: data type (real\_t, float, ...)
- NEN\_: Number of element nodes
- NEE\_: Number of element equations
- NSN\_: Number of side nodes
- NSN\_: Number of side equations

Author

Rachid Touzani

Copyright

GNU Lesser Public License

#### 7.35.2 Constructor & Destructor Documentation

### Equation ( )

Default constructor. Constructs an "empty" equation

# Equation ( Mesh & mesh )

Constructor with mesh instance.

#### Parameters

ſ	in	mesh	Mesh instance
---	----	------	---------------

# Equation ( Mesh & mesh, Vect $< T_- > & u$ )

Constructor with mesh instance and solution vector.

#### **Parameters**

in	mesh	Mesh instance
in	и	Vect instance containing solution.

# Equation ( Mesh & mesh, Vect< $T_-$ > & u, real\_t & init\_time, real\_t & final\_time, real\_t & time\_step )

Constructor with mesh instance, matrix and right-hand side.

#### Parameters

in	mesh	Mesh instance
in	и	Vect instance containing Right-hand side.
in	init_time	Initial Time value
in	final_time	Final Time value
in	time_step	Time step value

#### 7.35.3 Member Function Documentation

void updateBC (const Element & el, const Vect<  $T_- > \& bc$ )

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

# 7.35. EQUATION $< T_-$ , NEN\_, NEE\_, NSN\_, NSE\_ > CLEASE THE MP LOAD ASSET DEPOCH ENABLEM TATION

#### Parameters

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

# void DiagBC ( int dof\_type = NODE\_DOF, int dof = 0 )

Update element matrix to impose bc by diagonalization technique.

#### Parameters

in	dof_type	DOF type option. To choose among the enumerated values:	
		NODE_FIELD, DOFs are supported by nodes [Default]	
		ELEMENT_FIELD, DOFs are supported by elements	
		SIDE_FIELD, DOFs are supported by sides	
in	dof	DOF setting:	
		• = 0, All DOFs are taken into account [Default]	
		• != 0, Only DOF No. dof is handled in the system	

### void LocalNodeVector ( Vect< T $_->$ & b )

Localize Element Vector from a Vect instance.

#### Parameters

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

# void ElementNodeVector ( const Vect< $T_-$ > & b, LocalVect< $T_-$ , NEE\_- > & be)

Localize Element Vector from a Vect instance.

#### Parameters

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

# Remarks

All degrees of freedom are transferred to the local vector

#### void SideNodeVector ( const Vect< T $_->$ & b, LocalVect< T $_-$ , NSE $_->$ & bs)

Localize Side Vector from a Vect instance.

#### Parameters

in	b	Global vector to be localized.	
out	bs	Local vector, the length of which is the total number of side equations.	

#### Remarks

All degrees of freedom are transferred to the local vector

# void ElementNodeVectorSingleDOF ( const Vect< T $_->$ & b, LocalVect< T $_-$ , NEN $_->$ & be)

Localize Element Vector from a Vect instance.

#### **Parameters**

in	b	Global vector to be localized.	
ou	t be	Local vector, the length of which is the total number of element equations.	

#### Remarks

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

# void ElementNodeVector ( const Vect< $T_-$ > & b, LocalVect< $T_-$ , NEN\_- > & be, int dof )

Localize Element Vector from a Vect instance.

#### Parameters

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

#### Remarks

Only yhe dega dof is transferred to the local vector

# void ElementSideVector ( const Vect< T $_->$ & b, LocalVect< T $_-$ , NSE $_->$ & be)

Localize Element Vector from a Vect instance.

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

# void ElementVector ( const Vect< T $_->$ & b, int $dof\_type = NODE\_FIELD$ , int flag = 0)

Localize **Element** Vector.

#### Parameters

in	b	Global vector to be localized	
in	dof_type	DOF type option. To choose among the enumerated values:	
		NODE_FIELD, DOFs are supported by nodes [Default]	
		<ul> <li>ELEMENT_FIELD, DOFs are supported by elements</li> </ul>	
		SIDE_FIELD, DOFs are supported by sides	
in	flag	Option to set:	
		• = 0, All DOFs are taken into account [Default]	
		• != 0, Only DOF number dof is handled in the system	
		The resulting local vector can be accessed by attribute ePrev.	

# Remarks

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

# void SideVector ( const Vect< $T_- > \& b$ )

Localize Side Vector.

#### **Parameters**

in	b	Global vector to be localized	
		NODE_FIELD, DOFs are supported by nodes [ default ]	
		ELEMENT_FIELD, DOFs are supported by elements	
		SIDE_FIELD, DOFs are supported by sides	
		The resulting local vector can be accessed by attribute ePrev.	

#### Remarks

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

#### void ElementNodeCoordinates ( )

Localize coordinates of element nodes.

Coordinates are stored in array \_x[0], \_x[1], ... which are instances of class Point<real\_t>

#### Remarks

This member function uses the Side pointer \_theSide

#### void SideNodeCoordinates ( )

Localize coordinates of side nodes.

Coordinates are stored in array  $_{x}[0]$ ,  $_{x}[1]$ , ... which are instances of class  $_{coint}<_{coint}<_{coint}$ 

#### Remarks

This member function uses the **Element** pointer \_theElement

# void ElementAssembly ( Matrix $< T_- > *A$ )

Assemble element matrix into global one.

#### **Parameters**

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

#### Warning

The element pointer is given by the global variable the Element

# void ElementAssembly ( BMatrix $< T_- > & A$ )

Assemble element matrix into global one.

#### Parameters

A Global matrix stored as a BMatrix instance

# Warning

The element pointer is given by the global variable the Element

# void ElementAssembly ( SkSMatrix $< T_- > & A$ )

Assemble element matrix into global one.

#### **Parameters**

A Global matrix stored as an SkSMatrix instance

# Warning

The element pointer is given by the global variable the Element

### void ElementAssembly (SkMatrix $< T_- > & A$ )

Assemble element matrix into global one.

#### **Parameters**

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

# Warning

The element pointer is given by the global variable the Element

# void ElementAssembly (SpMatrix $< T_- > & A$ )

Assemble element matrix into global one.

#### **Parameters**

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

#### Warning

The element pointer is given by the global variable the Element

# void ElementAssembly ( $TrMatrix < T_- > & A$ )

Assemble element matrix into global one.

#### **Parameters**

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

#### Warning

The element pointer is given by the global variable the Element

# void DGElementAssembly ( Matrix $< T_- > *A$ )

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

### Parameters

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

#### Warning

The element pointer is given by the global variable the Element

OFELI's Reference Guide

#### void DGElementAssembly (SkSMatrix $< T_- > & A$ )

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

#### **Parameters**

A Global matrix stored as an SkSMatrix instance

#### Warning

The element pointer is given by the global variable the Element

#### void DGElementAssembly (SkMatrix $< T_- > & A$ )

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

#### **Parameters**

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

# Warning

The element pointer is given by the global variable the Element

# void DGElementAssembly (SpMatrix $< T_- > & A$ )

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

#### Parameters

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

#### Warning

The element pointer is given by the global variable the Element

### void DGElementAssembly ( $TrMatrix < T_- > & A$ )

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

#### Parameters

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

# Warning

The element pointer is given by the global variable the Element

### void SideAssembly ( Matrix $< T_- > *A$ )

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

#### **Parameters**

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

# Warning

The side pointer is given by the global variable the Side

# void SideAssembly (SkSMatrix $< T_- > & A$ )

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

#### **Parameters**

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

#### Warning

The side pointer is given by the global variable the Side

# void SideAssembly ( SkMatrix< $T_- > & A$ )

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

# Parameters

in	A	Global matrix stored as an SkMatrix instance

# Warning

The side pointer is given by the global variable the Side

# void SideAssembly (SpMatrix $< T_- > & A$ )

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

#### **Parameters**

	in	A	Global matrix stored as an SpMatrix instance	1
--	----	---	--	---

#### Warning

The side pointer is given by the global variable the Side

# void ElementAssembly ( Vect< T $_->$ & v )

Assemble element vector into global one.

#### **Parameters**

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

#### Warning

The element pointer is given by the global variable the Element

# void SideAssembly ( Vect< T $_->$ & v )

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

#### **Parameters**

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

#### Warning

The side pointer is given by the global variable the Side

# void AxbAssembly (const Element & el, const Vect< $T_-$ > & x, Vect< $T_-$ > & b)

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

#### **Parameters**

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

# void AxbAssembly ( const Side & sd, const Vect< T\_ > & x, Vect< T\_ > & b )

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

### Parameters

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

# real\_t setMaterialProperty ( const string & exp, const string & prop )

Define a material property by an algebraic expression.

#### **Parameters**

in	ехр	Algebraic expression
in	prop	Property name

#### Returns

Return value in expression evaluation:

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

# 7.36 Estimator Class Reference

To calculate an a posteriori estimator of the solution.

# **Public Types**

```
enum EstimatorType {
    ESTIM_ZZ = 0,
    ESTIM_ND_JUMP = 1 }
```

#### **Public Member Functions**

• Estimator ()

Default Constructor.

• Estimator (Mesh &m)

Constructor using finite element mesh.

• ∼Estimator ()

Destructor.

void setType (EstimatorType t=ESTIM\_ZZ)

Select type of a posteriori estimator.

• void setSolution (const Vect< real\_t > &u)

Provide solution vector in order to determine error index.

void getElementWiseIndex (Vect< real\_t > &e)

 $Get\ vector\ containing\ elementwise\ error\ index.$ 

• void getNodeWiseIndex (Vect< real\_t > &e)

Get vector containing nodewise error index.

• void getSideWiseIndex (Vect< real\_t > &e)

Get vector containing sidewise error index.

• real\_t getAverage () const

Return averaged error.

• Mesh & getMesh () const

Return a reference to the finite element mesh.

# 7.36.1 Detailed Description

To calculate an a posteriori estimator of the solution.

This class enables calculating an estimator of a solution in order to evaluate reliability. Estimation uses the so-called Zienkiewicz-Zhu estimator.

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

# 7.36.2 Member Enumeration Documentation

#### enum EstimatorType

Enumerate variable that selects an error estimator for mesh adaptation purposes

Enumerator

ESTIM\_ZZ Zhu-Zienckiewicz elementwise estimator ESTIM\_ND\_JUMP Normal derivative jump sidewise estimator

# **Constructor & Destructor Documentation**

#### Estimator (Mesh & m)

Constructor using finite element mesh.

**Parameters** 

7.36.3

in m	Mesh instance
------	---------------

# 7.36.4 Member Function Documentation

void setType ( EstimatorType  $t = ESTIM_ZZ$  )

Select type of a posteriori estimator.

**Parameters** 

in	t	Type of estimator. It has to be chosen among the enumerated values:	
		• ESTIM_ZZ: The Zhu-Zienckiewicz estimator (Default value)	
		• ESTIM_ND_JUMP: An estimator based on the jump of normal derivatives of the solution across mesh sides	

# void setSolution ( const Vect< real\_t > & u )

Provide solution vector in order to determine error index.

#### 7.37. FASTMARCHING2D CLASS REFERENCE CHAPTER 7. CLASS DOCUMENTATION

#### **Parameters**

j	in	и	Vector containing solution at mesh nodes
---	----	---	--

# void getElementWiseIndex ( Vect< real\_t > & e )

Get vector containing elementwise error index.

#### **Parameters**

in,out	e	Vector that contains once the member function setError is invoked a posteriori
		estimator at each element

# void getNodeWiseIndex ( Vect< real\_t > & e )

Get vector containing nodewise error index.

#### **Parameters**

in,out	е	Vector that contains once the member function setError is invoked a posteriori	
		estimator at each node	

#### void getSideWiseIndex ( Vect< real\_t > & e )

Get vector containing sidewise error index.

#### Parameters

in,out	е	Vector that contains once the member function setError is invoked a posteriori	
		estimator at each side	

# 7.37 FastMarching2D Class Reference

To run a Fast Marching Method on 2-D structured uniform grids.

# **Public Member Functions**

• FastMarching2D ()

Default constructor.

• FastMarching2D (const Grid &g, Vect< real\_t > &ls)

Constructor using grid and level set function.

• FastMarching2D (const Grid &g, Vect< real\_t > &ls, Vect< real\_t > &F)

Constructor using grid, level set function and velocity to extend.

• ~FastMarching2D ()

Destructor.

• void execute ()

Execute Fast Marching Procedure.

• void Check ()

Check distance function.

# 7.37.1 Detailed Description

To run a Fast Marching Method on 2-D structured uniform grids.

This class enables running a Fast Marching procedure to calculate the signed distance function and extend a given front speed.

#### Author

M. Sylla, B. Meden

# Copyright

GNU Lesser Public License

# 7.37.2 Constructor & Destructor Documentation

# FastMarching2D ( const Grid & g, Vect< real\_t > & ls )

Constructor using grid and level set function.

#### **Parameters**

in	8	Instance of class Grid
in	ls	Vector containing the level set function at grid nodes. The values are 0 on the interface (from which the distance is computed), positive on one side and negative on the other side. They must contain the signed distance on the nodes surrounding the interface but can take any value on other nodes, provided they have the right sign.

# FastMarching2D ( const Grid & g, Vect< real\_t > & ls, Vect< real\_t > & F)

Constructor using grid, level set function and velocity to extend.

in	8	Instance of class Grid
in	ls	Vector containing the level set function at grid nodes. The values are 0 on the interface (from which the distance is computed), positive on one side and negative on the other side. They must contain the signed distance on the nodes surrounding the interface but can take any value on other nodes, provided their sign is right.
in	F	Vector containing the front speed at grid nodes. Only values on nodes surrounding the interface are relevant.

#### 7.37.3 Member Function Documentation

#### void execute ( )

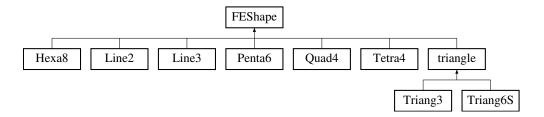
Execute Fast Marching Procedure.

Once this function was called, the vector 1s used in the constructor will contain the signed distance function and F will contain the extended speed.

# 7.38 FEShape Class Reference

Parent class from which inherit all finite element shape classes.

Inheritance diagram for FEShape:



### **Public Member Functions**

• FEShape ()

Default Constructor.

• FEShape (const Element \*el)

Constructor for an element.

FEShape (const Side \*sd)

Constructor for a side.

• virtual ~FEShape ()

Destructor.

• real\_t Sh (size\_t i) const

Return shape function of node i at given point.

real\_t Sh (size\_t i, Point < real\_t > s) const

Calculate shape function of node i at a given point s.

• real\_t getDet () const

Return determinant of jacobian.

• Point< real\_t > getCenter () const

Return coordinates of center of element.

• Point< real\_t > getLocalPoint () const

Localize a point in the element.

• Point< real\_t > getLocalPoint (const Point< real\_t > &s) const

Localize a point in the element.

# 7.38.1 Detailed Description

Parent class from which inherit all finite element shape classes.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

#### 7.38.2 Constructor & Destructor Documentation

#### FEShape (const Element \* el )

Constructor for an element.

#### Parameters

in	el	Pointer to element
----	----	--------------------

# FEShape (const Side \* sd)

Constructor for a side.

#### **Parameters**

in sd	Pointer to side
-------	-----------------

# 7.38.3 Member Function Documentation

# real\_t Sh ( size\_t i, Point< real\_t > s ) const

Calculate shape function of node i at a given point s.

#### Parameters

in	i	Local node label
in	S	Point in the reference triangle where the shape function is evaluated

# real\_t getDet ( ) const

Return determinant of jacobian.

If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calcuate relevant quantities.

# Point<real\_t> getLocalPoint ( ) const

Localize a point in the element.

Return actual coordinates in the reference element. If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calcuate relevant quantities.

### Point<real\_t> getLocalPoint ( const Point< real\_t> & s ) const

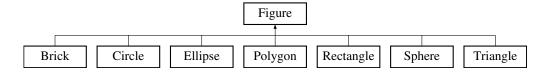
Localize a point in the element.

Return actual coordinates where s are coordinates in the reference element.

# 7.39 Figure Class Reference

To store and treat a figure (or shape) information.

Inheritance diagram for Figure:



# **Public Types**

```
    enum {
        UNION = 1,
        INTERSECTION = 2,
        MINUS = 3,
        SHIFT = 4,
        ROTATION = 5 }
```

#### **Public Member Functions**

• Figure ()

Default constructor.

• Figure (const Figure &f)

Copy constructor.

• virtual ~Figure ()

Destructor.

void setCode (int code)

Choose a code for the domain defined by the figure.

virtual real\_t getSignedDistance (const Point< real\_t > &p) const

Return signed distance from a given point to current figure.

• Figure & operator= (const Figure &f)

Operator =.

void getSignedDistance (const Grid &g, Vect< real\_t > &d) const

Calculate signed distance to current figure with respect to grid points.

real\_t dLine (const Point< real\_t > &p, const Point< real\_t > &a, const Point< real\_t > &b)
 const

Compute signed distance from a line.

# 7.39.1 Detailed Description

To store and treat a figure (or shape) information.

This class is essentially useful to construct data for mesh generators and for distance calculations.

#### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

#### 7.39.2 Member Function Documentation

virtual real\_t getSignedDistance ( const Point< real\_t > & p ) const [virtual]

Return signed distance from a given point to current figure.

#### **Parameters**

in	р	Point instance from which distance is computed
----	---	--

Reimplemented in Polygon, Triangle, Ellipse, Sphere, Circle, Brick, and Rectangle.

# void getSignedDistance ( const Grid & g, Vect< real t > & d ) const

Calculate signed distance to current figure with respect to grid points.

#### **Parameters**

in	8	Grid instance
in	d	Vect instance containing calculated distance from each grid index to Figure

#### Remarks

Vector d doesn't need to be sized before invoking this function

# real\_t dLine ( const Point< real\_t > & p, const Point< real\_t > & a, const Point< real\_t > & b) const

Compute signed distance from a line.

in	p	Point for which distance is computed
in	а	First vertex of line
in	b	Second vertex of line

Returns

Signed distance

# 7.40 FMM2D Class Reference

class for the fast marching 2-D algorithm Inherits FMM.

#### **Public Member Functions**

FMM2D (const Grid &g, Vect< real\_t > &phi, bool HA)

Constructor.

• FMM2D (const Grid &g, Vect< real\_t > &phi, const Vect< real\_t > &F, bool HA)

Constructor.

• void init ()

*Initialize the heap.* 

• void run ()

Execute Fast Marching Procedure.

• void eval (IPoint &pt, int sign)

compute the distance from node to interface

void ExtendSpeed (Vect< real\_t > &F)

Extend the speed function to the whole grid.

real\_t check\_error ()

Check error by comparing with the gradient norm.

# 7.40.1 Detailed Description

class for the fast marching 2-D algorithm

This class manages the 2-D Fast Marching method

Author

M. Sylla, B. Meden

Copyright

GNU Lesser Public License

#### 7.40.2 Constructor & Destructor Documentation

FMM2D ( const Grid & g, Vect< real\_t > & phi, bool HA )

Constructor.

Constructor using Grid instance

in	g	Instance of class Grid

#### Parameters

in	phi	Vector containing the level set function at grid nodes. The values are 0 on the interface (from which the distance is computed), positive on one side and negative on the other side. They must contain the signed distance on the nodes surrounding the interface but can take any value on other nodes, provided they have the right sign.
in	HA	true if the program must be executed with high accuracy, false otherwise

# FMM2D ( const Grid & g, Vect< real.t > & phi, const Vect< real.t > & F, bool HA)

#### Constructor.

Constructor using Grid instance

#### Parameters

in	8	Instance of class Grid
in	phi	Vector containing the level set function at grid nodes. The values are 0 on the interface (from which the distance is computed), positive on one side and negative on the other side. They must contain the signed distance on the nodes surrounding the interface but can take any value on other nodes, provided they have the right sign.
in	F	Right hand-side of the Eikonal equation
in	HA	true if the program must be executed with high accuracy, false otherwise

# 7.40.3 Member Function Documentation

# void eval ( IPoint & pt, int sign )

compute the distance from node to interface

#### Parameters

in	pt	node to treat
in	sign	Node sign

#### Returns

distance from node pt to interface

# void ExtendSpeed ( Vect< real\_t > & F )

Extend the speed function to the whole grid.

in,out	F	Vector containing the speed at interface nodes on input and extended speed to
		all grid nodes

#### real\_t check\_error( )

Check error by comparing with the gradient norm. This function returns discrete L<sup>2</sup> and Max. errors

# 7.41 FMM3D Class Reference

class for the 3-D fast marching algorithm Inherits FMM.

# **Public Member Functions**

• FMM3D (const Grid &g, Vect< real\_t > &phi, bool HA)

Constructor.

• FMM3D (const Grid &g, Vect< real\_t > &phi, const Vect< real\_t > &F, bool HA)

Constructor.

• void init ()

Initialize heap.

• void run ()

Execute Fast Marching Procedure.

• void eval (IPoint &pt, int sign)

Compute the distance from node to interface.

• void ExtendSpeed (Vect< real\_t > &F)

Extend the speed function to the whole grid.

real\_t check\_error ()

Check error by comparing with the gradient norm.

# 7.41.1 Detailed Description

class for the 3-D fast marching algorithm
This class manages the 3-D Fast Marching Method

#### 7.41.2 Constructor & Destructor Documentation

FMM3D ( const Grid & g, Vect< real\_t > & phi, bool HA )

Constructor.

Constructor using Grid instance

in	8	Instance of class Grid
in	phi	Vector containing the level set function at grid nodes. The values are 0 on the interface (from which the distance is computed), positive on one side and negative on the other side. They must contain the signed distance on the nodes surrounding the interface but can take any value on other nodes, provided they have the right sign.
in	HA	true if the program must be executed with high accuracy, false otherwise

#### Author

M. Sylla, B. Meden

# Copyright

GNU Lesser Public License

# FMM3D ( const Grid & g, Vect< real t > & phi, const Vect< real t > & F, bool HA )

#### Constructor.

Constructor using Grid instance

#### Parameters

in	8	Instance of class Grid
in	phi	Vector containing the level set function at grid nodes. The values are 0 on the interface (from which the distance is computed), positive on one side and negative on the other side. They must contain the signed distance on the nodes surrounding the interface but can take any value on other nodes, provided they have the right sign.
in	F	Vector containing the right-hand side of the Eikonal equation
in	HA	true if the program must be executed with high accuracy, false otherwise

#### Author

M. Sylla, B. Meden

# Copyright

GNU Lesser Public License

# 7.41.3 Member Function Documentation

# void eval ( IPoint & pt, int sign )

Compute the distance from node to interface.

#### Parameters

in	pt	Node to treat
in	sign	Node's sign

#### Returns

Distance from node pt to interface

# void ExtendSpeed ( Vect< real\_t > & F )

Extend the speed function to the whole grid.

#### **Parameters**

in,out	F	Vector containing the speed at interface nodes on input and extended speed at whole grid nodes
		whole grid nodes

#### real\_t check\_error ( )

Check error by comparing with the gradient norm. This function prints discrete L<sup>2</sup> and Max. errors

# 7.42 FMMSolver Class Reference

The Fast Marching Method solver.

#### **Public Member Functions**

• FMMSolver (const Grid &g, Vect< real\_t > &phi, bool ha=false)

Constructor.

• ~FMMSolver ()

Destructor.

• void run ()

Execute the fast marching program.

void ExtendSpeed (Vect< real\_t > &F)

Extend speed by Sethian's method.

real\_t check\_error ()

Return the consistency error of the method.

# 7.42.1 Detailed Description

The Fast Marching Method solver.

This class enables computing the signed distance function with respect to an interface. It works in 2-D and 3-D on a structured grid. The class is an interface for client. It points to FMM

Author

M. Sylla, B. Meden

Copyright

**GNU Lesser Public License** 

#### 7.42.2 Constructor & Destructor Documentation

FMMSolver (const Grid & g, Vect< real\_t > & phi, bool ha = false)

Constructor.

in <i>g</i> Instance of class Grid defining the grid on which the distance is computed.	
---	--

#### **Parameters**

in	phi	Vector containing the level set function at grid nodes. The vector entries are 0 on the interface (from which the distance is computed), positive on one side and negative on the other side. They must contain the signed distance on the nodes surrounding the interface. These values identify by linear interpolation the interface position. The vector entries can take any value on other grid nodes, provided they have the right sign.
in	ha	true if high accuracy FMM is active. The high accuracy version is more accurate but requires more accurate values on the nodes neighbouring the interface.

# 7.42.3 Member Function Documentation

# void ExtendSpeed ( Vect< real\_t > & F )

Extend speed by Sethian's method.

The method consists in calculating a speed F such that its gradient is orthogonal to the gradient of the level set function

#### **Parameters**

in,out	Speed function where on input the value of the function is meaningful on the interface. On output F contains the extended speed
	interface. On output F contains the extended speed

#### real\_t check\_error ( )

Return the consistency error of the method.

Consistency is measured by computing the discrete value of the norm of the gradient of the signed distance and subtracting the obtained norm from 1. The absolute value of the result is returned.

# 7.43 Funct Class Reference

A simple class to parse real valued functions.

# **Public Member Functions**

• Funct ()

Default constructor.

• Funct (string v)

Constructor for a function of one variable.

• Funct (string v1, string v2)

Constructor for a function of two variables.

• Funct (string v1, string v2, string v3)

Constructor for a function of three variables.

• Funct (string v1, string v2, string v3, string v4)

Constructor for a function of four variables.

• ~Funct ()

Destructor.

• real\_t operator() (real\_t x) const

Operator () to evaluate the function with one variable x

• real\_t operator() (real\_t x, real\_t y) const

Operator () to evaluate the function with two variables  $\boldsymbol{x},\,\boldsymbol{y}$ 

• real\_t operator() (real\_t x, real\_t y, real\_t z) const

Operator () to evaluate the function with three variables x, y, z

• real\_t operator() (real\_t x, real\_t y, real\_t z, real\_t t) const

Operator () to evaluate the function with four variables x, y, z

• void operator= (string e)

Operator = .

# 7.43.1 Detailed Description

A simple class to parse real valued functions.

Functions must have 1, 2, 3 or at most 4 variables.

#### Warning

Data in the file must be listed in the following order:

```
for x=x_0,...,x_I

for y=y_0,...,y_J

for z=z_0,...,z_K

read v(x,y,z)
```

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

#### 7.43.2 Constructor & Destructor Documentation

# Funct ( string v )

Constructor for a function of one variable.

#### Parameters

in	v	Name of the variable
----	---	----------------------

#### Funct ( string v1, string v2 )

Constructor for a function of two variables.

in	v1	Name of the first variable
in	<i>v</i> 2	Name of the second variable

#### Funct ( string v1, string v2, string v3 )

Constructor for a function of three variables.

#### Parameters

in	v1	Name of the first variable
in	<i>v</i> 2	Name of the second variable
in	<i>v</i> 3	Name of the third variable

# Funct ( string v1, string v2, string v3, string v4 )

Constructor for a function of four variables.

#### **Parameters**

in	v1	Name of the first variable
in	<i>v</i> 2	Name of the second variable
in	<i>v</i> 3	Name of the third variable
in	v4	Name of the fourth variable

# 7.43.3 Member Function Documentation

void operator= ( string e )

Operator =.

Define the function by an algebraic expression following regexp rules

### Parameters

in	e	Algebraic expression defining the function.
----	---	---

# 7.44 Gauss Class Reference

Calculate data for Gauss integration.

#### **Public Member Functions**

• Gauss ()

Default constructor.

• Gauss (size\_t np)

Constructor using number of Gauss points.

• void setNbPoints (size\_t np)

Set number of integration points.

• void setTriangle (LocalVect< real\_t, 7 > &w, LocalVect< Point< real\_t >, 7 > &x)

Choose integration on triangle (7-point formula)

• real\_t x (size\_t i) const

Return coordinate of i-th Gauss-Legendre point.

• const Point< real\_t > & xt (size\_t i) const

Return coordinates of points in the reference triangle.

• real\_t w (size\_t i) const

Return weight of i-th Gauss-Legendre point.

# 7.44.1 Detailed Description

Calculate data for Gauss integration.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# 7.44.2 Constructor & Destructor Documentation

Gauss ( size\_t np )

Constructor using number of Gauss points.

#### **Parameters**

	in	пр	Number of integration points
--	----	----	------------------------------

#### 7.44.3 Member Function Documentation

void setTriangle ( LocalVect< real\_t, 7 > & w, LocalVect< Point< real\_t >, 7 > & x)

Choose integration on triangle (7-point formula)

If this is not selected, Gauss integration formula on [-1,1] is calculated.

#### Parameters

out	w	Array of weights of integration points
out	х	Array of coordinates of integration points

# 7.45 Grid Class Reference

To manipulate structured grids.

# **Public Member Functions**

• Grid ()

Construct a default grid with 10 intervals in each direction.

• Grid (real\_t xm, real\_t xM, size\_t npx)

Construct a 1-D structured grid given its extremal coordinates and number of intervals.

• Grid (real\_t xm, real\_t xM, real\_t ym, real\_t yM, size\_t npx, size\_t npy)

Construct a 2-D structured grid given its extremal coordinates and number of intervals.

Grid (Point < real\_t > m, Point < real\_t > M, size\_t npx, size\_t npy)

Construct a 2-D structured grid given its extremal coordinates and number of intervals.

Grid (real\_t xm, real\_t xM, real\_t ym, real\_t yM, real\_t zm, real\_t zM, size\_t npx, size\_t npx, size\_t npz)

Construct a 3-D structured grid given its extremal coordinates and number of intervals.

Grid (Point < real\_t > m, Point < real\_t > M, size\_t npx, size\_t npy, size\_t npz)

Construct a 3-D structured grid given its extremal coordinates and number of intervals.

void setXMin (const Point < real\_t > &x)

Set min. coordinates of the domain.

- void setXMax (const Point < real\_t > &x)
- void setDomain (real\_t xmin, real\_t xmax)

Set Dimensions of the domain: 1-D case.

• void setDomain (real\_t xmin, real\_t xmax, real\_t ymin, real\_t ymax)

Set Dimensions of the domain: 2-D case.

- void setDomain (real\_t xmin, real\_t xmax, real\_t ymin, real\_t ymax, real\_t zmin, real\_t zmax) Set Dimensions of the domain: 3-D case.
- void setDomain (Point< real\_t > xmin, Point< real\_t > xmax)

Set Dimensions of the domain: 3-D case.

• const Point< real\_t > & getXMin () const

Return min. Coordinates of the domain.

• const Point < real\_t > & getXMax () const

Return max. Coordinates of the domain.

• void setN (size\_t nx, size\_t ny=0, size\_t nz=0)

Set number of grid intervals in the x, y and z-directions.

• size\_t getNx () const

Return number of grid intervals in the x-direction.

size\_t getNy () const

Return number of grid intervals in the y-direction.

• size\_t getNz () const

Return number of grid intervals in the z-direction.

• real\_t getHx () const

Return grid size in the x-direction.

• real\_t getHy () const

Return grid size in the y-direction.

real\_t getHz () const

Return grid size in the z-direction.

Point< real\_t > getCoord (size\_t i) const

Return coordinates a point with label i in a 1-D grid.

Point< real\_t > getCoord (size\_t i, size\_t j) const

Return coordinates a point with label (i, j) in a 2-D grid.
• Point< real\_t > getCoord (size\_t i, size\_t j, size\_t k) const

Patrima coordinates a noint suith label (i i h) in a 2 D anid

Return coordinates a point with label (i, j, k) in a 3-D grid.

real\_t getX (size\_t i) const

Return x-coordinate of point with index i

real\_t getY (size\_t j) const

Return y-coordinate of point with index j

real\_t getZ (size\_t k) const

Return z-coordinate of point with index k

Point2D< real\_t > getXY (size\_t i, size\_t j) const

Return coordinates of point with indices (i, j)

Point< real\_t > getXYZ (size\_t i, size\_t j, size\_t k) const

Return coordinates of point with indices (i, j, k)

real\_t getCenter (size\_t i) const

Return coordinates of center of a 1-D cell with indices i, i+1

Point< real\_t > getCenter (size\_t i, size\_t j) const

Return coordinates of center of a 2-D cell with indices (i,j), (i+1,j), (i+1,j+1), (i,j+1)

• Point< real\_t > getCenter (size\_t i, size\_t j, size\_t k) const

Return coordinates of center of a 3-D cell with indices (i, j, k), (i+1, j, k), (i+1, j+1, k), (i, j+1, k), (i, j, k+1), (i+1, j, k+1), (i+1, j+1, k+1), (i, j+1, k+1)

• void setCode (string exp, int code)

Set a code for some grid points.

• void setCode (int side, int code)

Set a code for grid points on sides.

• int getCode (int side) const

Return code for a side number.

int getCode (size\_t i, size\_t j) const

Return code for a grid point.

int getCode (size\_t i, size\_t j, size\_t k) const

Return code for a grid point.

• size\_t getDim () const

Return space dimension.

void Deactivate (size\_t i)

Change state of a cell from active to inactive (1-D grid)

• void Deactivate (size\_t i, size\_t j)

Change state of a cell from active to inactive (2-D grid)

void Deactivate (size\_t i, size\_t j, size\_t k)

Change state of a cell from active to inactive (2-D grid)

• int isActive (size\_t i) const

Say if cell is active or not (1-D grid)

• int isActive (size\_t i, size\_t j) const

Say if cell is active or not (2-D grid)

• int isActive (size\_t i, size\_t j, size\_t k) const

Say if cell is active or not (3-D grid)

# 7.45.1 Detailed Description

To manipulate structured grids.

Author

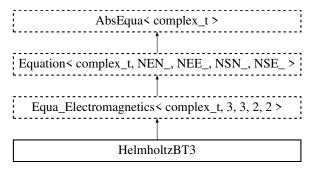
Rachid Touzani

Copyright

GNU Lesser Public License

# 7.46 HelmholtzBT3 Class Reference

Builds finite element arrays for Helmholtz equations in a bounded media using 3-Node triangles. Inheritance diagram for HelmholtzBT3:



# **Public Member Functions**

• HelmholtzBT3 ()

Default Constructor.

• HelmholtzBT3 (Mesh &ms)

Constructor using mesh data.

• HelmholtzBT3 (Mesh &ms, Vect< complex\_t > &u)

Constructor using mesh and solution vector.

• ~HelmholtzBT3 ()

Destructor.

• void build ()

Builds system of equations.

• void LHS ()

Add element Left-Hand Side.

void BodyRHS (Vect< complex\_t > &f)

Add element Right-Hand Side.

void BoundaryRHS (Vect< complex\_t > &f)

Add side Right-Hand Side.

# **Additional Inherited Members**

# 7.46.1 Detailed Description

Builds finite element arrays for Helmholtz equations in a bounded media using 3-Node triangles. Problem being formulated in time harmonics, the solution is complex valued.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

#### 7.46.2 Constructor & Destructor Documentation

#### HelmholtzBT3 ( Mesh & ms )

Constructor using mesh data.

#### **Parameters**

in ms Mesh insta
------------------

# HelmholtzBT3 ( Mesh & ms, Vect< complex\_t > & u )

Constructor using mesh and solution vector.

#### **Parameters**

in	ms	Mesh instance
in,out	и	Vect instance containing solution

# 7.47 Hexa8 Class Reference

Defines a three-dimensional 8-node hexahedral finite element using Q1-isoparametric interpolation.

Inheritance diagram for Hexa8:



#### **Public Member Functions**

• Hexa8 ()

Default Constructor.

• Hexa8 (const Element \*el)

Constructor when data of *Element* el are given.

• ~Hexa8 ()

Destructor.

• void setLocal (const Point < real\_t > &s)

Initialize local point coordinates in element.

• void atGauss (int n, std::vector< Point< real\_t >> &dsh, std::vector< real\_t > &w)

Calculate shape function derivatives and integration weights.

• void atGauss (int n, std::vector< real\_t > &sh, std::vector< real\_t > &w)

Calculate shape functions and integration weights.

• real\_t getMaxEdgeLength () const

Return maximal edge length.

• real\_t getMinEdgeLength () const

Return minimal edge length.

• Point< real\_t > Grad (const LocalVect< real\_t, 8 > &u, const Point< real\_t > &s)

Return gradient of a function defined at element nodes.

# 7.47.1 Detailed Description

Defines a three-dimensional 8-node hexahedral finite element using Q1-isoparametric interpolation.

The reference element is the cube [-1,1]x[-1,1]x[-1,1]. The user must take care to the fact that determinant of jacobian and other quantities depend on the point in the reference element where they are calculated. For this, before any utilization of shape functions or jacobian, function **getLocal(s)** must be invoked.

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

#### 7.47.2 Member Function Documentation

void setLocal ( const Point < real t > & s )

Initialize local point coordinates in element.

#### **Parameters**

in	S	Point in the reference element This function computes jacobian, shape functions and
		their partial derivatives at s. Other member functions only return these values.

void atGauss ( int n, std::vector< Point< real\_t >> & dsh, std::vector< real\_t > & w)

Calculate shape function derivatives and integration weights.

#### **Parameters**

in	n	Number of Gauss-Legendre integration points in each direction
in	dsh	Vector of shape function derivatives at the Gauss points
in	w	Weights of integration formula at Gauss points

void atGauss ( int n, std::vector< real\_t > & sh, std::vector< real\_t > & w)

Calculate shape functions and integration weights.

in	n	Number of Gauss-Legendre integration points in each direction
in	sh	Vector of shape functions at the Gauss points

#### **Parameters**

in	w	Weights of integration formula at Gauss points
----	---	--

#### Point<real\_t> Grad ( const LocalVect< real\_t, 8 > & u, const Point< real\_t > & s )

Return gradient of a function defined at element nodes.

#### **Parameters**

in	и	Vector of values at nodes
in	S	Local coordinates (in $[-1,1]*[-1,1]*[-1,1]$ ) of point where the gradient is evaluated

#### Returns

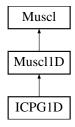
Value of gradient

#### Note

If the derivatives of shape functions were not computed before calling this function (by calling setLocal), this function will compute them

# 7.48 ICPG1D Class Reference

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 1-D. Inheritance diagram for ICPG1D:



# **Public Member Functions**

• ICPG1D (Mesh &ms)

Constructor using Mesh instance.

• ICPG1D (Mesh &ms, Vect< real\_t > &r, Vect< real\_t > &v, Vect< real\_t > &p)

Constructor using mesh and initial data.

• ~ICPG1D ()

Destructor.

• void setReconstruction ()

Set reconstruction from class Muscl.

• real\_t runOneTimeStep ()

Advance one time step.

void Forward (const Vect< real\_t > &flux, Vect< real\_t > &field)

Add flux to field.

• void setSolver (SolverType solver)

Choose solver type.

• void setGamma (real\_t gamma)

Set value of constant Gamma for gases.

• void setCv (real\_t Cv)

*Set value of Cv (specific heat at constant volume)* 

void setCp (real\_t Cp)

*Set value of*  $C_p$  (specific heat at constant pressure)

• void setKappa (real\_t Kappa)

Set value of constant Kappa.

• real\_t getGamma () const

Return value of constant Gamma.

real\_t getCv () const

*Return value of*  $C_v$  (specific heat at constant volume)

real\_t getCp () const

Return value of  $C_p$  (specific heat at constant pressure)

• real\_t getKappa () const

Return value of constant Kappa.

void getMomentum (Vect< real\_t > &m) const

Get vector of momentum at elements.

• void getInternalEnergy (Vect< real\_t > &ie) const

Get vector of internal energy at elements.

void getTotalEnergy (Vect< real\_t > &te) const

Get vector of total energy at elements.

• void getSoundSpeed (Vect< real\_t > &s) const

Get vector of sound speed at elements.

• void getMach (Vect< real\_t > &m) const

Get vector of elementwise Mach number.

• void setInitialCondition\_shock\_tube (const LocalVect< real\_t, 3 > &BcG, const LocalVect< real\_t, 3 > &BcD, real\_t x0)

*Initial condition corresponding to the shock tube.* 

• void setInitialCondition (const LocalVect< real\_t, 3 > &u)

A constant initial condition.

• void setBC (const Side &sd, real\_t u)

Assign a boundary condition as a constant to a given side.

• void setBC (int code, real\_t a)

Assign a boundary condition value.

• void setBC (real\_t a)

Assign a boundary condition value.

• void setBC (const Side &sd, const LocalVect< real\_t, 3 > &u)

Assign a Dirichlet boundary condition vector.

• void setBC (int code, const LocalVect< real\_t, 3 > &U)

Assign a Dirichlet boundary condition vector.

• void setBC (const LocalVect < real\_t, 3 > &u)

Assign a Dirichlet boundary condition vector.

- void setInOutflowBC (const Side &sd, const LocalVect< real\_t, 3 > &u)

  Impose a constant inflow or outflow boundary condition on a given side.
- void setInOutflowBC (int code, const LocalVect< real\_t, 3 > &u)

Impose a constant inflow or outflow boundary condition on sides with a given code.

• void setInOutflowBC (const LocalVect< real\_t, 3 > &u)

Impose a constant inflow or outflow boundary condition on boundary sides.

#### **Additional Inherited Members**

# 7.48.1 Detailed Description

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 1-D. Solution method is a second-order MUSCL Finite Volume scheme

#### Author

S. Clain, V. Clauzon

# Copyright

**GNU Lesser Public License** 

#### 7.48.2 Constructor & Destructor Documentation

ICPG1D ( Mesh & ms, Vect< real\_t > & r, Vect< real\_t > & v, Vect< real\_t > & p )

Constructor using mesh and initial data.

#### Parameters

in	ms	Reference to Mesh instance
in	r	Vector containing initial (elementwise) density
in	v	Vector containing initial (elementwise) velocity
in	р	Vector containing initial (elementwise) pressure

#### 7.48.3 Member Function Documentation

void Forward ( const Vect< real\_t > & flux, Vect< real\_t > & field )

Add flux to field.

If this function is used, the user must call getFlux himself

#### **Parameters**

in	flux	Vector containing fluxes at sides (points)
out	field	Vector containing solution vector

# void getMomentum ( Vect< real\_t > & m ) const

Get vector of momentum at elements.

#### Parameters

in,out	m	Vect instance that contains on output element momentum	
--------	---	--	--

# void getInternalEnergy ( Vect< real\_t > & ie ) const

Get vector of internal energy at elements.

#### **Parameters**

in,out	ie	Vect instance that contains on output element internal energy	
--------	----	---	--

# void getTotalEnergy ( $Vect < real_t > \& te$ ) const

Get vector of total energy at elements.

#### Parameters

in,out	te	Vect instance that contains on output element total energy
--------	----	--

# void getSoundSpeed ( Vect< real\_t > & s ) const

Get vector of sound speed at elements.

#### Parameters

in,out	S	Vect instance that contains on output element sound speed
	l	

# void getMach ( Vect< real\_t > & m ) const

Get vector of elementwise Mach number.

#### Parameters

in,out	m	Vect instance that contains on output element Mach number
--------	---	---

# void setInitialCondition ( const LocalVect< real\_t, 3 > & u )

A constant initial condition.

d			
	in	и	LocalVect instance containing density, velocity and pressure

# void setBC ( const Side & sd, real\_t u )

Assign a boundary condition as a constant to a given side.

#### Parameters

in	sd	Side to which the value is assigned
in	и	Value to assign

# void setBC ( int code, real\_t a )

Assign a boundary condition value.

#### Parameters

in	code	Code value to which boundary condition is assigned
in	а	Value to assign to sides that have code code

# void setBC ( real\_t a )

Assign a boundary condition value.

#### Parameters

in	а	Value to assign to all boundary sides
----	---	---------------------------------------

# void setBC ( const Side & sd, const LocalVect< real\_t, 3 > & u)

Assign a Dirichlet boundary condition vector.

#### Parameters

in	sd	Side instance to which the values are assigned
in	и	LocalVect instance that contains values to assign to the side

# void setBC ( int code, const LocalVect< real\_t, 3 > & U )

Assign a Dirichlet boundary condition vector.

in	code	Side code for which the values are assigned
in	U	LocalVect instance that contains values to assign to sides with code <i>code</i>

#### void setBC ( const LocalVect< real\_t, 3 > & u )

Assign a Dirichlet boundary condition vector.

#### **Parameters**

in	и	LocalVect instance that contains values to assign to all boundary sides
----	---	---

### void setInOutflowBC ( const Side & sd, const LocalVect< real\_t, 3 > & u)

Impose a constant inflow or outflow boundary condition on a given side.

#### Parameters

in	sd	Instance of Side on which the condition is prescribed
in	и	LocalVect instance that contains values to assign to the side

### void setInOutflowBC ( int code, const LocalVect< real\_t, 3 > & u )

Impose a constant inflow or outflow boundary condition on sides with a given code.

#### **Parameters**

in	code	Value of code for which the condition is prescribed	
in	и	LocalVect instance that contains values to assign to the sides	

### void setInOutflowBC ( const LocalVect< real\_t, 3 > & u )

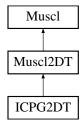
Impose a constant inflow or outflow boundary condition on boundary sides.

#### Parameters

in	и	LocalVect instance that contains values to assign to the sides

### 7.49 ICPG2DT Class Reference

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 2-D. Inheritance diagram for ICPG2DT:



### **Public Member Functions**

• ICPG2DT (Mesh &ms)

Constructor using mesh instance.

• ICPG2DT (Mesh &ms, Vect< real\_t > &r, Vect< real\_t > &v, Vect< real\_t > &p)

Constructor using mesh and initial data.

• ~ICPG2DT ()

Destructor.

• void setReconstruction ()

Reconstruct.

real\_t runOneTimeStep ()

Advance one time step.

• void Forward (const Vect< real\_t > &Flux, Vect< real\_t > &Field)

Add Flux to Field.

• real\_t getFlux ()

Get flux.

• void setSolver (SolverType s)

Choose solver.

• void setGamma (real\_t gamma)

Set Gamma value.

• void setCv (real\_t Cv)

Set value of heat capacity at constant volume.

• void setCp (real\_t Cp)

Set value of heat capacity at constant pressure.

• void setKappa (real\_t Kappa)

Set Kappa value.

real\_t getGamma () const

Return value of Gamma.

real\_t getCv () const

Return value of heat capacity at constant volume.

real\_t getCp () const

Return value of heat capacity at constant pressure.

• real\_t getKappa () const

Return value of Kappa.

• Mesh & getMesh ()

Return reference to mesh instance.

• void getMomentum (Vect< real\_t > &m) const

Calculate elementwise momentum.

• void getInternalEnergy (Vect< real\_t > &e) const

Calculate elementwise internal energy.

void getTotalEnergy (Vect< real\_t > &e) const

Return elementwise total energy.

• void getSoundSpeed (Vect< real\_t > &s) const

Return elementwise sound speed.

• void getMach (Vect< real\_t > &m) const

Return elementwise Mach number.

void setInitialConditionShockTube (const LocalVect< real\_t, 4 > &BcL, const LocalVect< real\_t, 4 > &BcR, real\_t x0)

Set initial condition for the schock tube problem.

• void setInitialCondition (const LocalVect< real\_t, 4 > &u)

Set initial condition.

• void setBC (const Side &sd, real\_t a)

Prescribe a constant boundary condition at given side.

• void setBC (int code, real\_t a)

Prescribe a constant boundary condition for a given code.

• void setBC (real\_t u)

Prescribe a constant boundary condition on all boundary sides.

• void setBC (const Side &sd, const LocalVect< real\_t, 4 > &u)

Prescribe a constant boundary condition at a given side.

• void setBC (int code, const LocalVect< real\_t, 4 > &u)

Prescribe a constant boundary condition for a given code.

• void setBC (const LocalVect< real\_t, 4 > &u)

Prescribe a constant boundary condition at all boundary sides.

• real\_t getR (size\_t i) const

Return density at given element label.

- real\_t getV (size\_t i, size\_t j) const
- real\_t getP (size\_t i) const

Return pressure at given element label.

#### **Additional Inherited Members**

### 7.49.1 Detailed Description

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 2-D. Solution method is a second-order MUSCL Finite Volume scheme on triangles

Author

S. Clain, V. Clauzon

Copyright

**GNU Lesser Public License** 

### 7.49.2 Constructor & Destructor Documentation

ICPG2DT ( Mesh & ms, Vect< real\_t > & r, Vect< real\_t > & v, Vect< real\_t > & p)

Constructor using mesh and initial data.

in	ms	Mesh instance
in	r	Initial density vector (as instance of Vect)
in	v	Initial velocity vector (as instance of Vect)
in	p	Initial pressure vector (as instance of Vect)

### 7.49.3 Member Function Documentation

#### void setReconstruction ( )

Reconstruct.

exit(3) if reconstruction fails

### void Forward ( const Vect< real\_t > & Flux, Vect< real\_t > & Field )

Add Flux to Field.

If this function is used, the function getFlux must be called

### void setSolver ( SolverType s )

Choose solver.

#### **Parameters**

in	S	Index of solver in the enumerated variable SolverType Available values are:
		ROE_SOLVER, VFROE_SOLVER, LF_SOLVER, RUSANOV_SOLVER, HLL_SOLVER, HLLC_SOLVER,
		MAX_SOLVER

### void setBC ( const Side & sd, real\_t a )

Prescribe a constant boundary condition at given side.

### Parameters

in	sd	Reference to Side instance
in	а	Value to prescribe

### void setBC ( int code, real\_t a )

Prescribe a constant boundary condition for a given code.

#### **Parameters**

in	code	Code for which value is imposed
in	а	Value to prescribe

### void setBC ( real\_t u )

Prescribe a constant boundary condition on all boundary sides.

in	и	Value to prescribe
		I

### void setBC ( const Side & sd, const LocalVect< real\_t, 4 > & u )

Prescribe a constant boundary condition at a given side.

#### Parameters

in	sd	Reference to Side instance
in	и	Vector (instance of class LocalVect) with as components the constant values to
		prescribe for the four fields (r, vx, vy, p)

### void setBC ( int *code*, const LocalVect< real\_t, 4 > & u )

Prescribe a constant boundary condition for a given code.

#### **Parameters**

in	code	Code for which value is imposed
in	и	Vector (instance of class LocalVect) with as components the constant values to
		prescribe for the four fields (r, vx, vy, p)

### void setBC ( const LocalVect< real\_t, 4 > & u )

Prescribe a constant boundary condition at all boundary sides.

### Parameters

in	и	Vector (instance of class Local Vect) with as components the constant values to
		prescribe for the four fields (r, vx, vy, p)

### real\_t getR ( size\_t i ) const

Return density at given element label.

#### Parameters

in	i	Element label

### real\_t getV ( size\_t i, size\_t j ) const

Return velocity at given element label

in	i	Element label
in	j	component index (1 or 2)

#### real\_t getP ( size\_t i ) const

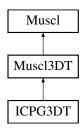
Return pressure at given element label.

#### **Parameters**

in i Element label
--------------------

### 7.50 ICPG3DT Class Reference

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 3-D. Inheritance diagram for ICPG3DT:



#### **Public Member Functions**

• ICPG3DT (Mesh &ms)

Constructor using mesh data.

• ICPG3DT (Mesh &ms, Vect< real\_t > &r, Vect< real\_t > &v, Vect< real\_t > &p)

Constructor using mesh and initial data.

• ~ICPG3DT ()

Destructor.

• void setReconstruction ()

Reconstruct.

• real\_t runOneTimeStep ()

Advance one time step.

void Forward (const Vect< real\_t > &flux, Vect< real\_t > &field)

Add flux to field.

• real\_t getFlux ()

Return flux.

• void setSolver (SolverType solver)

Choose solver.

• void setReferenceLength (real\_t dx)

Assign a reference length.

• void setTimeStep (real\_t dt)

Assign a time step.

• void setCFL (real\_t CFL)

Assign CFL value.

• real\_t getReferenceLength () const

Return reference length.

real\_t getTimeStep () const

Return time step.

• real\_t getCFL () const

Return CFL.

• void setGamma (real\_t gamma)

Set  $\gamma$  value.

• void setCv (real\_t Cv)

*Set value of*  $C_v$  (*Heat capacity at constant volume*)

void setCp (real\_t Cp)

*Set value of*  $C_v$  (*Heat capacity at constant pressure*)

void setKappa (real\_t Kappa)

Set Kappa value.

• real\_t getGamma () const

Return value of  $\gamma$ .

real\_t getCv () const

Return value of  $C_v$  (Heat capacity at constant volume)

real\_t getCp () const

Return value of  $C_p$  (Heat capacity at constant pressure)

• real\_t getKappa () const

Return value of  $\kappa$ .

Mesh & getMesh ()

Return reference to mesh instance.

Mesh \* getPtrMesh ()

Return pointer to mesh.

• void getMomentum (Vect< real\_t > &m) const

Calculate elementwise momentum.

• void getInternalEnergy (Vect< real\_t > &e) const

Calculate elementwise internal energy.

• void getTotalEnergy (Vect< real\_t > &e) const

Return elementwise total energy.

• void getSoundSpeed (Vect< real\_t > &s) const

Return elementwise sound speed.

• void getMach (Vect< real\_t > &m) const

Return elementwise Mach number.

void setInitialConditionShockTube (const LocalVect< real\_t, 5 > &BcG, const LocalVect< real\_t, 5 > &BcD, real\_t x0)

Set initial condition for the schock tube problem.

• void setInitialCondition (const LocalVect< real\_t, 5 > &u)

Set initial condition.

### **Additional Inherited Members**

### 7.50.1 Detailed Description

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 3-D. Solution method is a second-order MUSCL Finite Volume scheme with tetrahedra

#### Author

S. Clain, V. Clauzon

### Copyright

**GNU Lesser Public License** 

### 7.50.2 Constructor & Destructor Documentation

#### ICPG3DT ( Mesh & ms )

Constructor using mesh data.

#### Parameters

	in	ms	Mesh instance
--	----	----	---------------

#### ICPG3DT ( Mesh & ms, Vect< real\_t > & r, Vect< real\_t > & v, Vect< real\_t > & p)

Constructor using mesh and initial data.

#### **Parameters**

in	ms	Mesh instance
in	r	Elementwise initial density vector (as instance of Element Vect)
in	v	Elementwise initial velocity vector (as instance of Element Vect)
in	р	Elementwise initial pressure vector (as instance of Element Vect)

#### 7.50.3 Member Function Documentation

void setReconstruction ( )

Reconstruct.

exit(3) if reconstruction failed

# 7.51 Integration Class Reference

Class for numerical integration methods.

### **Public Member Functions**

• Integration ()

Default constructor.

• Integration (real\_t low, real\_t high, function< real\_t(real\_t)> const &f, IntegrationScheme s, real\_t error)

Constructor.

• ~Integration ()

Destructor.

• void setFunction (function < real\_t(real\_t) > const &f)

Define function to integrate numerically.

• void setScheme (IntegrationScheme s)

Set time inegration scheme.

• void setTriangle (real\_t x1, real\_t y1, real\_t x2, real\_t y2, real\_t x3, real\_t y3)

Define integration domain as a quadrilateral.

• void setQuadrilateral (real\_t x1, real\_t y1, real\_t x2, real\_t y2, real\_t x3, real\_t y3, real\_t x4, real\_t y4)

 $Define\ integration\ domain\ as\ a\ quadrilateral.$ 

• real\_t run ()

Run numerical integration.

### 7.51.1 Detailed Description

Class for numerical integration methods.

Class NumInt defines and stores numerical integration data

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

### 7.51.2 Constructor & Destructor Documentation

Integration ( real\_t low, real\_t high, function < real\_t(real\_t) > const & f, IntegrationScheme s, real\_t error)

Constructor.

in	low	Lower value of integration interval
in	high	Upper value of integration interval
in	f	Function to integrate
in	S	Integration scheme. To choose among enumerated values:
		• LEFT_RECTANGLE:
		• RIGHT_RECTANGLE:
		• MID_RECTANGLE:
		• TRAPEZOIDAL:
		• SIMPSON:
		• GAUSS_LEGENDRE:
in	error	

### 7.51.3 Member Function Documentation

### void setFunction ( function < real\_t(real\_t) > const & f )

Define function to integrate numerically.

#### Parameters

-	in	f	Function to integrate
---	----	---	-----------------------

### void setScheme ( IntegrationScheme s )

Set time inegration scheme.

#### Parameters

in	S	Scheme to choose among enumerated values:
		• LEFT_RECTANGLE:
		• RIGHT_RECTANGLE:
		• MID_RECTANGLE:
		• TRAPEZOIDAL:
		• SIMPSON:
		• GAUSS_LEGENDRE:

### void setTriangle ( real\_t x1, real\_t y1, real\_t x2, real\_t y2, real\_t x3, real\_t y3 )

Define integration domain as a quadrilateral.

#### Parameters

in	<i>x</i> 1	x-coordinate of first vertex of triangle
in	у1	y-coordinate of first vertex of triangle
in	<i>x</i> 2	x-coordinate of second vertex of triangle
in	<i>y</i> 2	y-coordinate of second vertex of triangle
in	х3	x-coordinate of third vertex of triangle
in	у3	y-coordinate of third vertex of triangle

# void setQuadrilateral ( real\_t x1, real\_t y1, real\_t x2, real\_t y2, real\_t x3, real\_t y3, real\_t x4, real\_t y4)

Define integration domain as a quadrilateral.

#### **Parameters**

in	x1	x-coordinate of first vertex of quadrilateral
in	у1	y-coordinate of first vertex of quadrilateral
in	<i>x</i> 2	x-coordinate of second vertex of quadrilateral
in	<i>y</i> 2	y-coordinate of second vertex of quadrilateral
in	х3	x-coordinate of third vertex of quadrilateral
in	у3	y-coordinate of third vertex of quadrilateral
in	<i>x</i> 4	x-coordinate of fourth vertex of quadrilateral
in	y4	y-coordinate of fourth vertex of quadrilateral

#### real\_t run ( )

Run numerical integration.

Returns

Computed approximate value of integral

### 7.52 IOField Class Reference

Enables working with files in the XML Format. Inherits XMLParser.

### **Public Types**

```
    enum AccessType {
        IN = 1,
        OUT = 2 }
```

Enumerated values for file access type.

#### **Public Member Functions**

• IOField ()

Default constructor.

- IOField (const string &file, AccessType access, bool compact=true)
  - Constructor using file name.
- IOField (const string &mesh\_file, const string &file, Mesh &ms, AccessType access, bool compact=true)

Constructor using file name, mesh file and mesh.

- IOField (const string &file, Mesh &ms, AccessType access, bool compact=true)
  - Constructor using file name and mesh.
- IOField (const string &file, AccessType access, const string &name)

Constructor using file name and field name.

• ~IOField ()

Destructor.

• void setMeshFile (const string &file)

Set mesh file.

• void open ()

Open file.

• void open (const string &file, AccessType access)

Open file.

• void close ()

Close file.

void put (Mesh &ms)

Store mesh in file.

• void put (const Vect< real\_t > &v)

Store Vect instance v in file.

• real\_t get (Vect< real\_t > &v)

Get Vect v instance from file.

• int get (Vect< real\_t > &v, const string &name)

Get Vect v instance from file if the field has the given name.

• int get (DMatrix< real\_t > &A, const string &name)

Get *DMatrix* A instance from file if the field has the given name.

int get (DSMatrix < real\_t > &A, const string &name)

Get DSMatrix A instance from file if the field has the given name.

• int get (Vect< real\_t > &v, real\_t t)

Get Vect v instance from file corresponding to a specific time value.

void saveGMSH (string output\_file, string mesh\_file)

Save field vectors in a file using GMSH format.

### 7.52.1 Detailed Description

Enables working with files in the XML Format.

This class has methods to store vectors in files and read from files.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

### 7.53 IPF Class Reference

To read project parameters from a file in IPF format.

#### **Public Member Functions**

• IPF ()

Default constructor.

• IPF (const string &file)

Constructor that gives the data file name.

• IPF (const string &prog, const string &file)

Constructor that reads parameters in file file and prints header information for the calling program prog. It reads parameters in IPF Format from this file.

• ~IPF ()

Destructor.

real\_t getDisplay ()

Display acquired parameters.

• int getVerbose ()

Return parameter read using keyword Verbose.

• int getOutput () const

Return parameter read using keyword **Output**.

• int getSave () const

Return parameter read using keyword Save.

• int getPlot () const

Return parameter read using keyword Plot.

• int getBC () const

Return parameter read using keyword BC.

• int getBF () const

Return parameter read using keyword BF.

• int getSF () const

Return parameter read using keyword SF.

• int getInit () const

Return parameter read using keyword Init.

• int getData () const

Return parameter read using keyword **Data**.

• size\_t getNbSteps () const

Return parameter read using keyword NbSteps.

• size\_t getNbIter () const

Return parameter read using keyword NbIter.

real\_t getTimeStep () const

Return parameter read using keyword TimeStep.

• real\_t getMaxTime () const

Return parameter read using keyword **MaxTime**.

real\_t getTolerance () const

Return parameter read using keyword Tolerance.

• int getIntPar (size\_t n=1) const

Return n-th parameter read using keyword IntPar

• string getStringPar (size\_t n=1) const

Return n-th parameter read using keyword **StringPar**.

• real\_t getDoublePar (size\_t n=1) const

Return n-th parameter read using keyword DoublePar.

• Point< real\_t > getPointDoublePar (size\_t n=1) const

Return n-th parameter read using keyword PointDoublePar.

complex\_t getComplexPar (size\_t n=1) const

Return n-th parameter read using keyword StringPar.

• string getString (const string &label) const

Return parameter corresponding to a given label, when its value is a string.

• string getString (const string &label, string def) const

Return parameter corresponding to a given label, when its value is a string.

• int getInteger (const string &label) const

Return parameter corresponding to a given label, when its value is an integer.

• int getInteger (const string &label, int def) const

Return parameter corresponding to a given label, when its value is an integer.

• real\_t getDouble (const string &label) const

Return parameter corresponding to a given label, when its value is a real\_t.

real\_t getDouble (const string &label, real\_t def) const

Return parameter corresponding to a given label, when its value is a real\_t.

complex\_t getComplex (const string &label) const

Return parameter corresponding to a given label, when its value is a complex number.

complex\_t getComplex (const string &label, complex\_t def) const

Return parameter corresponding to a given label, when its value is a complex number.

int contains (const string &label) const

check if the project file contains a given parameter

• void get (const string &label, Vect< real\_t > &a) const

Read an array of real values, corresponding to a given label.

real\_t getArraySize (const string &label, size\_t j) const

Return an array entry for a given label.

• void get (const string &label, int &a) const

Return integer parameter corresponding to a given label.

• void get (const string &label, real\_t &a) const

Return real parameter corresponding to a given label.

void get (const string &label, complex\_t &a) const

Return complex parameter corresponding to a given label.

• void get (const string &label, string &a) const

Return string parameter corresponding to a given label.

• string getProject () const

Return parameter read using keyword **Project**.

• string getDomainFile () const

Return pameter using keyword Mesh.

string getMeshFile (size\_t i=1) const

Return i-th parameter read using keyword mesh\_file.

• string getInitFile () const

Return parameter read using keyword **InitFile**.

string getRestartFile () const

Return parameter read using keyword **RestartFile**.

• string getBCFile () const

Return parameter read using keyword BCFile.

• string getBFFile () const

Return parameter read using keyword **BFFile**.

• string getSFFile () const

Return parameter read using keyword SFFile.

• string getSaveFile () const

Return parameter read using keyword SaveFile.

• string getPlotFile (int i=1) const

Return i-th parameter read using keyword PlotFile.

• string getPrescriptionFile (int i=1) const

Return parameter read using keyword DataFile.

• string getAuxFile (size\_t i=1) const

Return i-th parameter read using keyword Auxfile.

• string getDensity () const

Return expression (to be parsed, function of x, y, z, t) for density function.

• string getElectricConductivity () const

Return expression (to be parsed, function of x, y, z, t) for electric conductivity.

• string getElectricPermittivity () const

Return expression (to be parsed, function of x, y, z, t) for electric permittivity.

• string getMagneticPermeability () const

Return expression (to be parsed, function of x, y, z, t) for magnetic permeability.

• string getPoissonRatio () const

Return expression (to be parsed, function of x, y, z, t) for Poisson ratio.

• string getThermalConductivity () const

Return expression (to be parsed, function of x, y, z, t) for thermal conductivity.

string getRhoCp () const

Return expression (to be parsed, function of x, y, z, t) for density \* specific heat.

• string getViscosity () const

Return expression (to be parsed, function of x, y, z, t) for viscosity.

• string getYoungModulus () const

Return expression (to be parsed, function of x, y, z, t) for Young's modulus.

### 7.53.1 Detailed Description

To read project parameters from a file in IPF format.

This class can be used to acquire various parameters from a parameter file of IPF (Input Project File). The declaration of an instance of this class avoids reading data in your main program. The acquired parameters are retrieved through information members of the class. Note that all the parameters have default values

Author

Rachid Touzani

Copyright

GNU Lesser Public License

#### 7.53.2 Constructor & Destructor Documentation

IPF (const string & file)

Constructor that gives the data file name.

It reads parameters in IPF Format from this file.

### 7.53.3 Member Function Documentation

#### int getOutput ( ) const

Return parameter read using keyword **Output**.

This parameter can be used to control output behavior in a program.

#### int getSave ( ) const

Return parameter read using keyword Save.

This parameter can be used to control result saving in a program (e.g. for a restarting purpose).

### int getPlot ( ) const

Return parameter read using keyword Plot.

This parameter can be used to control result saving for plotting in a program.

#### int getBC ( ) const

Return parameter read using keyword BC.

This parameter can be used to set a boundary condition flag.

#### int getBF ( ) const

Return parameter read using keyword BF.

This parameter can be used to set a body force flag.

#### int getSF ( ) const

Return parameter read using keyword SF.

This parameter can be used to set a surface force flag.

#### int getInit( ) const

Return parameter read using keyword Init.

This parameter can be used to set an initial data flag.

### int getData ( ) const

Return parameter read using keyword Data.

This parameter can be used to set a various data flag.

### size\_t getNbSteps ( ) const

Return parameter read using keyword NbSteps.

This parameter can be used to read a number of time steps.

### size\_t getNbIter ( ) const

Return parameter read using keyword NbIter.

This parameter can be used to read a number of iterations.

#### real\_t getTimeStep ( ) const

Return parameter read using keyword **TimeStep**.

This parameter can be used to read a time step value.

### real\_t getMaxTime ( ) const

Return parameter read using keyword MaxTime.

This parameter can be used to read a maximum time value.

#### real\_t getTolerance ( ) const

Return parameter read using keyword **Tolerance**.

This parameter can be used to read a tolerance value to control convergence.

#### int getIntPar ( size\_t n = 1 ) const

Return n-th parameter read using keyword IntPar

Here we have at most 20 integer extra parameters that can be used for any purpose. Default value for n is 1

#### string getStringPar ( $size_t n = 1$ ) const

Return *n-th* parameter read using keyword **StringPar**.

Here we have at most 20 integer extra parameters that can be used for any purpose. Default value for n is 1

### real\_t getDoublePar ( size\_t n = 1 ) const

Return n-th parameter read using keyword **DoublePar**.

Here we have at most 20 integer extra parameters that can be used for any purpose. Default value for n is 1

### Point<real\_t> getPointDoublePar ( size\_t n = 1 ) const

Return n-th parameter read using keyword PointDoublePar.

Here we have at most 20 integer extra parameters that can be used for any purpose. Default value for n is 1

#### complex\_t getComplexPar ( size\_t n = 1 ) const

Return n-th parameter read using keyword **StringPar**.

Here we have at most 20 integer extra parameters that can be used for any purpose. Default value for n is 1

### string getString (const string & label) const

Return parameter corresponding to a given label, when its value is a string.

#### **Parameters**

in	label	Label that identifies the string (read from input file) If this label is not found an
		error message is displayed and program stops

#### string getString ( const string & label, string def ) const

Return parameter corresponding to a given label, when its value is a string. Case where a default value is provided

in	label	Label that identifies the string (read from input file)
in	def	Default value: Value to assign if the sought parameter is not found

### int getInteger ( const string & label ) const

Return parameter corresponding to a given label, when its value is an integer.

#### **Parameters**

in	label	Label that identifies the integer number (read from input file) If this label is not
		found an error message is displayed and program stops

### int getInteger ( const string & label, int def ) const

Return parameter corresponding to a given label, when its value is an integer. Case where a default value is provided

#### Parameters

in	label	Label that identifies the integer number (read from input file).
in	def	Default value: Value to assign if the sought parameter is not found

### real\_t getDouble ( const string & label ) const

Return parameter corresponding to a given label, when its value is a real\_t.

#### Parameters

in	label	Label that identifies the real number (read from input file). If this label is not
		found an error message is displayed and program stops.

### real\_t getDouble ( const string & label, real\_t def ) const

Return parameter corresponding to a given label, when its value is a real\_t. Case where a default value is provided

#### **Parameters**

in	label	Label that identifies the real number (read from input file)
in	def	Default value: Value to assign if the sought parameter is not found

### complex\_t getComplex ( const string & label ) const

Return parameter corresponding to a given label, when its value is a complex number.

in	label	Label that identifies the complex number (read from input file) If this label is not
		found an error message is displayed and program stops

### complex\_t getComplex ( const string & label, complex\_t def ) const

Return parameter corresponding to a given label, when its value is a complex number. Case where a default value is provided

#### **Parameters**

in	label	Label that identifies the complex number (read from input file)
in	def	Default value: Value to assign if the sought parameter is not found

### int contains ( const string & label ) const

check if the project file contains a given parameter

#### **Parameters**

Г	in <i>label</i>	Label that identifies the label to seek in file
---	-----------------	---

#### Returns

0 if the parameter is not found,  $\tt n$  if the parameter is found, where  $\tt n$  is the parameter index in the parameter list

### void get ( const string & label, Vect< real\_t > & a ) const

Read an array of real values, corresponding to a given label.

### Parameters

in	label	Label that identifies the array (read from input file).
in	а	Vector that contain the array. The vector is properly resized before filling.

#### Remarks

If this label is not found an error message is displayed.

### real\_t getArraySize ( const string & label, size\_t j ) const

Return an array entry for a given label.

#### **Parameters**

in	label	Label that identifies the array (read from input file).
in	j	Index of entry in the array (Starting from 1)

### Remarks

If this label is not found an error message is displayed and program stops.

### void get (const string & label, int & a) const

Return integer parameter corresponding to a given label.

#### Parameters

in	label	Label that identifies the integer number (read from input file).	
out	а	Returned value. If this label is not found an error message is displayed and	
		program stops. Note: This member function can be used instead of getInteger	

### void get ( const string & label, real\_t & a ) const

Return real parameter corresponding to a given label.

#### Parameters

in	label	Label that identifies the real (real_t) number (read from input file).
out	а	Returned value. If this label is not found an error message is displayed and
		program stops. Note: This member function can be used instead of getReal_T

### void get (const string & label, complex\_t & a) const

Return complex parameter corresponding to a given label.

#### **Parameters**

in	label	Label that identifies the complex number (read from input file).
out	а	Returned value. If this label is not found an error message is displayed and
		program stops.

### void get ( const string & label, string & a ) const

Return string parameter corresponding to a given label.

#### **Parameters**

j	n	label	Label that identifies the atring (read from input file).	
C	out	а	Returned value. Note: This member function can be used instead of getString If this label is not found an error message is displayed and program stops. Note: This member function can be used instead of getString	

### string getProject ( ) const

Return parameter read using keyword Project.

This parameter can be used to read a project's name.

#### string getMeshFile ( size\_t i = 1 ) const

Return i-th parameter read using keyword mesh\_file.

Here we have at most 10 integer extra parameters that can be used for any purpose. Default value for  $\mathtt{i}$  is 1

#### string getInitFile ( ) const

Return parameter read using keyword InitFile.

This parameter can be used to read an initial data file name.

#### string getRestartFile ( ) const

Return parameter read using keyword RestartFile.

This parameter can be used to read a restart file name.

### string getBCFile ( ) const

Return parameter read using keyword BCFile.

This parameter can be used to read a boundary condition file name.

#### string getBFFile ( ) const

Return parameter read using keyword BFFile.

This parameter can be used to read a body force file name.

#### string getSFFile ( ) const

Return parameter read using keyword SFFile.

This parameter can be used to read a source force file name.

#### string getSaveFile ( ) const

Return parameter read using keyword SaveFile.

This parameter can be used to read a save file name.

### string getPlotFile ( int i = 1 ) const

Return i-th parameter read using keyword PlotFile.

Here we have at most 10 integer extra parameters that can be used for plot file names. Default value for i is 1

#### string getPrescriptionFile ( int i = 1 ) const

Return parameter read using keyword DataFile.

This parameter can be used to read a Prescription file.

#### string getAuxFile ( size\_t i = 1 ) const

Return i-th parameter read using keyword **Auxfile**.

Here we have at most 10 integer extra parameters that can be used for any auxiliary file names. Default value for i is 1

# 7.54 Iter< T\_> Class Template Reference

Class to drive an iterative process.

#### **Public Member Functions**

• Iter ()

Default Constructor.

• Iter (int max\_it, real\_t toler)

Constructor with iteration parameters.

• ~Iter ()

Destructor.

void setMaxIter (int max\_it)

Set maximal number of iterations.

void setTolerance (real\_t toler)

Set tolerance value for convergence.

void setVerbose (int v)

Set verbosity parameter.

• bool check (Vect< T $_->$  &u, const Vect< T $_->$  &v, int opt=2)

Check convergence.

• bool check (T<sub>-</sub> &u, const T<sub>-</sub> &v)

Check convergence for a scalar case (one equation)

### 7.54.1 Detailed Description

```
template<class T_> class OFELI::Iter< T_>
```

Class to drive an iterative process.

This template class enables monitoring any iterative process. It simply sets default values for tolerance, maximal number of iterations and enables checking convergence using two successive iterates.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

#### 7.54.2 Member Function Documentation

```
void setMaxIter ( int max_it )
```

Set maximal number of iterations.

in	max←	Maximal number of iterations [Default: 100]	1
	_it		

#### void setTolerance ( real\_t toler )

Set tolerance value for convergence.

#### **Parameters**

in toler Tol	erance value [Default: 1.e-8]
--------------	-------------------------------

#### void set Verbose ( int v )

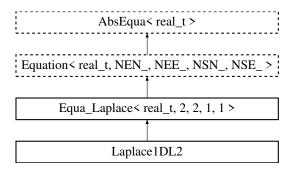
Set verbosity parameter.

#### **Parameters**

	in	v	Verbosity parameter [Default: 0]
--	----	---	----------------------------------

## 7.55 Laplace1DL2 Class Reference

To build element equation for a 1-D elliptic equation using the 2-Node line element  $(P_1)$ . Inheritance diagram for Laplace1DL2:



### **Public Member Functions**

• Laplace1DL2()

Default constructor.

- Laplace1DL2 (Mesh &ms, Vect< real\_t > &u)
- Laplace1DL2 (Mesh &ms)
- ~Laplace1DL2 ()

Destructor.

• void LHS()

Add finite element matrix to left hand side.

• void buildEigen (int opt=0)

Build global stiffness and mass matrices for the eigen system.

• void BodyRHS (const Vect< real\_t > &f)

Add Right-Hand Side Contribution.

void BoundaryRHS (const Vect< real\_t > &f)

Add Neumann contribution to Right-Hand Side.

• void setBoundaryCondition (real\_t f, int lr)

Set Dirichlet boundary data.

• void setTraction (real\_t f, int lr)

Set Traction data.

### 7.55.1 Detailed Description

To build element equation for a 1-D elliptic equation using the 2-Node line element (P<sub>1</sub>).

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

### 7.55.2 Constructor & Destructor Documentation

Laplace1DL2 ( Mesh & ms, Vect< real\_t > & u )

Constructor using mesh instance and solution vector

#### **Parameters**

in	ms	Mesh instance	
in,out	и	Vect instance that contains, after execution of run() the solution	

### Laplace1DL2 ( Mesh & ms )

Constructor using mesh instance

### Parameters

in	ms	Mesh instance

### 7.55.3 Member Function Documentation

**void buildEigen (int** opt = 0) [virtual]

Build global stiffness and mass matrices for the eigen system.

#### Parameters

in	opt	Flag to choose a lumped mass matrix (0) or consistent (1) [Default: 0]
----	-----	--

Reimplemented from Equa\_Laplace< real\_t, 2, 2, 1, 1 >.

 ${f void\ BodyRHS}$  (  ${f const\ Vect}{<\ real\_t} > \&f$  ) [virtual]

Add Right-Hand Side Contribution.

#### **Parameters**

in	f	Vector containing the source given function at mesh nodes	
----	---	---	--

Reimplemented from Equa\_Laplace< real\_t, 2, 2, 1, 1 >.

### ${f void\ Boundary RHS}$ ( ${f const\ Vect}{<{\it real\_t}} > \& f$ ) [virtual]

Add Neumann contribution to Right-Hand Side.

#### **Parameters**

in	f	Vector with size the total number of nodes. The first entry stands for the force at the
		first node (Neumann condition) and the last entry is the force at the last node
		(Neumann condition)

Reimplemented from Equa\_Laplace< real\_t, 2, 2, 1, 1 >.

### void setBoundaryCondition ( real\_t f, int lr )

Set Dirichlet boundary data.

#### Parameters

in	f	Value to assign
in	lr	Option to choose location of the value (-1: Left end, 1: Right end)

### void setTraction ( real\_t f, int lr )

Set Traction data.

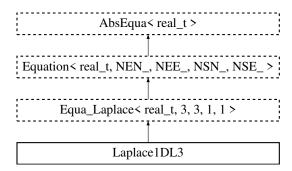
### Parameters

in	f	Value of traction (Neumann boundary condition)	
in	lr	Option to choose location of the traction (-1: Left end, 1: Right end)	

# 7.56 Laplace1DL3 Class Reference

To build element equation for the 1-D elliptic equation using the 3-Node line (P2).

Inheritance diagram for Laplace1DL3:



#### **Public Member Functions**

• Laplace1DL3 ()

Default constructor. Initializes an empty equation.

• Laplace1DL3 (Mesh &ms)

Constructor using mesh instance.

- Laplace1DL3 (Mesh &ms, Vect< real\_t > &u)
- ~Laplace1DL3 ()

Destructor.

• void LHS ()

Compute element matrix.

void BodyRHS (const Vect< real\_t > &f)

Add Right-hand side contribution.

• void BoundaryRHS (const Vect< real\_t > &h)

Add Neumann contribution to Right-Hand Side.

• void setTraction (real\_t f, int lr)

Set Traction data.

### 7.56.1 Detailed Description

To build element equation for the 1-D elliptic equation using the 3-Node line (P<sub>2</sub>).

Author

Rachid Touzani

Copyright

GNU Lesser Public License

### 7.56.2 Constructor & Destructor Documentation

### Laplace1DL3 ( Mesh & ms )

Constructor using mesh instance.

in	ms	Mesh instance

### Laplace1DL3 ( Mesh & ms, Vect< real\_t > & u )

Constructor using mesh instance and solution vector

#### **Parameters**

in	ms	Mesh instance
in,out	и	Vect instance that contains, after execution of run() the solution

### 7.56.3 Member Function Documentation

 ${f void\ BodyRHS}$  (  ${f const\ Vect}{<}\ {f real\_t} > \&f$  ) [virtual]

Add Right-hand side contribution.

#### **Parameters**

in	f	Vector of right-hand side of the Poisson equation at nodes
----	---	--

Reimplemented from Equa\_Laplace< real\_t, 3, 3, 1, 1 >.

### void BoundaryRHS ( const Vect< real\_t> & h ) [virtual]

Add Neumann contribution to Right-Hand Side.

#### **Parameters**

in	h	Vector with size the total number of nodes. The first entry stands for the force at the
		first node (Neumann condition) and the last entry is the force at the last node
		(Neumann condition)

Reimplemented from Equa\_Laplace< real\_t, 3, 3, 1, 1 >.

### void setTraction ( real\_t f, int lr )

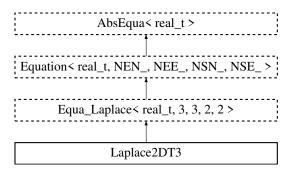
Set Traction data.

#### Parameters

in	f	Value of traction (Neumann boundary condition)
in	lr	Option to choose location of the traction (-1: Left end, 1: Right end)

# 7.57 Laplace2DT3 Class Reference

To build element equation for the Laplace equation using the 2-D triangle element (P<sub>1</sub>). Inheritance diagram for Laplace2DT3:



#### **Public Member Functions**

• Laplace2DT3 ()

Default constructor.

• Laplace2DT3 (Mesh &ms)

Constructor with mesh.

Laplace2DT3 (Mesh &ms, Vect< real\_t > &u)

Constructor using mesh and solution vector.

Laplace2DT3 (Mesh &ms, Vect< real\_t > &b, Vect< real\_t > &Dbc, Vect< real\_t > &Nbc, Vect< real\_t > &u)

Constructor that initializes a standard Poisson equation.

• ∼Laplace2DT3 ()

Destructor.

• void LHS ()

Add finite element matrix to left-hand side.

void BodyRHS (const Vect< real\_t > &f)

Add body source term to right-hand side.

void BoundaryRHS (const Vect< real\_t > &h)

Add boundary source term to right-hand side.

• void buildEigen (int opt=0)

Build global stiffness and mass matrices for the eigen system.

void Post (const Vect< real\_t > &u, Vect< Point< real\_t >> &p)

Perform post calculations.

### 7.57.1 Detailed Description

To build element equation for the Laplace equation using the 2-D triangle element  $(P_1)$ . To build element equation for the Laplace equation using the 3-D tetrahedral element  $(P_1)$ .

Author

Rachid Touzani

Copyright

GNU Lesser Public License

#### 7.57.2 Constructor & Destructor Documentation

Laplace2DT3 ( Mesh & ms )

Constructor with mesh.

#### Parameters

in ms	Mesh instance
-------	---------------

#### Laplace2DT3 ( Mesh & ms, Vect< real\_t > & u )

Constructor using mesh and solution vector.

#### **Parameters**

in	ms	Mesh instance
in	и	Problem right-hand side

# Laplace2DT3 ( Mesh & ms, Vect< real\_t > & b, Vect< real\_t > & Dbc, Vect< real\_t > & Nbc, Vect< real\_t > & u)

Constructor that initializes a standard Poisson equation.

This constructor sets data for the Poisson equation with mixed (Dirichlet and Neumann) boundary conditions.

#### Parameters

in	ms	Mesh instance
in	b	Vector containing the source term (right-hand side of the equation) at mesh nodes
in	Dbc	Vector containing prescribed values of the solution (Dirichlet boundary condition) at nodes with positive code. Its size is the total number of nodes
in	Nbc	Vector containing prescribed fluxes (Neumann boundary conditions) at sides, its size is the total number of sides
in	и	Vector to contain the finite element solution at nodes once the member function run() is called.

### 7.57.3 Member Function Documentation

void BodyRHS ( const Vect < real t > & f ) [virtual]

Add body source term to right-hand side.

#### Parameters

in	f	Vector containing the source given function at mesh nodes
----	---	---

Reimplemented from Equa\_Laplace< real\_t, 3, 3, 2, 2 >.

void BoundaryRHS ( const Vect< real\_t> & h ) [virtual]

Add boundary source term to right-hand side.

#### **Parameters**

in	h	Vector containing the source given function at mesh nodes	
----	---	---	--

Reimplemented from Equa\_Laplace< real\_t, 3, 3, 2, 2 >.

### **void buildEigen ( int** opt = 0 **)** [virtual]

Build global stiffness and mass matrices for the eigen system.

#### **Parameters**

in	opt	Flag to choose a lumped mass matrix (0) or consistent (1) [Default: 0]
----	-----	--

Reimplemented from Equa\_Laplace< real\_t, 3, 3, 2, 2 >.

void Post ( const Vect< real\_t > & u, Vect< Point< real\_t > > & p)

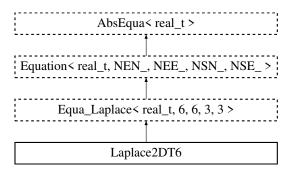
Perform post calculations.

#### **Parameters**

in	и	Solution at nodes	
out	p	Vector containing gradient at elements	

# 7.58 Laplace2DT6 Class Reference

To build element equation for the Laplace equation using the 2-D triangle element ( $P_2$ ). Inheritance diagram for Laplace2DT6:



### **Public Member Functions**

• Laplace2DT6 ()

Default constructor.

• Laplace2DT6 (Mesh &ms)

Constructor with mesh.

• Laplace2DT6 (Mesh &ms, Vect< real\_t > &u)

Constructor using mesh and solution vector.

• ~Laplace2DT6 ()

Destructor.

• void LHS ()

Add finite element matrix to left-hand side.

void BodyRHS (const Vect< real\_t > &f)

Add body source term to right-hand side.

• void BoundaryRHS (const Vect< real\_t > &h)

Add boundary source term to right-hand side.

• void buildEigen (int opt=0)

Build global stiffness and mass matrices for the eigen system.

### 7.58.1 Detailed Description

To build element equation for the Laplace equation using the 2-D triangle element (P2).

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

#### 7.58.2 Constructor & Destructor Documentation

#### Laplace2DT6 ( Mesh & ms )

Constructor with mesh.

**Parameters** 

in	ms	Mesh instance

### Laplace2DT6 ( Mesh & ms, Vect< real\_t > & u )

Constructor using mesh and solution vector.

### Parameters

in	ms	Mesh instance
in	и	Problem right-hand side

### 7.58.3 Member Function Documentation

 ${f void\ BodyRHS}$  (  ${f const\ Vect}{<{\it real\_t}} > \& f$  ) [virtual]

Add body source term to right-hand side.

in	f	Vector containing the source given function at mesh nodes

Reimplemented from Equa\_Laplace< real\_t, 6, 6, 3, 3 >.

### void BoundaryRHS ( const Vect< real\_t> & h ) [virtual]

Add boundary source term to right-hand side.

#### **Parameters**

in	h	Vector containing the source given function at mesh nodes	
----	---	---	--

Reimplemented from Equa\_Laplace< real\_t, 6, 6, 3, 3 >.

### void buildEigen ( int opt = 0 ) [virtual]

Build global stiffness and mass matrices for the eigen system.

#### **Parameters**

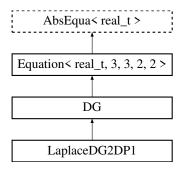
in	opt	Flag to choose a lumed mass matrix (0) or consistent (1) [Default: 0]
----	-----	---

Reimplemented from Equa\_Laplace< real\_t, 6, 6, 3, 3 >.

## 7.59 LaplaceDG2DP1 Class Reference

To build and solve the linear system for the Poisson problem using the  $\overline{DG}$   $P_1$  2-D triangle element.

Inheritance diagram for LaplaceDG2DP1:



#### **Public Member Functions**

LaplaceDG2DP1 (Mesh &ms, Vect< real\_t > &f, Vect< real\_t > &Dbc, Vect< real\_t > &Nbc, Vect< real\_t > &u)

Constructor with mesh and vector data.

• ~LaplaceDG2DP1 ()

Destructor.

• void set (real\_t sigma, real\_t eps)

*Set parameters for the* **DG** *method.* 

• void set (const LocalMatrix < real\_t, 2, 2 > &K)

Set diffusivity matrix.

• void build ()

Build global matrix and right-hand side.

• void Smooth (Vect< real\_t > &u)

Perform post calculations.

• int run ()

Build and solve the linear system of equations using an iterative method.

### 7.59.1 Detailed Description

To build and solve the linear system for the Poisson problem using the  $\overline{DG}$   $P_1$  2-D triangle element

This class build the linear system of equations for a standard elliptic equation using the Discontinuous Galerkin  $P_1$  finite element method.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

#### 7.59.2 Constructor & Destructor Documentation

LaplaceDG2DP1 ( Mesh & ms, Vect< real\_t > & f, Vect< real\_t > & Dbc, Vect< real\_t > & Nbc, Vect< real\_t > & u )

Constructor with mesh and vector data.

#### **Parameters**

in	ms	Mesh instance	
in	f	Vector containing the right-hand side of the elliptic equation at triangle vertices	
in	Dbc	Vector containing prescribed values of the solution (Dirichlet boundary condition) at nodes having a positive code	
in	Nbc	Vector containing prescribed values of the flux (Neumann boundary condition) at each side having a positive code	
in	и	Vector where the solution is stored once the linear system is solved	

#### 7.59.3 Member Function Documentation

void set ( real\_t sigma, real\_t eps )

Set parameters for the DG method.

in	sigma	Penalty parameters to enforce continuity at nodes (Must be positive) [Default:
		100]

#### **Parameters**

in	eps	Epsilon value of the DG method to choose among the values:	
		0 Incomplete Interior Penalty Galerkin method (IIPG)	
		-1 Symmetric Interior Penalty Galerkin method (SIPG)	
		1 Non symmetric interior penalty Galerkin method (NIPG)	
		For a user not familiar with the method, please choose the value of eps=-1 and sigma>100 which leads to a symmetric positive definite matrix [Default: -1]	

### void set ( const LocalMatrix< real\_t, 2, 2 > & K )

Set diffusivity matrix.

This function provides the diffusivity matrix as instance of class LocalMatrix. The default diffusivity matrix is the identity matrix

#### Parameters

in	K	Diffusivity matrix
----	---	--------------------

#### void build ( )

Build global matrix and right-hand side.

The problem matrix and right-hand side are the ones used in the constructor. They are updated in this member function.

#### void Smooth ( Vect< real\_t > & u )

Perform post calculations.

This function gives an averaged solution given at mesh nodes (triangle vertices) by a standard  $L_2$ -projection method.

### Parameters

in	и	Solution at nodes

### int run ( )

Build and solve the linear system of equations using an iterative method.

The matrix is preconditioned by the diagonal ILU method. The linear system is solved either by the Conjugate Gradient method if the matrix is symmetric positive definite (eps=-1) or the GMRES method if not. The solution is stored in the vector u given in the constructor.

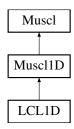
### Returns

Number of performed iterations. Note that the maximal number is 1000 and the tolerance is 1.e-8

### 7.60 LCL1D Class Reference

Class to solve the linear conservation law (Hyperbolic equation) in 1-D by a MUSCL Finite Volume scheme.

Inheritance diagram for LCL1D:



#### **Public Member Functions**

• LCL1D (Mesh &m)

Constructor using mesh instance.

LCL1D (Mesh &m, Vect< real\_t > &U)

Constructor.

• ~LCL1D ()

Destructor.

• Vect< real\_t > & getFlux ()

Return sidewise fluxes.

• void setInitialCondition (Vect< real\_t > &u)

Assign initial condition by a vector.

void setInitialCondition (real\_t u)

Assign a constant initial condition.

• void setReconstruction ()

Run MUSCL reconstruction.

real\_t runOneTimeStep ()

Run one time step of the linear conservation law.

void setBC (real\_t u)

Set Dirichlet boundary condition.

• void setBC (const Side &sd, real\_t u)

Set Dirichlet boundary condition.

• void setBC (int code, real\_t u)

Set Dirichlet boundary condition.

void setVelocity (Vect< real\_t > &v)

Set convection velocity.

void setVelocity (real\_t v)

Set (constant) convection velocity.

• void setReferenceLength (real\_t dx)

Assign reference length value.

real\_t getReferenceLength () const

Return reference length.

• void Forward (const Vect< real\_t > &Flux, Vect< real\_t > &Field)

Computation of the primal variable n->n+1.

### **Additional Inherited Members**

### 7.60.1 Detailed Description

Class to solve the linear conservation law (Hyperbolic equation) in 1-D by a MUSCL Finite Volume scheme.

Author

S. Clain, V. Clauzon

Copyright

**GNU Lesser Public License** 

### 7.60.2 Member Function Documentation

### void setInitialCondition ( Vect< real\_t > & u )

Assign initial condition by a vector.

#### **Parameters**

	in	и	Vector containing initial condition
--	----	---	-------------------------------------

#### void setInitialCondition ( real\_t u )

Assign a constant initial condition.

#### Parameters

in	и	Constant value for the initial condition
----	---	--

### real\_t runOneTimeStep ( )

Run one time step of the linear conservation law.

Returns

Value of the time step

#### void setBC ( real\_t u )

Set Dirichlet boundary condition.

Assign a constant value u to all boundary sides

### void setBC ( const Side & sd, real\_t u )

Set Dirichlet boundary condition.
Assign a constant value to a side

in	sd	Side to which value is prescibed
in	и	Value to prescribe

## void setBC ( int code, real\_t u )

Set Dirichlet boundary condition.

Assign a constant value sides with a given code

#### **Parameters**

in	code	Code of sides to which value is prescibed
in	и	Value to prescribe

## void setVelocity ( Vect< real\_t > & v )

Set convection velocity.

#### **Parameters**

in	v	Vect instance containing velocity
----	---	-----------------------------------

#### void Forward ( const Vect< real\_t > & Flux, Vect< real\_t > & Field )

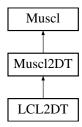
Computation of the primal variable n-n+1.

Vector Flux contains elementwise fluxes issued from the Riemann problem, calculated with, as left element, getNeighborElement(1) and right element getNeighborElement(2) if getNeighbor← Element(2) doesn't exist, we are on a boundary and we prescribe a symmetry condition

## 7.61 LCL2DT Class Reference

Class to solve the linear hyperbolic equation in 2-D by a MUSCL Finite Volume scheme on triangles.

Inheritance diagram for LCL2DT:



## **Public Member Functions**

• LCL2DT (Mesh &m)

Constructor using Mesh instance.

• LCL2DT (Mesh &m, Vect < real\_t > &U)

Constructor using mesh and initial data.

• ~LCL2DT ()

Destructor.

Vect< real\_t > & getFlux ()

Return sidewise flux vector.

• void setInitialCondition (Vect< real\_t > &u)

Set elementwise initial condition.

void setInitialCondition (real\_t u)

*Set a constant initial condition.* 

• void setReconstruction ()

Reconstruct flux using Muscl scheme.

• real\_t runOneTimeStep ()

Run one time step of the linear conservation law.

• void setBC (real\_t u)

Set Dirichlet boundary condition.

• void setBC (const Side &sd, real\_t u)

Set Dirichlet boundary condition.

• void setBC (int code, real\_t u)

Set Dirichlet boundary condition.

• void setVelocity (const Vect< real\_t > &v)

*Set convection velocity.* 

• void setVelocity (const LocalVect< real\_t, 2 > &v)

Set (constant) convection velocity.

void Forward (const Vect< real\_t > &Flux, Vect< real\_t > &Field)

Computation of the primal variable n->n+1.

### **Additional Inherited Members**

## 7.61.1 Detailed Description

Class to solve the linear hyperbolic equation in 2-D by a MUSCL Finite Volume scheme on triangles.

Author

S. Clain, V. Clauzon

Copyright

**GNU Lesser Public License** 

## 7.61.2 Constructor & Destructor Documentation

LCL2DT ( Mesh & m, Vect< real\_t > & U )

Constructor using mesh and initial data.

in	m	Reference to Mesh instance
in	U	Vector containing initial (elementwise) solution

## 7.61.3 Member Function Documentation

## void setInitialCondition ( Vect< real\_t > & u )

Set elementwise initial condition.

#### **Parameters**

in	и	Vect instance containing initial condition values
----	---	---

## void setInitialCondition ( real\_t u )

Set a constant initial condition.

#### **Parameters**

in	и	Value of initial condition to assign to all elements
----	---	--

## real\_t runOneTimeStep ( )

Run one time step of the linear conservation law.

#### Returns

Value of the time step

## void setBC ( real\_t u )

Set Dirichlet boundary condition.
Assign a constant value u to all boundary sides

## void setBC ( const Side & sd, real\_t u )

Set Dirichlet boundary condition.
Assign a constant value to a side

#### Parameters

in	sd	Side to which value is prescibed
in	и	Value to prescribe

## void setBC ( int code, real\_t u )

Set Dirichlet boundary condition.

Assign a constant value sides with a given code

#### **Parameters**

in	code	Code of sides to which value is prescibed
in	и	Value to prescribe

## void setVelocity ( const Vect< real\_t > & v )

Set convection velocity.

#### **Parameters**

	in	v	Vect instance containing velocity
--	----	---	-----------------------------------

## void setVelocity ( const LocalVect< real\_t, 2 > &v )

Set (constant) convection velocity.

#### **Parameters**

in	v	Vector containing constant velocity to prescribe
----	---	--

## void Forward ( const Vect< real\_t > & Flux, Vect< real\_t > & Field )

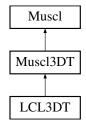
Computation of the primal variable n->n+1.

Vector *Flux* contains elementwise fluxes issued from the Riemann problem, calculated with, as left element, **getNeighborElement(1)** and right element **getNeighborElement(2)** if **getNeighbor**← **Element(2)** doesn't exist, we are on a boundary and we prescribe a symmetry condition

## 7.62 LCL3DT Class Reference

Class to solve the linear conservation law equation in 3-D by a MUSCL Finite Volume scheme on tetrahedra.

Inheritance diagram for LCL3DT:



## **Public Member Functions**

• LCL3DT (Mesh &m)

Constructor using mesh.

• LCL3DT (Mesh &m, Vect < real\_t > &U)

Constructor using mesh and initial field.

• ~LCL3DT ()

Destructor.

void setInitialCondition (Vect< real\_t > &u)

Set elementwise initial condition.

• void setInitialCondition (real\_t u)

Set a constant initial condition.

• void setReconstruction ()

Reconstruct flux using Muscl scheme.

real\_t runOneTimeStep ()

Run one time step.

• void setBC (real\_t u)

Set Dirichlet boundary condition. Assign a constant value u to all boundary sides.

• void setBC (const Side &sd, real\_t u)

Set Dirichlet boundary condition.

• void setBC (int code, real\_t u)

Set Dirichlet boundary condition.

• void setVelocity (const Vect< real\_t > &v)

Set convection velocity.

• void setVelocity (const LocalVect< real\_t, 3 > &v)

Set (constant) convection velocity.

void setReferenceLength (real\_t dx)

Assign reference length value.

real\_t getReferenceLength () const

Return reference length.

void Forward (const Vect< real\_t > &Flux, Vect< real\_t > &Field)

Computation of the primal variable n->n+1.

#### **Additional Inherited Members**

## 7.62.1 Detailed Description

Class to solve the linear conservation law equation in 3-D by a MUSCL Finite Volume scheme on tetrahedra.

Author

S. Clain, V. Clauzon

Copyright

**GNU Lesser Public License** 

### 7.62.2 Constructor & Destructor Documentation

LCL3DT ( Mesh &  $m_t$  Vect< real\_t > & U )

Constructor using mesh and initial field.

in	т	Reference to Mesh instance
in	U	Vector containing initial (elementwise) solution

## 7.62.3 Member Function Documentation

## void setInitialCondition ( Vect< real\_t > & u )

Set elementwise initial condition.

#### **Parameters**

in	и	Vect instance containing initial condition values
----	---	---

## void setInitialCondition ( real\_t u )

Set a constant initial condition.

#### **Parameters**

in	и	Value of initial condition to assign to all elements
----	---	--

## void setBC ( const Side & sd, real\_t u )

Set Dirichlet boundary condition.
Assign a constant value to a side

#### **Parameters**

in	sd	Side to which value is prescibed
in	и	Value to prescribe

## void setBC ( int code, real\_t u )

Set Dirichlet boundary condition.
Assign a constant value sides with a given code

#### **Parameters**

in	code	Code of sides to which value is prescibed
in	и	Value to prescribe

## void setVelocity ( const Vect< real\_t > & v )

Set convection velocity.

in	v	Vect instance containing velocity
----	---	-----------------------------------

## void setVelocity ( const LocalVect< real\_t, 3 > & v )

Set (constant) convection velocity.

#### **Parameters**

in	v	Vector containing constant velocity to prescribe
----	---	--

#### void Forward (const Vect< real\_t > & Flux, Vect< real\_t > & Field)

Computation of the primal variable n-n+1.

Vector Flux contains elementwise fluxes issued from the Riemann problem, calculated with, as left element, **getNeighborElement(1)** and right element **getNeighborElement(2)** if **getNeighbor**← **Element(2)** doesn't exist, we are on a boundary and we prescribe a symmetry condition

## 7.63 Line2 Class Reference

To describe a 2-Node planar line finite element. Inheritance diagram for Line2:



## **Public Member Functions**

- Line2()
  - Default Constructor.
- Line2 (const Element \*el)

Constructor for an element.

- Line2 (const Side \*side)
  - Constructor for a side.
- Line2 (const Edge \*edge)
  - Constructor for an edge.
- ~Line2 ()
  - Destructor.
- real\_t getLength () const

Return element length.

- Point< real\_t > getNormal () const
  - Return unit normal vector to line.
- Point< real\_t > getTangent () const

Return unit tangent vector to line.

• real\_t Sh (size\_t i, real\_t s) const

Calculate shape function of a given node at a given point.

• std::vector< Point< real\_t >> DSh () const

Return partial derivatives of shape functions of element nodes.

Point< real\_t > getRefCoord (const Point< real\_t > &x)

Return reference coordinates of a point x in element.

• bool isIn (const Point < real\_t > &x)

Check whether point x is in current line element or not.

real\_t getInterpolate (const Point< real\_t > &x, const LocalVect< real\_t, 2 > &v)

Return interpolated value at a given point.

## 7.63.1 Detailed Description

To describe a 2-Node planar line finite element.

Defines geometric quantities associated to 2-node linear segment element  $P_1$  in the space. The reference element is the segment [-1,1]. Note that the line length is not checked unless the function check is called.

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

## 7.63.2 Constructor & Destructor Documentation

Line2 (const Element \* el)

Constructor for an element.

**Parameters** 

i	n	el	Pointer to element

## Line2 ( const Side \* side )

Constructor for a side.

**Parameters** 

in	side	Pointer to side

Line2 (const Edge \* edge )

Constructor for an edge.

#### Parameters

in	edge	Pointer to edge
----	------	-----------------

#### 7.63.3 Member Function Documentation

## real\_t Sh ( size\_t i, real\_t s ) const

Calculate shape function of a given node at a given point.

#### **Parameters**

in	i	Node number (1 or 2).
in	S	Localization of point in natural coordinates (must be between -1 and 1).

## $std::vector < Point < real_t > > DSh ( ) const$

Return partial derivatives of shape functions of element nodes.

#### Returns

LocalVect instance of partial derivatives of shape functions e.g. dsh(i).x, dsh(i).y, are partial derivatives of the i-th shape function.

## Point<real\_t> getRefCoord ( const Point< real\_t> & x )

Return reference coordinates of a point x in element.

Only the x-coordinate of the returned value has a meaning

## real\_t getInterpolate ( const Point< real\_t > & x, const LocalVect< real\_t, 2 > & v)

Return interpolated value at a given point.

#### **Parameters**

in	x	Point where interpolation is evaluated (in the reference element).
out	v	Computed value.

## 7.64 Line3 Class Reference

To describe a 3-Node quadratic planar line finite element.

Inheritance diagram for Line3:



## **Public Member Functions**

• Line3()

Default Constructor.

• Line3 (const Element \*el)

Constructor for an element.

Line3 (const Side \*sd)

Constructor for a side.

• ~Line3 ()

Destructor.

• void setLocal (real\_t s)

*Initialize local point coordinates in element.* 

• LocalVect< Point< real\_t >, 3 > DSh () const

Return partial derivatives of shape functions of element nodes.

Point< real\_t > getLocalPoint () const

Return actual coordinates of localized point.

## 7.64.1 Detailed Description

To describe a 3-Node quadratic planar line finite element.

Defines geometric quantities associated to 3-node quadratic element  $P_2$  in the space. The reference element is the segment [-1,1]. The user must take care to the fact that determinant of jacobian and other quantities depend on the point in the reference element where they are calculated. For this, before any utilization of shape functions or jacobian, function **setLocal()** must be invoked.

Element nodes are ordered as the following: the left one, the central one and the right one.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

#### 7.64.2 Member Function Documentation

LocalVect<Point<real\_t>,3> DSh ( ) const

Return partial derivatives of shape functions of element nodes.

Returns

LocalVect instance of partial derivatives of shape functions e.g. dsh(i).x, dsh(i).y, are partial derivatives of the i-th shape function.

Note

The local point at which the derivatives are computed must be chosen before by using the member function setLocal

## 7.65 LinearSolver< T $_->$ Class Template Reference

Class to solve systems of linear equations by iterative methods.

## **Public Member Functions**

• LinearSolver ()

Default Constructor.

• LinearSolver (int max\_it, real\_t tolerance)

Constructor with iteration parameters.

• LinearSolver (SpMatrix <  $T_- > &A$ , const Vect <  $T_- > &b$ , Vect <  $T_- > &x)$ 

Constructor using matrix, right-hand side and solution vector.

• LinearSolver (SkMatrix <  $T_- > &A$ , const Vect <  $T_- > &b$ , Vect <  $T_- > &x)$ 

Constructor using skyline-stored matrix, right-hand side and solution vector.

• LinearSolver (TrMatrix  $< T_- > &A$ , const Vect  $< T_- > &b$ , Vect  $< T_- > &x$ )

Constructor using a tridiagonal matrix, right-hand side and solution vector.

• LinearSolver (BMatrix < T $_-$  > &A, const Vect < T $_-$  > &b, Vect < T $_-$  > &x)

Constructor using a banded matrix, right-hand side and solution vector.

• LinearSolver (DMatrix< T $_->$  &A, const Vect< T $_->$  &b, Vect< T $_->$  &x)

Constructor using a dense matrix, right-hand side and solution vector.

• LinearSolver (DSMatrix $< T_- > &A$ , const Vect $< T_- > &b$ , Vect $< T_- > &x$ )

Constructor using a dense symmetric matrix, right-hand side and solution vector.

• LinearSolver (SkSMatrix  $< T_- > &A$ , const Vect  $< T_- > &b$ , Vect  $< T_- > &x$ )

Constructor using skyline-stored symmetric matrix, right-hand side and solution vector.

• LinearSolver (SkMatrix  $< T_- > &A$ , Vect  $< T_- > &b$ , Vect  $< T_- > &x$ )

Constructor using matrix, right-hand side.

• virtual ~LinearSolver ()

Destructor.

• void setMaxIter (int m)

Set Maximum number of iterations.

• void setTolerance (real\_t tol)

Set tolerance value.

void setSolution (Vect< T<sub>−</sub> > &x)

Set solution vector.

• void setRHS (Vect< T\_> &b)

Set right-hand side vector.

void setMatrix (OFELI::Matrix < T<sub>-</sub> > \*A)

Set matrix in the case of a pointer to Matrix.

• void setMatrix (SpMatrix < T<sub>−</sub> > &A)

*Set matrix in the case of a pointer to matrix.* 

void setMatrix (SkMatrix < T<sub>-</sub> > &A)

Set matrix in the case of a skyline matrix.

• void set (SpMatrix  $< T_- > &A$ , const Vect  $< T_- > &b$ , Vect  $< T_- > &x$ )

Set matrix, right-hand side and initial guess.

void setSolver (Iteration s, Preconditioner p=DIAG\_PREC)

Set solver and preconditioner.

• Iteration getSolver () const

Return solver code.

• Preconditioner getPreconditioner () const

Return solver preconditioner.

int solve (SpMatrix < T<sub>-</sub> > &A, const Vect < T<sub>-</sub> > &b, Vect < T<sub>-</sub> > &x, Iteration s, Preconditioner p=DIAG\_PREC)

Solve equations using system data, prescribed solver and preconditioner.

• int solve (Iteration s, Preconditioner p=DIAG\_PREC)

Solve equations using prescribed solver and preconditioner.

• int solve ()

Solve equations all arguments must have been given by other member functions.

• void setFact ()

Factorize matrix.

• void setNoFact ()

Do not factorize matrix.

• int getNbIter () const

Get number of performed iterations.

## 7.65.1 Detailed Description

template < class T\_> class OFELI::LinearSolver < T\_>

Class to solve systems of linear equations by iterative methods.

## 7.65.2 Constructor & Destructor Documentation

## LinearSolver ( )

Default Constructor.

Initializes default parameters and pointers to 0.

#### LinearSolver ( int max\_it, real\_t tolerance )

Constructor with iteration parameters.

#### Parameters

j	in	max_it	Maximal number of iterations	
j	in	tolerance	Tolerance for convergence (measured in relative weighted 2-Norm) in input,	
			effective discrepancy in output.	

#### Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

## LinearSolver (SpMatrix $< T_- > & A$ , const Vect $< T_- > & b$ , Vect $< T_- > & x$ )

Constructor using matrix, right-hand side and solution vector.

in	A	Reference to instance of class SpMatrix	
in	b	Vect instance that contains the right-hand side	
in,out	x	Vect instance that contains initial guess on input and solution on output	

## LinearSolver (SkMatrix< T $_->$ & A, const Vect< T $_->$ & b, Vect< T $_->$ & x)

Constructor using skyline-stored matrix, right-hand side and solution vector.

#### **Parameters**

in	A	SkMatrix instance that contains matrix
in	b	Vect instance that contains the right-hand side
in,out	х	Vect instance that contains initial guess on input and solution on output

## LinearSolver ( TrMatrix< T $_->$ & A, const Vect< T $_->$ & b, Vect< T $_->$ & x)

Constructor using a tridiagonal matrix, right-hand side and solution vector.

#### **Parameters**

in	A	TrMatrix instance that contains matrix	
in	b	Vect instance that contains the right-hand side	
in,out	х	Vect instance that contains initial guess on input and solution on output	

## LinearSolver ( BMatrix< T $_->$ & A, const Vect< T $_->$ & b, Vect< T $_->$ & x)

Constructor using a banded matrix, right-hand side and solution vector.

## Parameters

in	A	BMatrix instance that contains matrix	
in	b	Vect instance that contains the right-hand side	
in,out	x	Vect instance that contains initial guess on input and solution on output	

## LinearSolver ( DMatrix $< T_- > & A$ , const Vect $< T_- > & b$ , Vect $< T_- > & x$ )

Constructor using a dense matrix, right-hand side and solution vector.

#### **Parameters**

in	A	DMatrix instance that contains matrix
in	b	Vect instance that contains the right-hand side
in,out	x	Vect instance that contains initial guess on input and solution on output

## LinearSolver ( DSMatrix $< T_- > & A$ , const Vect $< T_- > & b$ , Vect $< T_- > & x$ )

Constructor using a dense symmetric matrix, right-hand side and solution vector.

#### **Parameters**

in	A	DSMatrix instance that contains matrix	
in	b	Vect instance that contains the right-hand side	
in,out	х	Vect instance that contains initial guess on input and solution on output	

## LinearSolver (SkSMatrix $< T_- > & A$ , const Vect $< T_- > & b$ , Vect $< T_- > & x$ )

Constructor using skyline-stored symmetric matrix, right-hand side and solution vector.

#### **Parameters**

in	A	SkMatrix instance that contains matrix	
in	b	Vect instance that contains the right-hand side	
in,out	х	Vect instance that contains initial guess on input and solution on output	

## LinearSolver ( SkMatrix< $T_-$ > & A, Vect< $T_-$ > & b, Vect< $T_-$ > & x)

Constructor using matrix, right-hand side.

## Parameters

in	A	SkMatrix instance that contains matrix
in	b	Vect instance that contains the right-hand side
in,out	x	Vect instance that contains the initial guess on input and solution on output

## 7.65.3 Member Function Documentation

## void setMaxIter ( int m )

Set Maximum number of iterations. Default value is 1000

## void setMatrix ( OFELI::Matrix $< T_- > *A$ )

Set matrix in the case of a pointer to Matrix.

#### Parameters

in	A	Pointer to abstract Matrix class
----	---	----------------------------------

**OFELI's Reference Guide** 

## CHAPTER 7. CLASS DOCUMENTE ATION ARSOLVER $< T_{-} > CLASS$ TEMPLATE REFERENCE

## void setMatrix ( SpMatrix $< T_- > & A$ )

Set matrix in the case of a pointer to matrix.

#### **Parameters**

in	Α	Pointer to abstract Matrix class
----	---	----------------------------------

## void setMatrix ( SkMatrix $< T_- > & A$ )

Set matrix in the case of a skyline matrix.

#### **Parameters**

in	Α	Matrix as instance of class SkMatrix
----	---	--------------------------------------

## void set ( SpMatrix< $T_-$ > & A, const Vect< $T_-$ > & b, Vect< $T_-$ > & x)

Set matrix, right-hand side and initial guess.

#### Parameters

in	A	Reference to matrix as a SpMatrix instance	
in	b	Vector containing right-hand side	
in,out	х	Vector containing initial guess on input and solution on output	

## void setSolver ( Iteration s, Preconditioner $p = DIAG_PREC$ )

Set solver and preconditioner.

## Parameters

in	S	Solver identification parameter. To be chosen in the enumeration variable Iteration: DIRECT_SOLVER, CG_SOLVER, CGS_SOLVER, BICG_SOLVER, BICG_STAB_SOLVER, GMRES_SOLVER, QMR_SOLVER
in	р	Preconditioner identification parameter. By default, the diagonal preconditioner is used. To be chosen in the enumeration variable Preconditioner:  IDENT_PREC, DIAG_PREC, SSOR_PREC, ILU_PREC [Default: ILU_PREC]

#### Note

The argument p has no effect if the solver is DIRECT\_SOLVER

# int solve ( SpMatrix< $T_-$ > & A, const Vect< $T_-$ > & b, Vect< $T_-$ > & x, Iteration s, Preconditioner p = DIAG\_PREC )

Solve equations using system data, prescribed solver and preconditioner.

in	A	Reference to matrix as a SpMatrix instance
in	b	Vector containing right-hand side
in,out	х	Vector containing initial guess on input and solution on output
in	s	Solver identification parameter To be chosen in the enumeration variable Iteration:  DIRECT_SOLVER, CG_SOLVER, CGS_SOLVER, BICG_SOLVER, BICG_STAB_SOLVER, GMRES_SOLVER, QMR_SOLVER [Default: CGS_SOLVER]
in	р	Preconditioner identification parameter. To be chosen in the enumeration variable Preconditioner:  IDENT_PREC, DIAG_PREC, SSOR_PREC, ILU_PREC, DILU_PREC [Default: DIAG_PREC]

#### Remarks

The argument p has no effect if the solver is DIRECT\_SOLVER

## Warning

If the library eigen is used, only the preconditioners IDENT\_PREC, DIAG\_PREC and ILU\_PREC are available.

## int solve ( Iteration s, Preconditioner $p = DIAG_PREC$ )

Solve equations using prescribed solver and preconditioner.

## Parameters

in	S	Solver identification parameter To be chosen in the enumeration variable Iteration: DIRECT_SOLVER, CG_SOLVER, CGS_SOLVER, BICG_SOLVER, BICG_STAB_SOLVER, GMRES_SOLVER, QMR_SOLVER [Default: CGS_SOLVER]
in	р	Preconditioner identification parameter. To be chosen in the enumeration variable Preconditioner:  IDENT_PREC, DIAG_PREC, SSOR_PREC, DILU_PREC, ILU_PREC [Default: DIAG_PREC]

## Note

The argument p has no effect if the solver is DIRECT\_SOLVER

## int solve ( )

Solve equations all arguments must have been given by other member functions.

Solver and preconditioner parameters must have been set by function setSolver. Otherwise, default values are set.

# 7.66 LocalMatrix< T\_, NR\_, NC\_ > Class Template Reference

Handles small size matrices like element matrices, with a priori known size.

## **Public Member Functions**

```
• LocalMatrix ()
```

Default constructor.

• LocalMatrix (const LocalMatrix < T\_, NR\_, NC\_ > &m)

Copy constructor.

• LocalMatrix (Element \*el, const SpMatrix < T\_ > &a)

Constructor of a local matrix associated to element from a SpMatrix.

• LocalMatrix (Element \*el, const SkMatrix < T<sub>-</sub> > &a)

Constructor of a local matrix associated to element from a SkMatrix.

• LocalMatrix (Element \*el, const SkSMatrix < T\_ > &a)

Constructor of a local matrix associated to element from a SkSMatrix.

• ~LocalMatrix ()

Destructor.

• T<sub>-</sub> & operator() (size\_t i, size\_t j)

Operator () (Non constant version)

• T\_ operator() (size\_t i, size\_t j) const

Operator () (Constant version)

void Localize (Element \*el, const SpMatrix< T<sub>-</sub> > &a)

*Initialize matrix as element matrix from global SpMatrix.* 

• void Localize (Element \*el, const SkMatrix < T\_> &a)

*Initialize matrix as element matrix from global SkMatrix.* 

void Localize (Element \*el, const SkSMatrix< T<sub>−</sub> > &a)

*Initialize matrix as element matrix from global SkSMatrix.* 

- LocalMatrix < T\_, NR\_, NC\_ > & operator = (const LocalMatrix < T\_, NR\_, NC\_ > &m)
   Operator =
- LocalMatrix < T\_, NR\_, NC\_ > & operator= (const T\_ &x)
   Overator =
- LocalMatrix < T\_, NR\_, NC\_ > & operator += (const LocalMatrix < T\_, NR\_, NC\_ > &m)
   Operator +=
- LocalMatrix < T\_, NR\_, NC\_ > & operator = (const LocalMatrix < T\_, NR\_, NC\_ > &m)
   Operator =
- LocalVect< T\_, NR\_ > operator\* (LocalVect< T\_, NC\_ > &x)
   Operator \*

• LocalMatrix< T\_, NR\_, NC\_ > & operator+= (const T\_ &x)

Operator +=

• LocalMatrix< T\_, NR\_, NC\_> & operator== (const T\_ &x)

Operator -=

• LocalMatrix< T\_, NR\_, NC\_> & operator\*= (const T\_ &x)

Operator \*=

LocalMatrix< T\_, NR\_, NC\_ > & operator/= (const T\_ &x)

Operator /=

void MultAdd (const LocalVect< T\_, NC\_ > &x, LocalVect< T\_, NR\_ > &y)

Multiply matrix by vector and add result to vector.

- void MultAddScal (const T<sub>-</sub> &a, const LocalVect< T<sub>-</sub>, NC<sub>-</sub> > &x, LocalVect< T<sub>-</sub>, NR<sub>-</sub> > &y)
   Multiply matrix by scaled vector and add result to vector.
- void Mult (const LocalVect< T\_, NC\_> &x, LocalVect< T\_, NR\_> &y)

Multiply matrix by vector.

• void Symmetrize ()

Symmetrize matrix.

• int Factor ()

Factorize matrix.

• int solve (LocalVect< T\_, NR\_> &b)

Forward and backsubstitute to solve a linear system.

• int FactorAndSolve (LocalVect< T\_, NR\_> &b)

Factorize matrix and solve linear system.

void Invert (LocalMatrix < T\_, NR\_, NC\_ > &A)

Calculate inverse of matrix.

- T\_ getInnerProduct (const LocalVect< T\_, NC\_ > &x, const LocalVect< T\_, NR\_ > &y)

  Calculate inner product with respect to matrix.
- T\_\* get ()

Return pointer to matrix as a C-array.

## 7.66.1 Detailed Description

```
template<class T_, size_t NR_, size_t NC_> class OFELI::LocalMatrix< T_, NR_, NC_>
```

Handles small size matrices like element matrices, with a priori known size.

The template class LocalMatrix treats small size matrices. Typically, this class is recommended to store element and side arrays.

Internally, no dynamic storage is used.

## **Template Parameters**

$T \leftarrow$	Data type (double, float, complex <double>,)</double>
_	
N⊷	number of rows of matrix
R⊷	
_	
N↔	number of columns of matrix
C⊷	
_	

Author

Rachid Touzani

Copyright

GNU Lesser Public License

#### 7.66.2 Constructor & Destructor Documentation

LocalMatrix ( )

Default constructor.

Constructs a matrix with 0 rows and 0 columns

## LocalMatrix ( Element \* el, const SpMatrix < $T_- > \& a$ )

Constructor of a local matrix associated to element from a SpMatrix.

#### **Parameters**

in	el	Pointer to Element
in	а	Global matrix as instance of class SpMatrix.

## LocalMatrix ( Element \* el, const SkMatrix < $T_- > \& a$ )

Constructor of a local matrix associated to element from a SkMatrix.

#### Parameters

in	el	Pointer to Element
in	а	Global matrix as instance of class SkMatrix.

## LocalMatrix ( Element \* el, const SkSMatrix< $T_- > \& a$ )

Constructor of a local matrix associated to element from a SkSMatrix.

## Parameters

in	el	Pointer to Element
in	а	Global matrix as instance of class SkSMatrix.

## 7.66.3 Member Function Documentation

## $T_{-}$ % operator() ( size\_t i, size\_t j )

Operator () (Non constant version) Returns entry at row i and column j.

## $T_{-}$ operator() ( size\_t i, size\_t j ) const

Operator () (Constant version)
Returns entry at row i and column j.

## void Localize ( Element \* el, const SpMatrix< $T_- > \& a$ )

Initialize matrix as element matrix from global SpMatrix.

#### **Parameters**

in	el	Pointer to Element
in	а	Global matrix as instance of class SpMatrix. This function is called by its
		corresponding constructor.

## void Localize ( Element \* el, const SkMatrix< $T_- > \& a$ )

Initialize matrix as element matrix from global SkMatrix.

#### **Parameters**

in	el	Pointer to Element	
in	а	Global matrix as instance of class SkMatrix. This function is called by its	
		corresponding constructor.	

## void Localize ( Element \* el, const SkSMatrix< $T_- > \& a$ )

Initialize matrix as element matrix from global SkSMatrix.

#### **Parameters**

in	el	Pointer to Element	
in	а	Global matrix as instance of class SkSMatrix. This function is called by its	
		corresponding constructor.	

## LocalMatrix<T<sub>-</sub>,NR<sub>-</sub>,NC<sub>-</sub>>& operator= ( const LocalMatrix< T<sub>-</sub>, NR<sub>-</sub>, NC<sub>-</sub>> & m )

Operator =

Copy instance m into current instance.

## LocalMatrix<T\_,NR\_,NC\_>& operator= ( const T\_ & x )

Operator =

Assign matrix to identity times x

## LocalMatrix<T<sub>-</sub>,NR<sub>-</sub>,NC<sub>-</sub>>& operator+= ( const LocalMatrix<T<sub>-</sub>, NR<sub>-</sub>, NC<sub>-</sub>>& m )

Operator +=

Add m to current matrix.

## LocalMatrix<T<sub>-</sub>,NR<sub>-</sub>,NC<sub>-</sub>>& operator-= ( const LocalMatrix< T<sub>-</sub>, NR<sub>-</sub>, NC<sub>-</sub>> & m )

Operator -=

Subtract m from current matrix.

## LocalVect<T\_,NR\_> operator\* ( LocalVect< T\_, NC\_> & x )

Operator \*

Return a Vect instance as product of current matrix by vector x.

## LocalMatrix<T\_,NR\_,NC\_>& operator+= ( const T\_ & x )

Operator +=

Add constant x to current matrix entries.

## LocalMatrix<T\_,NR\_,NC\_>& operator== ( const T\_ & x )

Operator -=

Subtract x from current matrix entries.

## LocalMatrix<T\_,NR\_,NC $_>$ & operator\*= ( const T $_-$ & x )

Operator \*=

Multiply matrix entries by constant x.

## LocalMatrix<T\_,NR\_,NC\_>& operator/= ( const T\_ & x )

Operator /=

Divide by x current matrix entries.

## void MultAdd ( const LocalVect< $T_-$ , $NC_-$ > & x, LocalVect< $T_-$ , $NR_-$ > & y)

Multiply matrix by vector and add result to vector.

#### Parameters

in	x	Vector to multiply matrix by.
out	y	Resulting vector (y += a * x)

# void MultAddScal ( const T\_ & a, const LocalVect< T\_, NC\_ > & x, LocalVect< T\_, NR\_ > & y )

Multiply matrix by scaled vector and add result to vector.

#### **Parameters**

in	а	Constant to premultiply by vector x.
in	х	(Scaled) vector to multiply matrix by.
out	y	Resulting vector (y += a * x)

## void Mult ( const LocalVect< $T_-$ , $NC_-$ > & x, LocalVect< $T_-$ , $NR_-$ > & y)

Multiply matrix by vector.

## Parameters

	in	х	Vector to multiply matrix by.
Ī	out	y	Resulting vector.

## void Symmetrize ( )

Symmetrize matrix.

Fill upper triangle to form a symmetric matrix.

#### int Factor ( )

Factorize matrix.

Performs a LU factorization.

#### Returns

- 0: Factorization has ended normally,
- n: n-th pivot was zero.

## int solve ( LocalVect< T $_-$ , NR $_-$ > & b )

Forward and backsubstitute to solve a linear system.

#### **Parameters**

in	b	Right-hand side in input and solution vector in output.
----	---	---

#### Returns

- 0: Solution was performed normally.
- n: n-th pivot is zero.

#### Note

Matrix must have been factorized at first.

## int Factor And Solve ( Local Vect < T $_-$ , NR $_-$ > & b )

Factorize matrix and solve linear system.

#### Parameters

	in,out	b	Right-hand side in input and solution vector in output.	
--	--------	---	---	--

#### Returns

0 if solution was performed normally. n if n-th pivot is zero. This function simply calls **Factor()** then **Solve(b)**.

## void Invert ( LocalMatrix< T<sub>-</sub>, NR<sub>-</sub>, NC<sub>-</sub> > & A )

Calculate inverse of matrix.

#### **Parameters**

out	Α	Inverse of matrix

## T\_ getInnerProduct ( const LocalVect< T\_, NC\_ > & x, const LocalVect< T\_, NR\_ > & y)

Calculate inner product witrh respect to matrix.

Returns the product x<sup>T</sup>Ay

#### **Parameters**

in	x	Left vector
in	y	Right vector

#### Returns

Resulting product

## 7.67 LocalVect< T $_{-}$ , N $_{-}$ > Class Template Reference

Handles small size vectors like element vectors.

#### **Public Member Functions**

• LocalVect ()

Default constructor.

• LocalVect (const T<sub>-</sub> \*a)

Constructor using a C-array.

• LocalVect (const Element \*el)

Constructor using Element pointer.

LocalVect (const Side \*sd)

Constructor using Side pointer.

• LocalVect (const LocalVect < T\_, N\_ > &v)

Copy constructor.

• LocalVect (const Element \*el, const Vect< T\_> &v, int opt=0)

Constructor of an element vector from a global Vect instance.

• LocalVect (const Element &el, const Vect< T\_> &v, int opt=0)

Constructor of an element vector from a global Vect instance.

LocalVect (const Side \*sd, const Vect< T<sub>-</sub> > &v, int opt=0)

Constructor of a side vector from a global Vect instance.

• ~LocalVect ()

Destructor

• void getLocal (const Element &el, const Vect< T\_> &v, int type)

Localize an element vector from a global Vect instance.

• void Localize (const Element \*el, const Vect< T\_> &v, size\_t k=0)

Localize an element vector from a global Vect instance.

• void Localize (const Side \*sd, const Vect<  $T_-$  > &v, size\_t k=0)

Localize a side vector from a global Vect instance.

• T<sub>-</sub> & operator[] (size<sub>-</sub>t i)

Operator [] (Non constant version).

• T\_ operator[] (size\_t i) const

Operator [] (Constant version).

```
• T<sub>-</sub> & operator() (size<sub>-</sub>t i)
      Operator () (Non constant version).
• T_ operator() (size_t i) const
      Operator () (Constant version).

    Element * El ()

      Return pointer to Element if vector was constructed using an element and nullptr otherwise.
• Side * Sd ()
      Return pointer to Side if vector was constructed using a side and nullptr otherwise.
• LocalVect< T_-, N_-> & operator= (const LocalVect<math>< T_-, N_-> &v)
      Operator =
• LocalVect< T_-, N_-> & operator= (const T<math>_- &x)
• LocalVect< T<sub>-</sub>, N<sub>-</sub> > & operator+= (const LocalVect< T<sub>-</sub>, N<sub>-</sub> > &v)
      Operator +=
• LocalVect< T_-, N_- > \& operator += (const T_- \& a)
      Operator +=
• LocalVect< T_-, N_-> & operator= (const LocalVect<math>< T_-, N_-> &v)
      Operator -=
• LocalVect< T_, N_ > & operator= (const T_ &a)
      Operator -=
• LocalVect< T<sub>-</sub>, N<sub>-</sub> > & operator∗= (const T<sub>-</sub> &a)
      Operator *=
• LocalVect< T_-, N_-> & operator/= (const T<math>_- &a)
      Operator /=
• T_* get ()
      Return pointer to vector as a C-Array.
• T_- operator, (const LocalVect< T_-, N_- > \&v) const
      Return Dot (scalar) product of two vectors.
```

## 7.67.1 Detailed Description

```
template < class T_, size_t N_> class OFELI::LocalVect < T_, N_>
```

Handles small size vectors like element vectors.

The template class LocalVect treats small size vectors. Typically, this class is recommended to store element and side arrays. Operators =, [] and () are overloaded so that one can write for instance:

```
LocalVect<double,10> u, v;
v = -1.0;
u = v;
u(3) = -2.0;
```

to set vector  $\mathbf{v}$  entries to -1, copy vector  $\mathbf{v}$  into vector  $\mathbf{u}$  and assign third entry of  $\mathbf{v}$  to -2. Notice that entries of  $\mathbf{v}$  are here  $\mathbf{v}(1)$ ,  $\mathbf{v}(2)$ , ...,  $\mathbf{v}(10)$ , *i.e.* vector entries start at index 1. Internally, no dynamic storage is used.

## **Template Parameters**

$T \leftarrow$	Data type (double, float, complex <double>,)</double>
_←	
N⊷	Vector size
_←	

#### Author

Rachid Touzani

## Copyright

**GNU Lesser Public License** 

## 7.67.2 Constructor & Destructor Documentation

LocalVect ( const Element \* el, const Vect<  $T_- > & v$ , int opt = 0)

Constructor of an element vector from a global Vect instance. The constructed vector has local numbering of nodes

#### **Parameters**

in	el	Pointer to Element to localize	
in	v	Global vector to localize	
in	opt	Option for DOF treatment	
		• = 0, Normal case [Default]	
		<ul> <li>Any other value: only one DOF is handled (Local vector has as dimension number of degrees of freedom)</li> </ul>	

## LocalVect ( const Element & el, const Vect< $T_- > \& v$ , int opt = 0)

Constructor of an element vector from a global Vect instance. The constructed vector has local numbering of nodes

#### **Parameters**

in	el	Reference to Element instance to localize	
in	v	Global vector to localize	
in	opt	Option for DOF treatment	
		• = 0, Normal case [Default]	
		<ul> <li>Any other value: only one DOF is handled (Local vector has as dimension number of degrees of freedom)</li> </ul>	

## LocalVect (const Side \* sd, const Vect< $T_- > & v$ , int opt = 0)

Constructor of a side vector from a global Vect instance.

The constructed vector has local numbering of nodes

## Parameters

in	sd	Pointer to Side to localize	
in	v	Global vector to localize	
in	opt	Option for DOF treatment	
		• = 0, Normal case [Default]	
		<ul> <li>Any other value: only one DOF is handled (Local vector has as dimension number of degrees of freedom)</li> </ul>	

## 7.67.3 Member Function Documentation

## void getLocal ( const Element & el, const Vect $< T_- > \& v$ , int type )

Localize an element vector from a global Vect instance.

The constructed vector has local numbering of nodes This function is called by the constructor  $\leftarrow$ : LocalVect(const Element \*el, const Vect<T $_->$  &v)

#### **Parameters**

in	el	Pointer to Element to localize
in	v	Global vector to localize
in	type	Type of element. This is to be chosen among enumerated values: LINE2, TRIANG3, QUAD4, TETRA4, HEXA8, PENTA6

## void Localize (const Element \* el, const Vect< $T_- > \& v$ , size\_t k = 0)

Localize an element vector from a global Vect instance.

The constructed vector has local numbering of nodes This function is called by the constructor 
∴ Local Vect(const Element \*el, const Vect<T\_> &v)

#### **Parameters**

in	el	Pointer to Side to localize
in	v	Global vector to localize
in	k	Degree of freedom to localize [Default: All degrees of freedom are stored]

## void Localize (const Side \* sd, const Vect< $T_- > \& v$ , size\_t k = 0)

Localize a side vector from a global Vect instance.

The constructed vector has local numbering of nodes This function is called by the constructor  $\leftarrow$ : Local Vect(const Side \*sd, const Vect< $T_-$ > &v)

OFELI's Reference Guide

in	sd	Pointer to Side to localize
in	v	Global vector to localize
in	k	Degree of freedom to localize [Default: All degrees of freedom are stored]

## $T_{-}$ operator[] ( size\_t i )

Operator [] (Non constant version). v[i] starts at v[0] to v[size()-1]

## $T_{-}$ operator[] ( size\_t i ) const

Operator [] (Constant version).
v[i] starts at v[0] to v[size()-1]

## $T_{-}$ % operator() ( size\_t i )

Operator () (Non constant version). v(i) starts at v(1) to v(size()). v(i) is the same element as v[i-1]

## $T_{-}$ operator() ( size\_t i ) const

Operator () (Constant version). v(i) starts at v(1) to v(size()) v(i) is the same element as v[i-1]

## LocalVect<T<sub>-</sub>,N<sub>-</sub>>& operator= ( const LocalVect< T<sub>-</sub>, N<sub>-</sub>> & v )

Operator =

Copy a LocalVect instance to the current one

## LocalVect<T<sub>-</sub>,N<sub>-</sub>>& operator= ( const T<sub>-</sub> & x )

Operator =

Assign value x to all vector entries

#### LocalVect<T<sub>-</sub>,N<sub>-</sub>>& operator+= ( const LocalVect< T<sub>-</sub>, N<sub>-</sub>> & v )

Operator +=

Add vector v to this instance

## LocalVect<T\_, $N_->$ & operator+= ( const T\_ & a )

Operator +=

Add constant a to vector entries

## LocalVect<T $_{-}$ N $_{-}$ >& operator== ( const LocalVect< T $_{-}$ , N $_{-}$ > & v )

Operator -=

Subtract vector v from this instance

## LocalVect<T<sub>-</sub>,N<sub>-</sub>>& operator== ( const T<sub>-</sub> & a )

Operator -=

Subtract constant a from vector entries

## LocalVect<T\_, $N_->$ & operator\*= ( const T\_ & a )

Operator \*=

Multiply vector by constant a

## LocalVect<T<sub>-</sub>,N<sub>-</sub>>& operator/= ( const T<sub>-</sub> & a )

Operator /=

Divide vector by constant a

### $T_-$ operator, (const LocalVect $< T_-$ , $N_- > \& v$ ) const

Return Dot (scalar) product of two vectors.

A typical use of this operator is double a = (v,w) where v and w are 2 instances of Local $\leftarrow$  Vect<double,n>

#### **Parameters**

in	v	LocalVect instance by which the current instance is multiplied
----	---	--

## 7.68 Material Class Reference

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

## **Public Member Functions**

• Material ()

Default consructor.

Material (const Material &m)

Copy constructor.

• ~Material ()

Destructor.

• int set (int m, const string &name)

Associate to material code number n the material named name

• string getName (int m) const

Return material name for material with code m

int getCode (size\_t i) const

Return material code for i-th material.

• size\_t getNbMat () const

Return Number of read materials.

• void setCode (int m)

Associate code m to current material.

- int check (int c)
- real\_t Density ()

Return constant density.

• real\_t Density (const Point < real\_t > &x, real\_t t)

Return density at point x and time t

real\_t SpecificHeat ()

Return constant specific heat.

• real\_t SpecificHeat (const Point< real\_t > &x, real\_t t)

Return specific heat at point x and time t

real\_t ThermalConductivity ()

Return constant thermal conductivity.

real\_t ThermalConductivity (const Point < real\_t > &x, real\_t t)

Return thermal conductivity at point x and time t

real\_t MeltingTemperature ()

Return constant melting temperature.

• real\_t MeltingTemperature (const Point< real\_t > &x, real\_t t)

Return melting temperature at point x and time t

real\_t EvaporationTemperature ()

Return constant evaporation temperature.

real\_t EvaporationTemperature (const Point < real\_t > &x, real\_t t)

Return evaporation temperature at point x and time t

real\_t ThermalExpansion ()

Return constant thermal expansion coefficient.

real\_t ThermalExpansion (const Point < real\_t > &x, real\_t t)

Return thermal expansion coefficient at point x and time t

real\_t LatentHeatForMelting ()

Return constant latent heat for melting.

real\_t LatentHeatForMelting (const Point < real\_t > &x, real\_t t)

Return latent heat for melting at point x and time t

real\_t LatentHeatForEvaporation ()

Return constant latent heat for evaporation.

real\_t LatentHeatForEvaporation (const Point< real\_t > &x, real\_t t)

Return latent heat for evaporation at point x and time t

• real\_t DielectricConstant ()

Return constant dielectric constant.

• real\_t DielectricConstant (const Point< real\_t > &x, real\_t t)

Return dielectric constant at point x and time t

real\_t ElectricConductivity ()

Return constant electric conductivity.

• real\_t ElectricConductivity (const Point< real\_t > &x, real\_t t)

Return electric conductivity at point x and time t

real\_t ElectricResistivity ()

Return constant electric resistivity.

• real\_t ElectricResistivity (const Point< real\_t > &x, real\_t t)

Return electric resistivity at point x and time t

real\_t MagneticPermeability ()

Return constant magnetic permeability.

real\_t MagneticPermeability (const Point< real\_t > &x, real\_t t)

Return magnetic permeability at point x and time t

• real\_t Viscosity ()

Return constant viscosity.

• real\_t Viscosity (const Point< real\_t > &x, real\_t t)

Return viscosity at point x and time t

real\_t YoungModulus ()

Return constant Young modulus.

real\_t YoungModulus (const Point< real\_t > &x, real\_t t)

Return Young modulus at point x and time t

• real\_t PoissonRatio ()

Return constant Poisson ratio.

real\_t PoissonRatio (const Point< real\_t > &x, real\_t t)

Return Poisson ratio at point x and time t

• real\_t Property (int i)

Return constant i-th property.

• real\_t Property (int i, const Point< real\_t > &x, real\_t t)

Return i-th property at point x and time t

Material & operator= (const Material &m)

Operator = .

## 7.68.1 Detailed Description

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

#### 7.68.2 Constructor & Destructor Documentation

#### Material ( )

Default consructor.

It initializes the class and searches for the path where are material data files.

## 7.68.3 Member Function Documentation

```
int set ( int m, const string & name )
```

Associate to material code number n the material named name

Returns

Number of materials

## string getName ( int m ) const

Return material name for material with code m

If such a material is not found, return a blank string.

## int check ( int c )

Check if material code c is present.

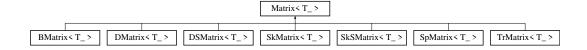
Returns

0 if succeeded, 1 if not.

# 7.69 Matrix $< T_- >$ Class Template Reference

Virtual class to handle matrices for all storage formats.

Inheritance diagram for Matrix< T₋>:



## **Public Member Functions**

• Matrix ()

Default constructor.

• Matrix (const Matrix < T\_ > &m)

Copy Constructor.

• virtual ~Matrix ()

Destructor.

virtual void reset ()

Set matrix to 0 and reset factorization parameter.

size\_t getNbRows () const

Return number of rows.

size\_t getNbColumns () const

Return number of columns.

void setPenal (real\_t p)

Set Penalty Parameter (For boundary condition prescription).

void setDiagonal ()

Set the matrix as diagonal.

• T\_getDiag (size\_t k) const

Return k-th diagonal entry of matrix.

• size\_t size () const

Return matrix dimension (Number of rows and columns).

• virtual void MultAdd (const Vect< T\_> &x, Vect< T\_> &y) const =0

Multiply matrix by vector x and add to y

virtual void MultAdd (T<sub>-</sub> a, const Vect< T<sub>-</sub> > &x, Vect< T<sub>-</sub> > &y) const =0

Multiply matrix by vector  $\mathbf{a} * \mathbf{x}$  and add to  $\mathbf{y}$ 

• virtual void Mult (const Vect< T\_> &x, Vect< T\_> &y) const =0

Multiply matrix by vector x and save in y

• virtual void TMult (const Vect< T\_> &v, Vect< T\_> &w) const =0

Multiply transpose of matrix by vector x and save in y

• virtual void Axpy  $(T_a, const Matrix < T_x > *x)=0$ 

Add to matrix the product of a matrix by a scalar.

void setDiagonal (Mesh &mesh)

*Initialize matrix storage in the case where only diagonal terms are stored.* 

• virtual void clear ()

brief Set all matrix entries to zero

void Assembly (const Element &el, T<sub>-</sub> \*a)

Assembly of element matrix into global matrix.

void Assembly (const Side &sd, T<sub>-</sub>\*a)

Assembly of side matrix into global matrix.

• void Prescribe (Vect< T\_> &b, const Vect< T\_> &u, int flag=0)

Impose by a penalty method an essential boundary condition, using the Mesh instance provided by the constructor.

• void Prescribe (int dof, int code, Vect<  $T_->$  &b, const Vect<  $T_->$  &u, int flag=0)

Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.

• void Prescribe (Vect< T<sub>-</sub> > &b, int flag=0)

*Impose by a penalty method a homegeneous* (=0) *essential boundary condition.* 

• void Prescribe (size\_t dof, Vect< T\_> &b, const Vect< T\_> &u, int flag=0)

Impose by a penalty method an essential boundary condition when only one DOF is treated.

• void PrescribeSide ()

*Impose by a penalty method an essential boundary condition when DOFs are supported by sides.* 

virtual void add (size\_t i, size\_t j, const T\_ &val)=0

Add val to entry (i, j).

• virtual int Factor ()=0

Factorize matrix. Available only if the storage class enables it.

• virtual int solve (Vect< T\_> &b, bool fact=true)=0

Solve the linear system.

• virtual int solve (const Vect< T\_> &b, Vect< T\_> &x, bool fact=true)=0

Solve the linear system.

• int FactorAndSolve (Vect< T<sub>-</sub> > &b)

Factorize matrix and solve the linear system.

• int FactorAndSolve (const Vect<  $T_->$  &b, Vect<  $T_->$  &x)

Factorize matrix and solve the linear system.

• size\_t getLength () const

Return number of stored terms in matrix.

• int isDiagonal () const

Say if matrix is diagonal or not.

• int isFactorized () const

Say if matrix is factorized or not.

• virtual size\_t getColInd (size\_t i) const

Return Column index for column i (See the description for class SpMatrix).

virtual size\_t getRowPtr (size\_t i) const

Return Row pointer for row i (See the description for class SpMatrix).

• virtual void set (size\_t i, size\_t j, const T\_ &val)=0

Assign a value to an entry of the matrix.

• virtual T<sub>-</sub> & operator() (size\_t i, size\_t j)=0

Operator () (Non constant version).

• virtual T\_ operator() (size\_t i, size\_t j) const =0

Operator () (Non constant version).

• T\_operator() (size\_t i) const

Operator () with one argument (Constant version).

• T<sub>-</sub> & operator() (size<sub>-</sub>t i)

Operator () with one argument (Non Constant version).

• T<sub>-</sub> & operator[] (size<sub>-</sub>t k)

Operator [] (Non constant version).

• T\_operator[] (size\_t k) const

Operator [] (Constant version).

• Matrix & operator= (Matrix < T\_ > &m)

Operator = .

Matrix & operator+= (const Matrix < T<sub>-</sub> > &m)

Operator +=.

• Matrix & operator-= (const Matrix < T\_ > &m)

Operator -=.

• Matrix & operator= (const T<sub>-</sub> &x)

Operator = .

• Matrix & operator\*= (const T<sub>-</sub> &x)

Operator \*=.

Matrix & operator+= (const T<sub>-</sub> &x)

Operator +=.

• Matrix & operator-= (const T<sub>-</sub> &x)

Operator -=.

• virtual T\_get (size\_t i, size\_t j) const =0

Return entry (i, j) of matrix if this one is stored, 0 else.

## 7.69.1 Detailed Description

```
template < class T_> class OFELI::Matrix < T_>
```

Virtual class to handle matrices for all storage formats.

This class enables storing and manipulating dense matrices. The template parameter is the type of matrix entries. Any matrix entry can be accessed by the () operator: For instance, if A is an instance of this class, A(i,j) stands for the entry at the i-th row and j-th column, i and j starting from 1. Entries of A can be assigned a value by the same operator.

**Template Parameters** 

<t⊷< th=""><th>Data type (real_t, float, complex<real_t>,)</real_t></th></t⊷<>	Data type (real_t, float, complex <real_t>,)</real_t>
_>	

Author

Rachid Touzani

Copyright

GNU Lesser Public License

## 7.69.2 Constructor & Destructor Documentation

Matrix ( )

Default constructor.

Initializes a zero-size matrix.

#### 7.69.3 Member Function Documentation

## virtual void reset( ) [virtual]

Set matrix to 0 and reset factorization parameter.

#### Warning

This function must be used if after a factorization, the matrix has modified

Reimplemented in DMatrix< T<sub>-</sub>>, and DMatrix< real<sub>-</sub>t>.

## $T_{-}$ getDiag ( size\_t k ) const

Return k-th diagonal entry of matrix.

First entry is given by **getDiag(1)**.

## virtual void Axpy ( $T_-a$ , const Matrix $< T_- > *x$ ) [pure virtual]

Add to matrix the product of a matrix by a scalar.

#### **Parameters**

in	а	Scalar to premultiply
in	x	Matrix by which a is multiplied. The result is added to current instance

Implemented in SpMatrix<  $T_-$  >, SpMatrix< real\_t >, DSMatrix<  $T_-$  >, DSMatrix< real\_ $\leftrightarrow$  t >, DMatrix<  $T_-$  >, DMatrix< real\_t >, SkSMatrix<  $T_-$  >, SkMatrix<  $T_-$  >, TrMatrix<  $T_-$  >, BMatrix<  $T_-$  >, and BMatrix< real\_t >.

## void setDiagonal ( Mesh & mesh )

Initialize matrix storage in the case where only diagonal terms are stored.

This member function is to be used for explicit time integration schemes

## void Assembly (const Element & el, $T_-*a$ )

Assembly of element matrix into global matrix.

Case where element matrix is given by a C-array.

#### Parameters

in	el	Pointer to element instance
in	а	Element matrix as a C-array

## void Assembly (const Side & sd, $T_-*a$ )

Assembly of side matrix into global matrix.

Case where side matrix is given by a C-array.

#### **Parameters**

in	sd	Pointer to side instance

	in	а	Side matrix as a C-array instance	1
--	----	---	-----------------------------------	---

## void Prescribe ( Vect< T $_->$ & b, const Vect< T $_->$ & u, int flag=0)

Impose by a penalty method an essential boundary condition, using the Mesh instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **set**← **Penal**(..).

#### **Parameters**

in,out	b	Vect instance that contains right-hand side.
in	и	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	flag	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

## void Prescribe (int dof, int code, Vect< $T_- > \& b$ , const Vect< $T_- > \& u$ , int flag = 0)

Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **set**← **Penal**(..).

#### Parameters

in	dof	Degree of freedom for which a boundary condition is to be enforced
in	code	Code for which a boundary condition is to be enforced
in,out	b	Vect instance that contains right-hand side.
in	и	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	flag	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

## void Prescribe ( Vect< $T_-$ > & b, int flag = 0 )

Impose by a penalty method a homegeneous (=0) essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **set**← **Penal**(..).

in,out	b	Vect instance that contains right-hand side.
in	flag	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

#### void Prescribe ( size\_t dof, Vect< $T_- > \& b$ , const Vect< $T_- > \& u$ , int flag = 0)

Impose by a penalty method an essential boundary condition when only one DOF is treated.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. This gunction is to be used if only one DOF per node is treated in the linear system. The penalty parameter is by default equal to 1.e20. It can be modified by member function setPenal.

#### **Parameters**

in	dof	Label of the concerned degree of freedom (DOF).
in,out	b	Vect instance that contains right-hand side.
in	и	Vect instance that conatins imposed valued at DOFs where they are to be imposed.
in	flag	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

## void PrescribeSide ( )

Impose by a penalty method an essential boundary condition when DOFs are supported by sides. This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function set←

**Penal**(..).

virtual int solve ( Vect< T $_->$  & b, bool fact = true ) [pure virtual]

Solve the linear system.

If the inherited class is SpMatrix, the function uses an iterative method once this one has been chosen. Otherwise, the method solves the linear system by factorization.

Implemented in DMatrix< T $_->$ , DMatrix< real $_-$ t>, SkSMatrix< T $_->$ , SkMatrix< T $_->$ , D $\leftarrow$  SMatrix< T $_->$ , DSMatrix< real $_-$ t>, BMatrix< T $_->$ , BMatrix< T $_->$ , and TrMatrix< T $_->$ .

 $virtual\ int\ solve\ (\ const\ Vect < T_-> \&\ b,\ Vect < T_-> \&\ x,\ bool\ fact=\ true\ )\quad [\texttt{pure}\ \ virtual]$ 

Solve the linear system.

If the inherited class is SpMatrix, the function uses an iterative method once this one has been chosen. Otherwise, the method solves the linear system by factorization.

## Parameters

in	b	Vect instance that contains right-hand side

out	x	Vect instance that contains solution	
in	fact	Set to true if factorization is to be performed, false if not. [Default: true]	

#### Returns

- 0 if solution was normally performed
- n if the n-th pivot is null Solution is performed only is factorization has previouly been invoked.

Implemented in SpMatrix<  $T_-$  >, SpMatrix< real\_t >, DMatrix<  $T_-$  >, DMatrix< real\_t >, SkSMatrix<  $T_-$  >, SkMatrix<  $T_-$  >, DSMatrix<  $T_-$  >, DSMatrix< real\_t >, BMatrix<  $T_-$  >, B $\leftarrow$  Matrix< real\_t >, and TrMatrix<  $T_-$  >.

## int FactorAndSolve ( Vect< $T_- > \& b$ )

Factorize matrix and solve the linear system.

This is available only if the storage cass enables it.

#### **Parameters**

in,	out	b	Vect instance that contains right-hand side on input and solution on output	]
-----	-----	---	---	---

## int FactorAndSolve ( const Vect< T $_->$ & b, Vect< T $_->$ & x )

Factorize matrix and solve the linear system.

This is available only if the storage class enables it.

## Parameters

in	b	Vect instance that contains right-hand side	
out	x	Vect instance that contains solution	

## Returns

- 0 if solution was normally performed
- n if the n-th pivot is nul

## int isFactorized ( ) const

Say if matrix is factorized or not.

If the matrix was not factorized, the class does not allow solving by a direct solver.

## virtual void set ( size\_t i, size\_t j, const T\_ & val ) [pure virtual]

Assign a value to an entry of the matrix.

in	i	Row index

in	j	Column index
in	val	Value to assign

Implemented in SpMatrix<  $T_-$  >, SpMatrix< real\_t >, SkSMatrix<  $T_-$  >, DMatrix<  $T_-$  >, DMatrix<  $T_-$  >, TrMatrix<  $T_-$  >, BMatrix<  $T_-$  >, BMatrix< real\_t >, DS $\leftarrow$  Matrix<  $T_-$  >, and DSMatrix< real\_t >.

## virtual T\_& operator() ( size\_t i, size\_t j ) [pure virtual]

Operator () (Non constant version).

Returns the (i,j) entry of the matrix.

#### **Parameters**

in	i	Row index
in	j	Column index

Implemented in SpMatrix<  $T_-$  >, SpMatrix< real\_t >, DMatrix<  $T_-$  >, DMatrix< real\_t >, SkSMatrix<  $T_-$  >, SkMatrix<  $T_-$  >, DSMatrix<  $T_-$  >, DSMatrix< real\_t >, TrMatrix<  $T_-$  >, B $\leftarrow$  Matrix<  $T_-$  >, and BMatrix< real\_t >.

## virtual T\_ operator() ( size\_t i, size\_t j ) const [pure virtual]

Operator () (Non constant version).

Returns the (i,j) entry of the matrix.

## Parameters

in	i	Row index
in	j	Column index

Implemented in SpMatrix<  $T_-$  >, SpMatrix< real\_t >, DMatrix<  $T_-$  >, DMatrix< real\_t >, SkSMatrix<  $T_-$  >, SkMatrix<  $T_-$  >, DSMatrix<  $T_-$  >, DSMatrix< real\_t >, TrMatrix<  $T_-$  >, B $\leftarrow$  Matrix<  $T_-$  >, and BMatrix< real\_t >.

# $T_{-}$ operator() ( size\_t i ) const

Operator () with one argument (Constant version).

Returns i-th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

#### **Parameters**

in	i	entry index
----	---	-------------

## $T_{\infty}$ operator() ( size\_t i )

Operator () with one argument (Non Constant version).

Returns i-th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

#### **Parameters**

in i	entry index
------	-------------

## $T_{\infty}$ operator[] ( size\_t k )

Operator [] (Non constant version).

Returns k-th stored element in matrix Index k starts at 0.

# $T_{-}$ operator[] ( size\_t k ) const

Operator [] (Constant version).

Returns k-th stored element in matrix Index k starts at 0.

## Matrix& operator= ( Matrix $< T_- > \& m$ )

Operator =.

Copy matrix m to current matrix instance.

#### Matrix& operator+= ( const Matrix $< T_- > \& m$ )

Operator +=.

Add matrix m to current matrix instance.

## Matrix& operator== ( const Matrix $< T_- > & m$ )

Operator -=.

Subtract matrix m from current matrix instance.

## Matrix& operator= ( const $T_- \& x$ )

Operator =.

Assign constant value x to all matrix entries.

## Matrix& operator\*= ( const $T_- & x$ )

Operator \*=.

Premultiply matrix entries by constant value x

## Matrix& operator+= ( const $T_- & x$ )

Operator +=.

Add constant value x to all matrix entries.

## Matrix& operator== ( const $T_- \& x$ )

Operator -=.

Subtract constant value x from all matrix entries.

## 7.70 Mesh Class Reference

To store and manipulate finite element meshes.

#### **Public Member Functions**

• Mesh ()

Default constructor (Empty mesh)

• Mesh (const string &file, bool bc=false, int opt=NODE\_DOF, int nb\_dof=1)

Constructor using a mesh file.

Mesh (real\_t xmin, real\_t xmax, size\_t nb\_el, size\_t p=1, size\_t nb\_dof=1)

Constructor for a 1-D mesh. The domain is the interval [xmin,xmax].

Mesh (const Grid &g, int opt=QUADRILATERAL)

Constructor for a uniform finite difference grid given by and instance of class Grid.

Mesh (const Grid &g, int shape, int opt)

Constructor of dual mesh for a uniform finite difference grid given by and instance of class Grid.

• Mesh (real\_t xmin, real\_t xmax, size\_t ne, int c1, int c2, int p=1, size\_t nb\_dof=1)

Constructor for a uniform 1-D finite element mesh.

Mesh (real\_t xmin, real\_t xmax, real\_t ymin, real\_t ymax, size\_t nx, size\_t ny, int cx0, int cxN, int cy0, int cyN, int opt=0, size\_t nb\_dof=1)

Constructor for a uniform 2-D structured finite element mesh.

Mesh (real\_t xmin, real\_t xmax, real\_t ymin, real\_t ymax, real\_t zmin, real\_t zmax, size\_
 t nx, size\_t ny, size\_t nz, int cx0, int cxN, int cy0, int cyN, int cz0, int czN, int opt=0, size\_t nb\_dof=1)

Constructor for a uniform 3-D structured finite element mesh.

Mesh (const Mesh &m, const Point < real\_t > &x\_bl, const Point < real\_t > &x\_tr)

Constructor that extracts the mesh of a rectangular region from an initial mesh.

Mesh (const Mesh &mesh, int opt, size\_t dof1, size\_t dof2, bool bc=false)

Constructor that copies the input mesh and selects given degrees of freedom.

Mesh (const Mesh &ms)

Copy Constructor.

• ~Mesh ()

Destructor.

void setDim (size\_t dim)

Define space dimension. Normally, between 1 and 3.

void Add (Node \*nd)

Add a node to mesh.

void Add (Element \*el)

Add an element to mesh.

void Add (Side \*sd)

Add a side to mesh.

• void Add (Edge \*ed)

Add an edge to mesh.

Mesh & operator\*= (real\_t a)

Operator \*=

• void get (const string &mesh\_file)

Read mesh data in file.

• void get (const string &mesh\_file, int ff, int nb\_dof=1)

Read mesh data in file with giving its format.

• void setDOFSupport (int opt, int nb\_nodes=1)

Define supports of degrees of freedom.

void setNbDOFPerNode (size\_t nb\_dof=1)

Define number of degrees of freedom for each node.

• void setPointInDomain (Point< real\_t > x)

Define a point in the domain. This function makes sense only if boundary mesh is given without internal mesh (Case of Boundary Elements)

• void removeImposedDOF ()

Eliminate equations corresponding to imposed DOF.

• size\_t NumberEquations (size\_t dof=0)

Renumber Equations.

• size\_t NumberEquations (size\_t dof, int c)

Renumber Equations.

• int getAllSides (int opt=0)

Determine all mesh sides.

size\_t getNbSideNodes () const

Return the number of nodes on each side.

• size\_t getNbElementNodes () const

Return the number of nodes in each element.

• int getBoundarySides ()

Determine all boundary sides.

• int createBoundarySideList ()

Create list of boundary sides.

int getBoundaryNodes ()

Determine all boundary nodes.

• int createInternalSideList ()

Create list of internal sides (not on the boundary).

int getAllEdges ()

Determine all edges.

void getNodeNeighborElements ()

Create node neighboring elements.

• void getElementNeighborElements ()

Create element neighboring elements.

• void setMaterial (int code, const string &mname)

Associate material to code of element.

void Reorder (size\_t m=GRAPH\_MEMORY)

Renumber mesh nodes according to reverse Cuthill Mc Kee algorithm.

• void Add (size\_t num, real\_t \*x)

Add a node by giving its label and an array containing its coordinates.

void DeleteNode (size\_t label)

Remove a node given by its label.

void DeleteElement (size\_t label)

Remove an element given by its label.

• void DeleteSide (size\_t label)

Remove a side given by its label.

• void Delete (Node \*nd)

Remove a node given by its pointer.

• void Delete (Element \*el)

Remove a node given by its pointer.

• void Delete (Side \*sd)

Remove a side given by its pointer.

• void Delete (Edge \*ed)

Remove an edge given by its pointer.

• void RenumberNode (size\_t n1, size\_t n2)

Renumber a node.

• void RenumberElement (size\_t n1, size\_t n2)

Renumber an element.

• void RenumberSide (size\_t n1, size\_t n2)

Renumber a side.

• void RenumberEdge (size\_t n1, size\_t n2)

Renumber an edge.

void setList (const std::vector < Node \* > &nl)

*Initialize list of mesh nodes using the input vector.* 

• void setList (const std::vector< Element \* > &el)

*Initialize list of mesh elements using the input vector.* 

void setList (const std::vector < Side \* > &sl)

Initialize list of mesh sides using the input vector.

• void Rescale (real\_t sx, real\_t sy=0., real\_t sz=0.)

Rescale mesh by multiplying node coordinates by constants.

• size\_t getDim () const

Return space dimension.

size\_t getNbNodes () const

Return number of nodes.

• size\_t getNbMarkedNodes () const

Return number of marked nodes.

size\_t getNbVertices () const

Return number of vertices.

size\_t getNbDOF () const

Return total number of degrees of freedom (DOF)

• size\_t getNbEq () const

Return number of equations.

size\_t getNbEq (int i) const

 $Return\ number\ of\ equations\ for\ the\ i-th\ set\ of\ degrees\ of\ freedom.$ 

• size\_t getNbElements () const

Return number of elements.

• size\_t getNbSides () const

Return number of sides.

• size\_t getNbEdges () const

Return number of sides.

• size\_t getNbBoundarySides () const

Return number of boundary sides.

• size\_t getNbInternalSides () const

Return number of internal sides.

• size\_t getNbMat () const

Return number of materials.

• void AddMidNodes (int g=0)

Add mid-side nodes.

Point< real\_t > getMaxCoord () const

Return maximum coordinates of nodes.

• Point< real\_t > getMinCoord () const

Return minimum coordinates of nodes.

void set (Node \*nd)

Replace node in the mesh.

• void set (Element \*el)

Replace element in the mesh.

void set (Side \*sd)

Choose side in the mesh.

• bool NodesAreDOF () const

Return information about DOF type.

bool SidesAreDOF () const

Return information about DOF type.

• bool EdgesAreDOF () const

Return information about DOF type.

• bool ElementsAreDOF () const

Return information about DOF type.

• int getDOFSupport () const

Return information on dof support Return an integer according to enumerated values: NODE\_DOF, EL← EMENT\_DOF SIDE\_DOF.

• void put (const string &mesh\_file) const

Write mesh data on file.

void save (const string &mesh\_file) const

Write mesh data on file in various formats.

• bool withImposedDOF () const

Return true if imposed DOF count in equations, false if not.

• bool isStructured () const

Return true is mesh is structured, false if not.

• size\_t getNodeNewLabel (size\_t n) const

Return new label of node of a renumbered node.

void getList (vector < Node \* > &nl) const

Fill vector nl with list of pointers to nodes.

• void getList (vector< Element \* > &el) const

Fill vector el with list of pointers to elements.

void getList (vector < Side \* > &sl) const

Fill vector sl with list of pointers to sides.

Node \* getPtrNode (size\_t i) const

Return pointer to node with label i.

Node & getNode (size\_t i) const

Return refenrece to node with label i

• Element \* getPtrElement (size\_t i) const

Return pointer to element with label i

Element & getElement (size\_t i) const

Return reference to element with label i

• Side \* getPtrSide (size\_t i) const

Return pointer to side with label i

• Side & getSide (size\_t i) const

Return reference to side with label i

Edge \* getPtrEdge (size\_t i) const

Return pointer to edge with label i

Edge & getEdge (size\_t i) const

Return reference to edge with label i

size\_t getNodeLabel (size\_t i) const

Return label of i-th node.

size\_t getElementLabel (size\_t i) const

Return label of *i*-th element.

• size\_t getSideLabel (size\_t i) const

Return label of i-th side.

• size\_t getEdgeLabel (size\_t i) const

Return label of i-th edge.

• void topNode () const

Reset list of nodes at its top position (Non constant version)

void topBoundaryNode () const

Reset list of boundary nodes at its top position (Non constant version)

• void topMarkedNode () const

Reset list of marked nodes at its top position (Non constant version)

• void topElement () const

Reset list of elements at its top position (Non constant version)

• void topSide () const

Reset list of sides at its top position (Non constant version)

• void topBoundarySide () const

Reset list of boundary sides at its top position (Non constant version)

• void topInternalSide () const

Reset list of intrenal sides at its top position (Non constant version)

• void topEdge () const

Reset list of edges at its top position (Non constant version)

• void topBoundaryEdge () const

Reset list of boundary edges at its top position (Non constant version)

• Node \* getNode () const

Return pointer to current node and move to next one (Non constant version)

Node \* getBoundaryNode () const

Return pointer to current boundary node and move to next one (Non constant version)

Node \* getMarkedNode () const

Return pointer to current marked node and move to next one (Non constant version)

• Element \* getElement () const

Return pointer to current element and move to next one (Non constant version)

Element \* getActiveElement () const

Return pointer to current element and move to next one (Non constant version)

• Side \* getSide () const

Return pointer to current side and move to next one (Non constant version)

Side \* getBoundarySide () const

Return pointer to current boundary side and move to next one (Non constant version)

Side \* getInternalSide () const

Return pointer to current internal side and move to next one (Non constant version)

• Edge \* getEdge () const

Return pointer to current edge and move to next one (Non constant version)

Edge \* getBoundaryEdge () const

Return pointer to current boundary edge and move to next one (Non constant version)

• int getShape () const

Determine shape of elements Return Shape index (see enum ElementShape) if all elements have the same shape, 0 if not.

Element \* operator() (size\_t i) const

*Operator () : Return pointer to i-th element.* 

Node \* operator[] (size\_t i) const

Operator []: Return pointer to i-th node.

• size\_t operator() (size\_t i, size\_t n) const

*Operator () : Return pointer to i-th node of n-th element.* 

• Mesh & operator= (Mesh &ms)

*Operator* = : *Assign a Mesh instance*.

#### **Friends**

• void Refine (Mesh &in\_mesh, Mesh &out\_mesh)

Refine mesh. Subdivide each triangle into 4 subtriangles. This member function is valid for 2-D triangular meshes only.

# 7.70.1 Detailed Description

To store and manipulate finite element meshes.

Class Mesh enables defining as an object a finite element mesh. A finite element mesh is characterized by its nodes, elements and sides. Each of these types of data constitutes a class in the OFELI library.

The standard procedure to introduce the finite element mesh is to provide an input file containing its data. For this, we have defined our own mesh data file (following the XML syntax). Of course, a developer can write his own function to read his finite element mesh file using the methods in Mesh.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

## 7.70.2 Constructor & Destructor Documentation

Mesh (const string & file, bool bc = false, int  $opt = NODE\_DOF$ , int  $nb\_dof = 1$ )

Constructor using a mesh file.

in	file	File containing mesh data. The extension of the file yields the file format: The extension .m implies OFELI file format and .msh implies GMSH msh file.
in	bc	Flag to remove (true) or not (false) imposed Degrees of Freedom [default: false]
in	opt	Type of DOF support: To choose among enumerated values NODE_DOF, SIDE_DOF or ELEMENT_DOF.  Say if degrees of freedom (unknowns) are supported by nodes, sides or elements.
in	nb_dof	Number of degrees of freedom per node [Default: 1].

# Mesh ( real\_t xmin, real\_t xmax, size\_t $nb_el$ , size\_t p = 1, size\_t $nb_dof = 1$ )

Constructor for a 1-D mesh. The domain is the interval [xmin,xmax].

#### **Parameters**

in	xmin	Value of xmin		
in	xmax	Value of xmax		
in	nb_el	Number of elements to generate		
in	p	Degree of finite element polynomial (Default = 1)		
in	nb_dof	Number of degrees of freedom for each node (Default = 1)		

# Mesh ( const Grid & g, int opt = QUADRILATERAL )

Constructor for a uniform finite difference grid given by and instance of class Grid.

#### **Parameters**

in	8	Grid instance	
in	opt	Optional value to say which type of elements to generate	
		• TRIANGLE: Mesh elements are triangles	
		QUADRILATERAL: Mesh elements are quadrilaterals [default]	

# Mesh (const Grid & g, int shape, int opt)

Constructor of dual mesh for a uniform finite difference grid given by and instance of class Grid.

in	8	Grid instance	
in	shape	Value to say which type of elements to generate	
		• TRIANGLE: Mesh elements are triangles	
		• QUADRILATERAL: Mesh elements are quadrilaterals [default]	

in	opt	This argument can take any value. It is here only to distinguish from the other
		constructor using Grid instance.

## Remarks

This constructor is to be used to obtain a dual mesh from a structured grid. It is mainly useful if a cell centered finite volume method is used.

## Mesh ( real\_t xmin, real\_t xmax, size\_t ne, int c1, int c2, int p = 1, size\_t nb\_dof = 1 )

Constructor for a uniform 1-D finite element mesh.

The domain is the line (xmin,xmax)

#### **Parameters**

in	xmin	Minimal coordinate	
in	xmax	Maximal coordinate	
in	пе	Number of elements	
in	c1	Code for the first node (x=xmin)	
in	c2	Code for the last node (x=xmax)	
in	р	Degree of approximation polynomial [Default: 1].	
in	nb_dof	Number of degrees of freedom per node [Default: 1].	

## Remarks

The option p can be set to 1 if the user intends to use finite differences.

# Mesh ( real\_t xmin, real\_t xmax, real\_t ymin, real\_t ymax, size\_t nx, size\_t ny, int cx0, int cxN, int cyN, int cyN, int opt = 0, size\_t $nb\_dof = 1$ )

Constructor for a uniform 2-D structured finite element mesh.

The domain is the rectangle (xmin,xmax)x(ymin,ymax)

in	xmin	Minimal x-coordinate			
in	xmax	Maximal x-coordinate			
in	ymin	Minimal y-coordinate			
in	ymax	Maximal y-coordinate			
in	пх	Number of subintervals on the x-axis			
in	ny	Number of subintervals on the y-axis			
in	cx0	Code for nodes generated on the line $x=x0$ if $>0$ , for sides on this line if $<0$			
in	cxN	Code for nodes generated on the line $x=xN$ if $>0$ , for sides on this line if $<0$			
in	cy0	Code for nodes generated on the line $y=y0$ if $>0$ , for sides on this line if $<0$			
in	cyN	Code for nodes generated on the line $y=yN$ if $>0$ , for sides on this line if $<0$			

in	opt	Flag to generate elements as well (if not zero) [Default: 0]. If the flag is not 0, it can take one of the enumerated values: TRIANGLE or QUADRILATERAL, with obvious meaning.	
in	nb_dof	Number of degrees of freedom per node [Default: 1].	

#### Remarks

The option opt can be set to 0 if the user intends to use finite differences.

Mesh ( real\_t xmin, real\_t xmax, real\_t ymin, real\_t ymax, real\_t zmin, real\_t zmax, size\_t nx, size\_t ny, size\_t nz, int cx0, int cxN, int cyN, int cyN, int czN, int czN, int opt = 0, size\_t  $nb\_dof = 1$ )

Constructor for a uniform 3-D structured finite element mesh.

The domain is the parallepiped (xmin,xmax)x(ymin,ymax)x(zmin,zmax)

## Parameters

in	xmin	Minimal x-coordinate			
in	xmax	Maximal x-coordinate			
in	ymin	Minimal y-coordinate			
in	ymax	Maximal y-coordinate			
in	zmin	Minimal z-coordinate			
in	zmax	Maximal z-coordinate			
in	nx	Number of subintervals on the x-axis			
in	ny	Number of subintervals on the y-axis			
in	nz	Number of subintervals on the z-axis			
in	cx0	Code for nodes generated on the line $x=xmin if >0$ , for sides on this line if $<0$			
in	cxN	Code for nodes generated on the line $x=xmax$ if $>0$ , for sides on this line if $<0$			
in	cy0	Code for nodes generated on the line y=ymin if $>0$ , for sides on this line if $<0$			
in	cyN	Code for nodes generated on the line y=ymax if $>0$ , for sides on this line if $<0$			
in	cz0	Code for nodes generated on the line $z=zmin$ if $>0$ , for sides on this line if $<0$			
in	czN	Code for nodes generated on the line $z=zmax$ if $>0$ , for sides on this line if $<0$			
in	opt	Flag to generate elements as well (if not zero) [Default: 0]. If the flag is not 0, it can take one of the enumerated values: HEXAHEDRON or TETRAHEDRON, with obvious meaning.			
in	nb_dof	Number of degrees of freedom per node [Default: 1].			

#### Remarks

The option opt can be set to 0 if the user intends to use finite differences.

Mesh ( const Mesh & m, const Point< real\_t > &  $x_bl$ , const Point< real\_t > &  $x_tr$ )

Constructor that extracts the mesh of a rectangular region from an initial mesh.

This constructor is useful for zooming purposes for instance.

#### Parameters

in	m	Initial mesh from which the submesh is extracted
in	x⊷	Coordinate of bottom left vertex of the rectangle
	_bl	
in	$\chi \leftarrow$	Coordinate of top right vertex of the rectangle
	_tr	

# Mesh ( const Mesh & mesh, int opt, size\_t dof1, size\_t dof2, bool bc = false)

Constructor that copies the input mesh and selects given degrees of freedom.

This constructor is to be used for coupled problems where each subproblem uses a choice of degrees of freedom.

#### Parameters

in	mesh	Initial mesh from which the submesh is extracted		
in	opt	Type of DOF support: To choose among enumerated values NODE_DOF, SIDE_DOF or ELEMENT_DOF.		
in	dof1	Label of first degree of freedom to select to the output mesh		
in	dof2	Label of last degree of freedom to select to the output mesh		
in	bc	Flag to remove (true) or not (false) imposed Degrees of Freedom [Default: false]		

## Mesh (const Mesh & ms)

Copy Constructor.

#### Parameters

in ms Mesh instance to cop	y
----------------------------	---

## 7.70.3 Member Function Documentation

## void setDim ( size\_t dim )

Define space dimension. Normally, between 1 and 3.

# Parameters

in	dim	Space dimension to set (must be between 1 and 3)
----	-----	--

## void Add ( Node \* nd )

Add a node to mesh.

in	nd	Pointer to Node to add
----	----	------------------------

## void Add ( Element \* el )

Add an element to mesh.

#### Parameters

in	el	Pointer to Element to add
----	----	---------------------------

# void Add ( Side \* sd )

Add a side to mesh.

#### Parameters

in	sd	Pointer to Side to add
----	----	------------------------

# void Add ( Edge \* ed )

Add an edge to mesh.

#### Parameters

in	ed	Pointer to Edge to add
----	----	------------------------

## Mesh& operator\*= ( real\_t a )

Operator \*=

Rescale mesh coordinates by myltiplying by a factor

## Parameters

in	а	Value to multiply by
----	---	----------------------

# void get ( const string & mesh\_file )

Read mesh data in file.

Mesh file must be in OFELI format. See "File Formats" page

in	mesh_file	Mesh file name

# void get ( const string & mesh\_file, int ff, int $nb\_dof = 1$ )

Read mesh data in file with giving its format.

File format can be chosen among a variety of choices. See "File Formats" page

#### **Parameters**

i	n	mesh_file	Mesh file name	
i	n	ff	File format: Integer to chose among enumerated values: OFELI_FF, GMSH,	
			MATLAB, EASYMESH, GAMBIT, BAMG, NETGEN, TRIANGLE_FF	
i	n	nb_dof	Number of degrees of freedom per node (Default value: 1)	

## void setDOFSupport ( int opt, int nb\_nodes = 1 )

Define supports of degrees of freedom.

#### Parameters

in	opt	DOF type:		
		NODE_DOF: Degrees of freedom are supported by nodes		
		<ul> <li>SIDE_DOF: Degrees of freedom are supported by sides</li> </ul>		
		<ul> <li>EDGE_DOF: Degrees of freedom are supported by edges</li> </ul>		
		ELEMENT_DOF: Degrees of freedom are supported by elements		
in	nb_nodes	Number of nodes on sides or elements (default=1). This parameter is useful only if dofs are supported by sides or elements		

## Note

This member function creates all mesh sides if the option ELEMENT\_DOF or SIDE\_DOF is selected. So it not necessary to call getAllSides() after

# void setNbDOFPerNode ( size\_t nb\_dof = 1 )

Define number of degrees of freedom for each node.

in	nb_dof	Number of degrees of freedom (unknowns) for each mesh node (Default value
		is 1)

#### Note

This function first declares nodes as unknown supports, sets the number of degrees of freedom and renumbers equations

## void setPointInDomain ( Point < real\_t > x )

Define a point in the domain. This function makes sense only if boundary mesh is given without internal mesh (Case of Boundary Elements)

#### **Parameters**

	in	x	Coordinates of point to define	
--	----	---	--------------------------------	--

## size\_t NumberEquations ( size\_t dof = 0 )

Renumber Equations.

#### **Parameters**

in	dof	Label of degree of freedom for which numbering is performed. Default value (0)
		means that all degrees of freedom are taken into account

# size\_t NumberEquations ( size\_t dof, int c )

Renumber Equations.

#### Parameters

in	dof	Label of degree of freedom for which numbering is performed.
in	С	code for which degrees of freedom are enforced.

## int getAllSides ( int opt = 0 )

Determine all mesh sides.

Returns

Number of all sides.

## int getBoundarySides ( )

Determine all boundary sides.

Returns

Number of boundary sides.

## int createBoundarySideList ( )

Create list of boundary sides.

This function is useful to loop over boundary sides without testing Once this one is called, the function <a href="mailto:getNbBoundarySides">getNbBoundarySides</a>() is available. Moreover, looping over boundary sides is available via the member functions topBoundarySide() and <a href="mailto:getBoundarySide">getBoundarySide</a>()

#### Returns

Number of boundary sides.

## int getBoundaryNodes ( )

Determine all boundary nodes.

Returns

n Number of boundary nodes.

## int createInternalSideList( )

Create list of internal sides (not on the boundary).

This function is useful to loop over internal sides without testing Once this one is called, the function <a href="mailto:getNbInternalSides">getNbInternalSides</a>() is available. Moreover, looping over internal sides is available via the member functions topInternalSide() and <a href="mailto:getInternalSide">getInternalSide</a>()

#### Returns

n Number of internal sides.

## int getAllEdges ( )

Determine all edges.

Returns

Number of all edges.

## void getNodeNeighborElements ( )

Create node neighboring elements.

This function is generally useful when, for a numerical method, one looks for a given node to the list of elements that share this node. Once this function is invoked, one can retrieve the list of neighboring elements of any node (Node::getNeigEl)

## void getElementNeighborElements ( )

Create element neighboring elements.

This function creates for each element the list of elements that share a side with it. Once this function is invoked, one can retrieve the list of neighboring elements of any element (Element← ::getNeigborElement)

## void setMaterial ( int code, const string & mname )

Associate material to code of element.

in	code	Element code for which material is assigned
in	mname	Name of material

## void Reorder ( size\_t m = GRAPH\_MEMORY )

Renumber mesh nodes according to reverse Cuthill Mc Kee algorithm.

#### **Parameters**

in m Memory size needed for matrix graph (default value is GRAPH_MEMORY, OFELI_Config.h)	see
--	-----

## void Add ( size\_t num, real\_t \* x )

Add a node by giving its label and an array containing its coordinates.

#### **Parameters**

in	num	Label of node to add	
in	x	C-array of node coordinates	

## void DeleteNode ( size\_t label )

Remove a node given by its label.

This function does not release the space previously occupied

## Parameters

## void DeleteElement ( size\_t label )

Remove an element given by its label.

This function does not release the space previously occupied

## Parameters

in	label	Label of element to delete

## void DeleteSide ( size\_t label )

Remove a side given by its label.

This function does not release the space previously occupied

in <i>label</i>	Label of side to delete
-----------------	-------------------------

## void Delete ( Node \* nd )

Remove a node given by its pointer.

This function does not release the space previously occupied

#### **Parameters**

in	nd	Pointer to node to delete
----	----	---------------------------

## void Delete ( Element \* el )

Remove a node given by its pointer.

This function does not release the space previously occupied

## Parameters

	in	el	Pointer to element to delete	
--	----	----	------------------------------	--

## void Delete (Side \* sd)

Remove a side given by its pointer.

This function does not release the space previously occupied

#### **Parameters**

in	sd	Pointer to side to delete
----	----	---------------------------

# void Delete ( Edge \* ed )

Remove an edge given by its pointer.

This function does not release the space previously occupied

#### Parameters

in	ed	Pointer to edge to delete

# void RenumberNode ( size\_t n1, size\_t n2 )

Renumber a node.

in	n1	Old label
in	n2	New label

## void RenumberElement ( size\_t n1, size\_t n2 )

Renumber an element.

#### Parameters

in	n1	Old label
in	n2	New label

# void RenumberSide ( size\_t n1, size\_t n2 )

Renumber a side.

#### Parameters

in	n1	Old label
in	n2	New label

# void RenumberEdge ( size\_t n1, size\_t n2 )

Renumber an edge.

#### Parameters

in	n1	Old label
in	n2	New label

## void setList ( const std::vector< Node \* > & nl )

Initialize list of mesh nodes using the input vector.

## Parameters

in	nl	vector instance that contains the list of pointers to nodes
----	----	---

## void setList ( const std::vector< Element \* > & el )

Initialize list of mesh elements using the input vector.

## Parameters

in	el	vector instance that contains the list of pointers to elements
----	----	--

## void setList ( const std::vector< Side \* > & sl )

Initialize list of mesh sides using the input vector.

in	sl	vector instance that contains the list of pointers to sides	1
----	----	---	---

## void Rescale ( real\_t sx, real\_t sy = 0., real\_t sz = 0. )

Rescale mesh by multiplying node coordinates by constants.

This function can be used e.g. for changing coordinate units

#### **Parameters**

in	sx	Factor to multiply by x coordinates
in	sy	Factor to multiply by y coordinates [Default: sx]
in	SZ	Factor to multiply by z coordinates [Default: sx]

## size\_t getNbBoundarySides ( ) const

Return number of boundary sides.

This function is valid if member function **getAllSides** or **getBoundarySides** has been invoked before

## size\_t getNbInternalSides ( ) const

Return number of internal sides.

This function is valid if member functions **getAllSides** and **createInternalSideList** have been invoked before

## void AddMidNodes ( int g = 0 )

Add mid-side nodes.

This is function is valid for triangles only

## Parameters

in	Q	Option to say of barycentre node is to be added (>0) or not (=0)
	0	•

## void set ( Node \* nd )

Replace node in the mesh.

If the node label exists already, the existing node pointer will be replaced by the current one. If not, an error message is displayed.

in	nd	Pointer to node
----	----	-----------------

#### void set ( Element \* el )

Replace element in the mesh.

If the element label exists already, the existing element pointer will be replaced by the current one. If not, an error message is displayed.

#### **Parameters**

ſ	in	el	Pointer to element
---	----	----	--------------------

#### void set ( Side \* sd )

Choose side in the mesh.

If the side label exists already, the existing side pointer will be replaced by the current one. If not, an error message is displayed.

#### **Parameters**

	- 1	Datatanta at 1a
1n	Su	Pointer to side

## bool NodesAreDOF ( ) const

Return information about DOF type.

Returns

true if DOF are supported by nodes, false otherwise

## bool SidesAreDOF ( ) const

Return information about DOF type.

Returns

true if DOF are supported by sides, false otherwise

## bool EdgesAreDOF ( ) const

Return information about DOF type.

Returns

true if DOF are supported by edges, false otherwise

## bool ElementsAreDOF ( ) const

Return information about DOF type.

Returns

true if DOF are supported by elements, false otherwise

## void put ( const string & mesh\_file ) const

Write mesh data on file.

	in	mesh_file	Mesh file name
--	----	-----------	----------------

## void save ( const string & mesh\_file ) const

Write mesh data on file in various formats. File format depends on the extension in file name

#### Parameters

in	mesh_file	Mesh file name If the extension is '.m', the output file is an OFELI file If the
		extension is '.gpl', the output file is a Gnuplot file If the extension is '.msh' or
		'.geo', the output file is a Gmsh file If the extension is '.vtk', the output file is a
		VTK file

## void getList ( vector< Node \* > & nl ) const

Fill vector nl with list of pointers to nodes.

#### Parameters

	out	nl	Instance of class vector that contain on output the list
--	-----	----	--

## void getList ( vector< Element \* > & el ) const

Fill vector el with list of pointers to elements.

#### Parameters

	out	el	Instance of class vector that contain on output the list	_
--	-----	----	--	---

## void getList ( vector< Side \* > & sl ) const

Fill vector s1 with list of pointers to sides.

# Parameters

out	sl	Instance of class vector that contain on output the list

## size\_t getNodeLabel ( size\_t i ) const

Return label of i-th node.

in	i	Node index
----	---	------------

## size\_t getElementLabel ( size\_t i ) const

Return label of i-th element.

#### **Parameters**

in i Element inc
------------------

## size\_t getSideLabel ( size\_t i ) const

Return label of i-th side.

#### **Parameters**

in i Side	index
-----------	-------

## size\_t getEdgeLabel ( size\_t i ) const

Return label of i-th edge.

#### Parameters

in	i	Edge index
----	---	------------

# Element\* getActiveElement ( ) const

Return pointer to current element and move to next one (Non constant version)

This function returns pointer to the current element only is this one is active. Otherwise it goes to the next active element (To be used when adaptive meshing is involved)

## 7.70.4 Friends And Related Function Documentation

void Refine ( Mesh & in\_mesh, Mesh & out\_mesh ) [friend]

Refine mesh. Subdivide each triangle into 4 subtriangles. This member function is valid for 2-D triangular meshes only.

#### **Parameters**

in	in_mesh	Input mesh
out	out_mesh	Output mesh

# 7.71 MeshAdapt Class Reference

To adapt mesh in function of given solution.

## **Public Member Functions**

• MeshAdapt ()

Default constructor.

MeshAdapt (Mesh &ms)

Constructor using initial mesh.

• MeshAdapt (Domain &dom)

Constructor using a reference to class Domain.

• ~MeshAdapt ()

Destructor.

• Domain & getDomain () const

Get reference to Domain instance.

• Mesh & getMesh () const

Get reference to current mesh.

• void set (Domain &dom)

Set reference to Domain instance.

• void set (Mesh &ms)

Set reference to Mesh instance.

void setSolution (const Vect< real\_t > &u)

Define label of node.

• void setJacobi (int n)

Set number of Jacobi iterations for smoothing.

void setSmooth (int n)

Set number of smoothing iterations.

• void AbsoluteError ()

Metric is constructed with absolute error.

void RelativeError ()

Metric is constructed with relative error.

• void setError (real\_t err)

Set error threshold for adaption.

• void setHMin (real\_t h)

 $Set\ minimal\ mesh\ size.$ 

void setHMax (real\_t h)

Set maximal mesh size.

void setHMinAnisotropy (real\_t h)

Set minimal mesh size and set anisotropy.

void setRelaxation (real\_t omega)

Set relaxation parameter for smoothing.

void setAnisotropic ()

Set that adapted mesh construction is anisotropic.

void MaxAnisotropy (real\_t a)

Set maximum ratio of anisotropy.

• void setMaxSubdiv (real\_t s)

Change the metric such that the maximal subdivision of a background's edge is bounded by the given number (always limited by 10)

• void setMaxNbVertices (size\_t n)

Set maximum number of vertices.

void setRatio (real\_t r)

Set ratio for a smoothing of the metric.

• void setNoScaling ()

Do not scale solution before metric computation.

void setNoKeep ()

Do not keep old vertices.

• void setHessian ()

set computation of the Hessian

• void setOutputMesh (string file)

Create mesh output file.

void setGeoFile (string file)

Set Geometry file.

• void setGeoError (real\_t e)

Set error on geometry.

• void setBackgroundMesh (string bgm)

Set background mesh.

• void SplitBoundaryEdges ()

Split edges with two vertices on boundary.

• void CreateMetricFile (string mf)

Create a metric file.

void setMetricFile (string mf)

Set Metric file.

• void getSolutionMbb (string mbb)

Set solution defined on background mesh for metric construction.

void getSolutionMBB (string mBB)

Set solution defined on background mesh for metric construction.

• void getSolutionbb (string rbb)

Read solution defined on the background mesh in bb file.

• void getSolutionBB (string rBB)

Read solution defined on the background mesh in BB file.

• void getSolution (Vect< real\_t > &u, int is=1)

Get the interpolated solution on the new mesh.

void getInterpolatedSolutionbb ()

Write the file of interpolation of the solutions in bb file.

• void getInterpolatedSolutionBB ()

Write the file of interpolation of the solutions in BB file.

• void setTheta (real\_t theta)

Set angular limit for a corner (in degrees)

• void Split ()

Split triangles into 4 triangles.

void saveMbb (string file, const Vect< real\_t > &u)

Save a solution in metric file.

• int run ()

Run adaptation process.

• int run (const Vect< real\_t > &u)

Run adaptation process using a solution vector.

• int run (const Vect< real\_t > &u, Vect< real\_t > &v)

Run adaptation process using a solution vector and interpolates solution on the adapted mesh.

## 7.71.1 Detailed Description

To adapt mesh in function of given solution.

Class MeshAdapt enables modifying mesh according to a solution vector defining at nodes. It concerns 2-D triangular meshes only.

#### Remarks

Class MeshAdapt is mainly based on the software 'Bamg' developed by F. Hecht, Universite Pierre et Marie Curie, Paris. We warmly thank him for accepting incoporation of Bamg in the OFELI package

#### Author

Rachid Touzani

## Copyright

**GNU** Lesser Public License

## 7.71.2 Constructor & Destructor Documentation

## MeshAdapt (Mesh & ms)

Constructor using initial mesh.

#### **Parameters**

in   ms   Reference to initial mesh
-------------------------------------

# MeshAdapt ( Domain & dom )

Constructor using a reference to class Domain.

#### Parameters

in   dom   Reference to Domain class
--------------------------------------

## 7.71.3 Member Function Documentation

## void setRelaxation ( real\_t omega )

Set relaxation parameter for smoothing.

Default value for relaxation parameter is 1.8

#### void setMaxNbVertices ( size\_t n )

Set maximum number of vertices. Default value is 500000

## void setRatio ( real\_t r )

Set ratio for a smoothing of the metric.

in r Ratio	o value.
------------	----------

#### Note

If r is 0 then no smoothing is performed, if r lies in [1.1,10] then the smoothing changes the metric such that the largest geometrical progression (speed of mesh size variation in mesh is bounded by r) (by default no smoothing)

## void setNoScaling( )

Do not scale solution before metric computation. By default, solution is scaled (between 0 and 1)

## void setNoKeep ( )

Do not keep old vertices.

By default, old vertices are kept

## void getSolutionbb ( string rbb )

Read solution defined on the background mesh in bb file. Solution is interpolated on created mesh

## void getSolutionBB ( string rBB )

Read solution defined on the background mesh in BB file. Solution is interpolated on created mesh

## void getSolution ( Vect< real\_t > & $u_r$ int is = 1 )

Get the interpolated solution on the new mesh.

The solution must have been saved on an output bb file

#### **Parameters**

out	и	Vector that contains on output the obtained solutions. This vector is resized before being initialized
in	is	[Default: 1]

## void setTheta ( real\_t theta )

Set angular limit for a corner (in degrees)

The angle is defined from 2 normals of 2 consecutive edges

# void saveMbb ( string file, const Vect< real\_t > & u )

Save a solution in metric file.

in	file	File name where the metric is stored
in	и	Solution vector to store

## int run ( )

Run adaptation process.

## Returns

#### Return code:

- = 0: Adaptation has been normally completed
- = 1: An error occured

## int run ( const Vect< real\_t > & u )

Run adaptation process using a solution vector.

#### **Parameters**

in	и	Solution vector defined on the input mesh
----	---	---

## Returns

## Return code:

- = 0: Adaptation has been normally completed
- = 1: An error occured

## int run ( const Vect< real\_t > & u, Vect< real\_t > & v)

Run adaptation process using a solution vector and interpolates solution on the adapted mesh.

#### Parameters

in		и	Solution vector defined on the input mesh
in	.	v	Solution vector defined on the (adapted) output mesh

#### Returns

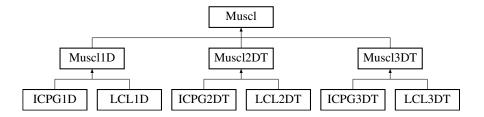
#### Return code:

- = 0: Adaptation has been normally completed
- = 1: An error occured

# 7.72 Muscl Class Reference

Parent class for hyperbolic solvers with Muscl scheme.

Inheritance diagram for Muscl:



# **Public Types**

```
    enum Method {
        FIRST_ORDER_METHOD = 0,
        MULTI_SLOPE_Q_METHOD = 1,
        MULTI_SLOPE_M_METHOD = 2 }
```

Enumeration for flux choice.

```
    enum Limiter {
        MINMOD_LIMITER = 0,
        VANLEER_LIMITER = 1,
        SUPERBEE_LIMITER = 2,
        VANALBADA_LIMITER = 3,
        MAX_LIMITER = 4 }
```

Enumeration of flux limiting methods.

```
    enum SolverType {
        ROE_SOLVER = 0,
        VFROE_SOLVER = 1,
        LF_SOLVER = 2,
        RUSANOV_SOLVER = 3,
        HLL_SOLVER = 4,
        HLLC_SOLVER = 5,
        MAX_SOLVER = 6 }
```

Enumeration of various solvers for the Riemann problem.

## **Public Member Functions**

• Muscl (Mesh &m)

Constructor using mesh instance.

• virtual ~Muscl ()

Destructor.

• void setTimeStep (real\_t dt)

Assign time step value.

real\_t getTimeStep () const

Return time step value.

• void setCFL (real\_t CFL)

Assign CFL value.

• real\_t getCFL () const

Return CFL value.

void setReferenceLength (real\_t dx)

Assign reference length value.

real\_t getReferenceLength () const

Return reference length.

• Mesh & getMesh () const

Return reference to Mesh instance.

• void setVerbose (int v)

Set verbosity parameter.

bool setReconstruction (const Vect< real\_t > &U, Vect< real\_t > &LU, Vect< real\_t > &RU, size\_t dof)

Function to reconstruct by the Muscl method.

void setMethod (const Method &s)

Choose a flux solver.

• void setSolidZoneCode (int c)

Choose a code for solid zone.

• bool getSolidZone () const

Return flag for presence of solid zones.

• int getSolidZoneCode () const

Return code of solid zone, 0 if this one is not present.

• void setLimiter (Limiter l)

Choose a flux limiter.

# 7.72.1 Detailed Description

Parent class for hyperbolic solvers with Muscl scheme.

Everything here is common for both 2D and 3D muscl methods! Virtual functions are implemented in Muscl2D and Muscl3D classes

Author

S. Clain, V. Clauzon

Copyright

GNU Lesser Public License

# 7.72.2 Member Enumeration Documentation

#### enum Method

Enumeration for flux choice.

Enumerator

FIRST\_ORDER\_METHOD First Order upwind method MULTI\_SLOPE\_Q\_METHOD Multislope Q method MULTI\_SLOPE\_M\_METHOD Multislope M method

#### enum Limiter

Enumeration of flux limiting methods.

## Enumerator

MINMOD\_LIMITER MinMod limiter

VANLEER\_LIMITER Van Leer limiter

SUPERBEE\_LIMITER Superbee limiter

VANALBADA\_LIMITER Van Albada limiter

MAX\_LIMITER Max limiter

## enum SolverType

Enumeration of various solvers for the Riemann problem.

#### Enumerator

ROE\_SOLVER Roe solver
VFROE\_SOLVER Finite Volume Roe solver
LF\_SOLVER LF solver
RUSANOV\_SOLVER Rusanov solver
HLL\_SOLVER HLL solver
HLLC\_SOLVER HLLC solver
MAX\_SOLVER Max solver

## 7.72.3 Member Function Documentation

# void setTimeStep ( real\_t dt )

Assign time step value.

## Parameters

in	dt	Time step value

## void setCFL ( real\_t CFL )

Assign CFL value.

#### Parameters

in	CFL	Value of CFL

# void setReferenceLength ( real\_t dx )

Assign reference length value.

	in	dx	Value of reference length
--	----	----	---------------------------

#### void set Verbose ( $\operatorname{int} v$ )

Set verbosity parameter.

#### Parameters

in	v	Value of verbosity parameter
----	---	------------------------------

# bool setReconstruction ( const Vect< real\_t > & U, Vect< real\_t > & LU, Vect< real\_t > & RU, size\_t dof )

Function to reconstruct by the Muscl method.

## Parameters

in	U	Field to reconstruct
out	LU	Left gradient vector
out	RU	Right gradient vector
in	dof	Label of dof to reconstruct

## void setMethod ( const Method & s )

Choose a flux solver.

#### Parameters

## void setLimiter ( Limiter l )

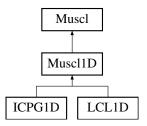
Choose a flux limiter.

## Parameters

in l	Limiter to choose
------	-------------------

# 7.73 Muscl1D Class Reference

Class for 1-D hyperbolic solvers with Muscl scheme. Inheritance diagram for Muscl1D:



## **Public Member Functions**

• Muscl1D (Mesh &m)

Constructor using mesh instance.

• ~Muscl1D ()

Destructor.

• real\_t getMeanLength () const

Return mean length.

• real\_t getMaximumLength () const

Return maximal length.

• real\_t getMinimumLength () const

Return mimal length.

• real\_t getTauLim () const

Return mean length.

void print\_mesh\_stat ()

Output mesh information.

## **Additional Inherited Members**

# 7.73.1 Detailed Description

Class for 1-D hyperbolic solvers with Muscl scheme.

Author

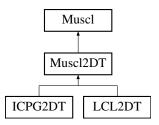
S. Clain, V. Clauzon

Copyright

GNU Lesser Public License

# 7.74 Muscl2DT Class Reference

Class for 2-D hyperbolic solvers with Muscl scheme. Inheritance diagram for Muscl2DT:



## **Public Member Functions**

• Muscl2DT (Mesh &m)

Constructor using mesh.

• ~Muscl2DT ()

Destructor.

bool setReconstruction (const Vect< real\_t > &U, Vect< real\_t > &LU, Vect< real\_t > &RU, size\_t dof)

Function to reconstruct by the Muscl method.

# **Protected Member Functions**

• void Initialize ()

Construction of normals to sides.

## **Additional Inherited Members**

## 7.74.1 Detailed Description

Class for 2-D hyperbolic solvers with Muscl scheme.

Author

S. Clain, V. Clauzon

Copyright

GNU Lesser Public License

## 7.74.2 Member Function Documentation

bool setReconstruction ( const Vect< real\_t > & U, Vect< real\_t > & LU, Vect< real\_t > & RU, size\_t dof )

Function to reconstruct by the Muscl method.

## Parameters

in	U	Field to reconstruct
out	LU	Left gradient vector
out	RU	Right gradient vector
in	dof	Label of dof to reconstruct

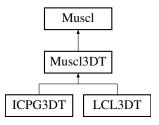
## void Initialize( ) [protected]

Construction of normals to sides.

Convention: for a given side, getPtrElement(1) is the left element and getPtrElement(2) is the right element. The normal goes from left to right. For boundary sides, the normal points outward.

# 7.75 Muscl3DT Class Reference

Class for 3-D hyperbolic solvers with Muscl scheme using tetrahedra. Inheritance diagram for Muscl3DT:



# **Public Member Functions**

• Muscl3DT (Mesh &m)

Constructor using mesh.

• ~Muscl3DT ()

Destructor.

bool setReconstruction (const Vect< real\_t > &U, Vect< real\_t > &LU, Vect< real\_t > &RU, size\_t dof)

Function to reconstruct by the Muscl method.

real\_t getMinimumFaceArea () const

Return minimum area of faces in the mesh.

• real\_t getMinimumElementVolume () const

Return minimum volume of elements in the mesh.

• real\_t getMaximumFaceArea () const

Return maximum area of faces in the mesh.

real\_t getMaximumElementVolume () const

Return maximum volume of elements in the mesh.

• real\_t getMeanFaceArea () const

Return mean area of faces in the mesh.

real\_t getMeanElementVolume () const

Return mean volume of elements in the mesh.

real\_t getMinimumEdgeLength () const

 $Return\ minimum\ length\ of\ edges\ in\ the\ mesh.$ 

• real\_t getMinimumVolumebyArea () const

Return minimum volume by area in the mesh.

• real\_t getMaximumEdgeLength () const

Return maximum length of edges in the mesh.

• real\_t getTauLim () const

Return value of tau lim.

• real\_t getComega () const

Return value of Comega.

• void setbetalim (real\_t bl)

Assign value of beta lim.

# **Additional Inherited Members**

# 7.75.1 Detailed Description

Class for 3-D hyperbolic solvers with Muscl scheme using tetrahedra.

Author

S. Clain, V. Clauzon

Copyright

**GNU Lesser Public License** 

#### 7.75.2 Member Function Documentation

bool setReconstruction ( const Vect< real\_t > & U, Vect< real\_t > & LU, Vect< real\_t > & RU, size\_t dof )

Function to reconstruct by the Muscl method.

#### **Parameters**

in	U	Field to reconstruct
out	LU	Left gradient vector
out	RU	Right gradient vector
in	dof	Label of dof to reconstruct

# 7.76 MyNLAS Class Reference

Abstract class to define by user specified function.

### **Public Member Functions**

• MyNLAS ()

Default Constructor.

• MyNLAS (const Mesh &mesh)

Constructor using mesh instance.

• virtual ~MyNLAS ()

Destructor.

• virtual real\_t Function (const Vect< real\_t > &x, int i=1)=0

Virtual member function to define nonlinear function to zeroe.

• virtual real\_t Gradient (const Vect< real\_t > &x, int i=1, int j=1)

Virtual member function to define partial derivatives of function.

# 7.76.1 Detailed Description

Abstract class to define by user specified function.

The user has to implement a class that inherits from the present one where the virtual functions are implemented.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

### 7.76.2 Constructor & Destructor Documentation

MyNLAS (const Mesh & mesh)

Constructor using mesh instance.

Parameters

*mesh* Reference to Mesh instance

# 7.76.3 Member Function Documentation

virtual real t Function ( const Vect < real t > & x, int i = 1 ) [pure virtual]

Virtual member function to define nonlinear function to zeroe.

#### **Parameters**

in	x	Vector of variables
in	i	component of function to define [Default: 1].

#### Returns

Value of function

# Warning

The component must not be larger than vector size

virtual real\_t Gradient ( const Vect< real\_t > & x, int i = 1, int j = 1 ) [virtual]

Virtual member function to define partial derivatives of function.

in	x	Vector of variables
in	i	Function component [Default: 1]
in	j	Index of partial derivative [Default: 1]

#### Returns

Value of partial derivative

# 7.77 MyOpt Class Reference

Abstract class to define by user specified optimization function.

### **Public Member Functions**

• MyOpt ()

Default Constructor.

• MyOpt (Mesh &mesh)

Constructor using mesh instance.

• virtual ~MyOpt ()

Destructor.

• virtual real\_t Objective (Vect< real\_t > &x)=0

Virtual member function to define objective.

• virtual void Gradient (Vect< real\_t > &x, Vect< real\_t > &g)

Virtual member function to define gradient vector of objective.

void setEquation (AbsEqua < real\_t > \*eq)

Define equation instance.

AbsEqua < real\_t > \* getEquation () const

Get pointer to equation instance.

# 7.77.1 Detailed Description

Abstract class to define by user specified optimization function.

The user has to implement a class that inherits from the present one where the virtual functions are implemented.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

### 7.77.2 Constructor & Destructor Documentation

MyOpt ( Mesh & mesh )

Constructor using mesh instance.

Parameters

*mesh* Reference to Mesh instance

# 7.77.3 Member Function Documentation

virtual real\_t Objective ( Vect < real\_t > & x ) [pure virtual]

Virtual member function to define objective.

#### **Parameters**

in	х	Vector of optimization variables
----	---	----------------------------------

#### Returns

Value of objective

# virtual void Gradient ( Vect< real\_t > & x, Vect< real\_t > & g ) [virtual]

Virtual member function to define gradient vector of objective.

#### **Parameters**

in	x	Vector of optimization variables
out	8	Gradient vector

# void setEquation ( AbsEqua < real\_t > \* eq )

Define equation instance.

# Parameters

in	eq	Pointer to equation instance

### Remarks

This member function is to be invoked in the user class defining the optimization problem

# AbsEqua<real\_t>\* getEquation ( ) const

Get pointer to equation instance.

### Returns

Pointer to equation instance

# 7.78 NLASSolver Class Reference

To solve a system of nonlinear algebraic equations of the form f(u) = 0.

# **Public Member Functions**

• NLASSolver ()

Default constructor.

• NLASSolver (NonLinearIter nl, int nb\_eq=1)

Constructor defining the iterative method to solve the equation.

• NLASSolver (real\_t &x, NonLinearIter nl=NEWTON)

Constructor defining a one-variable problem.

• NLASSolver (Vect< real\_t > &x, NonLinearIter nl=NEWTON)

Constructor defining a multi-variable problem.

• NLASSolver (MyNLAS &my\_nlas, NonLinearIter nl=NEWTON)

Constructor using a user defined class.

• ∼NLASSolver ()

Destructor.

• void setVerbose (int verb)

Set verbosity parameter Value must be between 0 and 5. Default value is 0.

• void setMaxIter (int max\_it)

Set Maximal number of iterations.

void setTolerance (real\_t toler)

Set tolerance value for convergence.

• void set (NonLinearIter nl)

Define an iterative procedure To be chosen among the enumerated values: BISECTION, REGULA\_FALSI or NEWTON

void setFunction (function < real\_t(real\_t) > f)

*Define the function associated to the equation to solve.* 

void setFunction (function < Vect < real\_t > (Vect < real\_t >) > f)

Define the function associated to the equation to solve.

void setGradient (function < real\_t(real\_t) > g)

*Define the function associated to the derivative of the equation to solve.* 

void setGradient (function < Vect < real\_t > (Vect < real\_t >) > g)

Define the function associated to the gradient of the equation to solve.

• void setf (string exp)

Set function for which zero is sought (case of one equation)

• void setDf (string exp, int i=1, int j=1)

Set pzrtial derivative of function for which zero is sought (case of many equations)

void setPDE (AbsEqua < real\_t > &eq)

Define a PDE.

• void setInitial (Vect< real\_t > &u)

*Set initial guess for the iterations.* 

• void setInitial (real\_t &x)

Set initial guess for a unique unknown.

• void setInitial (real\_t a, real\_t b)

Set initial guesses bisection or Regula falsi algorithms.

• void run ()

Run the solution procedure.

• real\_t get () const

Return solution (Case of a scalar equation)

void get (Vect< real\_t > &u) const

Return solution (case of a nonlinear system of equations)

• int getNbIter () const

Return number of iterations.

# 7.78.1 Detailed Description

To solve a system of nonlinear algebraic equations of the form f(u) = 0. Features:

- The nonlinear problem is solved by the Newton's method in the general case, and in the one variable case, either by the bisection or the Regula Falsi method
- The function and its gradient are given:
  - Either by regular expressions
  - Or by user defined functions
  - Or by a user defined class. This feature enables defining the function and its gradient through a PDE class for instance

### 7.78.2 Constructor & Destructor Documentation

### NLASSolver (NonLinearIter nl, int $nb_{eq} = 1$ )

Constructor defining the iterative method to solve the equation.

#### **Parameters**

in	nl	Choose an iterative procedure to solve the nonlinear system of equations: To be chosen among the enumerated values: BISECTION, REGULA_FALSI or NEWTON.
in	nb↔ _eq	Number of equations [Default: 1]

### NLASSolver ( real\_t & x, NonLinearIter nl = NEWTON )

Constructor defining a one-variable problem.

### Parameters

in	x	Variable containing on input initial guess and on output solution, if convergence is achieved
in	nl	Iterative procedure to solve the nonlinear system of equations: To be chosen among the enumerated values: BISECTION, REGULA_FALSI or NEWTON.

# NLASSolver ( Vect< real\_t > & x, NonLinearIter nl = NEWTON )

Constructor defining a multi-variable problem.

in	x	Variable containing on input initial guess and on output solution, if convergence is achieved
in	nl	Iterative procedure to solve the nonlinear system of equations: The only possible value (default one) in the current version is NEWTON.

### NLASSolver ( MyNLAS & my\_nlas, NonLinearIter nl = NEWTON )

Constructor using a user defined class.

#### **Parameters**

in	my_nlas	Reference to instance of user defined class. This class inherits from abstract class MyNLAS. It must contain the member function Vect <double> Function(const Vect<double>&amp; x) which returns the value of the nonlinear function, as a vector, for a given solution vector x. The user defined class must contain, if the iterative scheme requires it the member function Vect<double> Gradient(const Vect<real_t>&amp; x) which returns the gradient as a n*n vector, each index (i,j) containing the j-th partial derivative of the i-th function.</real_t></double></double></double>
in	nl	Iterative procedure to solve the nonlinear system of equations: To be chosen among the enumerated values: BISECTION, REGULA_FALSI or NEWTON.

# 7.78.3 Member Function Documentation

### void setMaxIter ( int max\_it )

Set Maximal number of iterations.

Default value of this parameter is 100

### void setTolerance ( real\_t toler )

Set tolerance value for convergence.

Default value of this parameter is 1.e-8

# void setFunction ( function < real\_t(real\_t) > f )

Define the function associated to the equation to solve.

This function can be used in the case where a user defined function is to be given. To be used in the one-variable case.

#### **Parameters**

in	f	Function given as a function of one real variable and returning a real number. This	
		function can be defined by the calling program as a C-function and then cast to an	
		instance of class function	

# void setFunction ( function < Vect< real\_t > (Vect< real\_t >) > f )

Define the function associated to the equation to solve.

This function can be used in the case where a user defined function is to be given.

in	f	Function given as a function of many variables, stored in an input vector, and
		returns a vector. This function can be defined by the calling program as a C-function
		and then cast to an instance of class function

# void setGradient ( function < real\_t(real\_t) > g )

Define the function associated to the derivative of the equation to solve.

#### **Parameters**

in	g	Function given as a function of one real variable and returning a real number. This	
		function can be defined by the calling program as a C-function and then cast to an	
		instance of class function	

# void setGradient ( function < Vect< real\_t > (Vect< real\_t >) > g )

Define the function associated to the gradient of the equation to solve.

#### **Parameters**

in	g	Function given as a function of many variables, stored in an input vector. and	
		returns a n*n vector (	
	n is the number of variables). This function can be defined by the calling progr		
		a C-function and then cast to an instance of class function	

# void setf ( string exp )

Set function for which zero is sought (case of one equation)

### Parameters

in	ехр	Regular expression defining the function using the symbol x as a variable
----	-----	---

# void setDf ( string exp, int i = 1, int j = 1 )

Set pzrtial derivative of function for which zero is sought (case of many equations)

#### Parameters

in	ехр	Regular expression defining the partial derivative. In this expression, the variables are x1, x2, x10 (up to 10 variables)	
in	i	Component of function [Default: =1]	
in	j	Index of the partial derivative [Default: =1]	

# void setPDE ( AbsEqua < real\_t > & eq )

Define a PDE.

The solver can be used to solve a nonlinear PDE. In this case, the PDE is defined as an instance of a class inheriting of AbsEqua.

#### Parameters

i:	n	eq	Pointer to equation instance
----	---	----	------------------------------

# void setInitial ( Vect< real\_t > & u )

Set initial guess for the iterations.

#### Parameters

in	и	Vector containing initial guess for the unknown
----	---	---

# void setInitial ( real\_t & x )

Set initial guess for a unique unknown.

#### Parameters

in	x	Rference to value of initial guess
----	---	------------------------------------

### void setInitial ( real\_t a, real\_t b )

Set initial guesses bisection or Regula falsi algorithms.

### Parameters

in	а	Value of first initial guess
in	b	Value of second initial guess

### Note

The function has to have opposite signs at these values i.e. f(a)f(b) < 0.

# Warning

This function makes sense only in the case of a unique function of one variable

# void get ( Vect < real\_t > & u ) const

Return solution (case of a nonlinear system of equations)

out	и	Vector that contains on output the solution
		rector trial corruins on output the services

# 7.79 Node Class Reference

To describe a node.

#### **Public Member Functions**

• Node ()

Default constructor.

• Node (size\_t label, const Point< real\_t > &x)

Constructor with label and coordinates.

• Node (const Node &node)

Copy Constructor.

• ~Node ()

Destructor.

• void setLabel (size\_t label)

Define label of node.

void setNbDOF (size\_t n)

Define number of DOF.

• void setFirstDOF (size\_t n)

Define First DOF.

• void setCode (size\_t dof, int code)

Define code for a given DOF of node.

• void setCode (const vector< int > &code)

Define codes for all node DOFs.

• void setCode (int \*code)

Define codes for all node DOFs.

• void setCode (const string &exp, int code, size\_t dof=1)

Define code by a boolean algebraic expression invoking node coordinates.

void setCoord (size\_t i, real\_t x)

Set i-th coordinate.

• void DOF (size\_t i, size\_t dof)

Define label of DOF.

• void setDOF (size\_t &first\_dof, size\_t nb\_dof)

Define number of DOF.

• void setOnBoundary ()

Set node as boundary node.

• size\_t n () const

Return label of node.

• size\_t getNbDOF () const

Return number of degrees of freedom (DOF)

• int getCode (size\_t dof=1) const

Return code for a given DOF of node.

real\_t getCoord (size\_t i) const

Return i-th coordinate of node. i = 1..3.

• Point< real\_t > getCoord () const

Return coordinates of node.

real\_t getX () const

Return x-coordinate of node.

• real\_t getY () const

Return y-coordinate of node.

• real\_t getZ () const

Return z-coordinate of node.

Point< real\_t > getXYZ () const

Return coordinates of node.

• size\_t getDOF (size\_t i) const

Return label of i-th dof.

size\_t getNbNeigEl () const

Return number of neighbor elements.

• Element \* getNeigEl (size\_t i) const

Return i-th neighbor element.

• size\_t getFirstDOF () const

Return label of first DOF of node.

• bool isOnBoundary () const

Say if node is a boundary node.

• void Add (Element \*el)

Add element pointed by el as neighbor element to node.

• void setLevel (int level)

Assign a level to current node.

int getLevel () const

Return node level.

# 7.79.1 Detailed Description

To describe a node.

A node is characterized by its label, its coordinates, its number of degrees of freedom (DOF) and codes that are associated to each DOF.

### Remarks

Once the mesh is constructed, information on neighboring elements of node can be retrieved (see appropriate member functions). However, the member function getNode $\leftarrow$  NeighborElements of Mesh must have been called before. If this is not the case, the program crashes down since no preliminary checking is done for efficiency reasons.

# 7.79.2 Constructor & Destructor Documentation

### Node ( )

Default constructor.

Initialize data to zero

### Node ( size\_t label, const Point < real\_t > & x )

Constructor with label and coordinates.

in	label	Label of node
in	x	Node coordinates

# 7.79.3 Member Function Documentation

# void setCode ( size\_t dof, int code )

Define code for a given DOF of node.

#### Parameters

in	dof	DOF index
in	code	Code to assign to DOF

# void setCode ( const vector< int > & code )

Define codes for all node DOFs.

#### Parameters

in	code	vector instance that contains code for each DOF of current node
----	------	---

# void setCode ( int \* code )

Define codes for all node DOFs.

#### **Parameters**

in	code	C-array that contains code for each DOF of current node
----	------	---

# void setCode ( const string & exp, int code, size\_t dof = 1 )

Define code by a boolean algebraic expression invoking node coordinates.

### Parameters

in	ехр	Boolean algebraic expression as required by fparser
in	code	Code to assign to node if the algebraic expression is true
in	dof	Degree of Freedom for which code is assigned [Default: 1]

# void setCoord ( size\_t i, real\_t x )

Set i-th coordinate.

in	i	Coordinate index (13)
in	x	Coordinate value

### void DOF ( size\_t i, size\_t dof )

Define label of DOF.

#### **Parameters**

in	i	DOF index
in	dof	Label of DOF

### void setDOF ( size\_t & first\_dof, size\_t nb\_dof )

Define number of DOF.

#### **Parameters**

in,out	first_dof	f Label of the first DOF in input that is actualize	
in	nb_dof	Number of DOF	

# void setOnBoundary ( )

Set node as boundary node.

This function is mostly internally used (Especially in class Mesh)

# int getCode ( $size_t dof = 1$ ) const

Return code for a given DOF of node.

### **Parameters**

in	dof	label of degree of freedom for which code is to be returned. Default value is 1.
----	-----	--

### Point<real\_t> getCoord ( ) const

Return coordinates of node.

Return value is an instance of class Point

# Point<real\_t> getXYZ ( ) const

Return coordinates of node.

Return value is an instance of class Point

# size\_t getNbNeigEl ( ) const

Return number of neighbor elements.

Neighbor elements are those that share node. Note that the returned information is valid only if the Mesh member function **getNodeNeighborElements()** has been invoked before

# Element\* getNeigEl ( size\_t i ) const

Return i-th neighbor element.

Note that the returned information is valid only if the Mesh member function **getNode**← **NeighborElements()** has been invoked before

### bool isOnBoundary ( ) const

Say if node is a boundary node.

Note this information is available only if boundary sides (and nodes) were determined (See class Mesh).

### void setLevel ( int level )

Assign a level to current node.

This member function is useful for mesh adaption.

Default node's level is zero

# int getLevel ( ) const

Return node level.

Node level decreases when element is refined (starting from 0). If the level is 0, then the element has no parents

# 7.80 NodeList Class Reference

Class to construct a list of nodes having some common properties.

### **Public Member Functions**

NodeList (Mesh &ms)

Constructor using a Mesh instance.

• ~NodeList ()

Destructor.

• void selectCode (int code, int dof=1)

Select nodes having a given code for a given degree of freedom.

• void unselectCode (int code, int dof=1)

Unselect nodes having a given code for a given degree of freedom.

void selectCoordinate (real\_t x, real\_t y=ANY, real\_t z=ANY)

Select nodes having given coordinates.

• size\_t getNbNodes () const

Return number of selected nodes.

void top ()

Reset list of nodes at its top position (Non constant version)

void top () const

Reset list of nodes at its top position (Constant version)

• Node \* get ()

Return pointer to current node and move to next one (Non constant version)

Node \* get () const

Return pointer to current node and move to next one (Constant version)

# 7.80.1 Detailed Description

Class to construct a list of nodes having some common properties.

This class enables choosing multiple selection criteria by using function select... However, the intersection of these properties must be empty.

### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

# 7.80.2 Member Function Documentation

void selectCode ( int code, int dof = 1 )

Select nodes having a given code for a given degree of freedom.

#### **Parameters**

in	code	Code that nodes share
in	dof	Degree of Freedom label [Default: 1]

### void unselectCode ( int code, int dof = 1 )

Unselect nodes having a given code for a given degree of freedom.

### Parameters

in	code	Code of nodes to exclude
in	dof	Degree of Freedom label [Default: 1]

# void selectCoordinate ( real\_t x, real\_t y = ANY, real\_t z = ANY )

Select nodes having given coordinates.

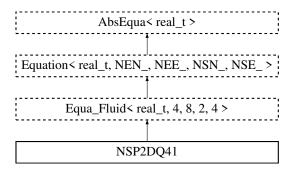
#### **Parameters**

in	x	x-coordinate that share the selected nodes
in	y	y-coordinate that share the selected nodes [Default: ANY]
in	z	z-coordinate that share the selected nodes [Default: ANY]

Coordinates can be assigned the value ANY. This means that any coordinate value is accepted. For instance, to select all nodes with x=0, use **selectCoordinate**(0.,ANY,ANY);

# 7.81 NSP2DQ41 Class Reference

Builds finite element arrays for incompressible Navier-Stokes equations in 2-D domains using  $Q_1/P_0$  element and a penaly formulation for the incompressibility condition. Inheritance diagram for NSP2DQ41:



# **Public Member Functions**

NSP2DQ41 (Mesh &ms)

Constructor using mesh data.

• NSP2DQ41 (Mesh &ms, Vect< real\_t > &u)

Constructor using mesh data and velocity vector.

• ~NSP2DQ41 ()

Destructor.

void setPenalty (real\_t lambda)

Define penalty parameter.

void setViscosity (real\_t visc)

Define constant viscosity.

• void setDensity (real\_t dens)

Define constant density.

• void setInput (EqDataType opt, Vect< real\_t > &u)

Set equation input data.

• void Periodic (real\_t coef=1.e20)

Add contribution of periodic boundary condition (by a penalty technique).

• void build ()

Build the linear system of equations.

• int runOneTimeStep ()

Run one time step.

### 7.81.1 Detailed Description

Builds finite element arrays for incompressible Navier-Stokes equations in 2-D domains using  $Q_1/P_0$  element and a penaly formulation for the incompressibility condition.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# 7.81.2 Constructor & Destructor Documentation

### NSP2DQ41 ( Mesh & ms )

Constructor using mesh data.

#### **Parameters**

in ms Mesh insta
------------------

### NSP2DQ41 ( Mesh & ms, Vect< real\_t > & u )

Constructor using mesh data and velocity vector.

#### **Parameters**

in	ms	Mesh instance
in,out	и	Velocity vector

### 7.81.3 Member Function Documentation

# void setPenalty ( real\_t lambda )

Define penalty parameter.

Penalty parameter is used to enforce the incompressibility constraint

### Parameters

in	lambda	Penaly parameter: Large value [Default: 1.e07]

# void setInput ( EqDataType opt, Vect< real\_t > & u )

Set equation input data.

### Parameters

in	opt	Parameter that selects data type for input. This parameter is to be chosen in the enumerated variable EqDataType
in	и	Vect instance that contains input vector data List of data types contains INITIAL_FIELD, BOUNDARY_CONDITION_DATA, SOURCE_DATA or FLUX with obvious meaning

### void Periodic ( real\_t coef = 1.e20 )

Add contribution of periodic boundary condition (by a penalty technique).

Boundary nodes where periodic boundary conditions are to be imposed must have codes equal to PERIODIC\_A on one side and PERIODIC\_B on the opposite side.

in	coef	Value of penalty parameter [Default: 1.e20].
----	------	--

### void build ( )

Build the linear system of equations.

Before using this function, one must have properly selected appropriate options for:

- The choice of a steady state or transient analysis. By default, the analysis is stationary
- In the case of transient analysis, the choice of a time integration scheme and a lumped or consistent capacity matrix. If transient analysis is chosen, the lumped capacity matrix option is chosen by default, and the implicit Euler scheme is used by default for time integration.

### int runOneTimeStep ( )

Run one time step.

This function performs one time step, once a time integration scheme has been selected.

# 7.82 ODESolver Class Reference

To solve a system of ordinary differential equations.

### **Public Member Functions**

• ODESolver ()

Default constructor.

• ODESolver (TimeScheme s, real\_t time\_step=theTimeStep, real\_t final\_time=theFinalTime, size\_t nb\_eq=1)

Constructor using time discretization data.

• ∼ODESolver ()

Destructor.

void set (TimeScheme s, real\_t time\_step=theTimeStep, real\_t final\_time=theFinalTime)

Define data of the differential equation or system.

void setNbEq (size\_t nb\_eq)

Set the number of equations [Default: 1].

void setCoef (real\_t a0, real\_t a1, real\_t a2, real\_t f)

Define coefficients in the case of a scalar differential equation.

void setCoef (string a0, string a1, string a2, string f)

Define coefficients in the case of a scalar differential equation.

void setLinear ()

Claim that ODE is linear.

• void setF (string f)

Set time derivative, given as an algebraic expression, for a nonlinear ODE.

• void setF (string f, int i)

Set time derivative, given as an algebraic expression, for a nonlinear ODE.

• void setDF (string df)

Set time derivative of the function defining the ODE.

void setDF (string df, int i)

Set time derivative with respect to the unknown of the function defining the ODE.

• void setRK4RHS (real\_t f)

Set intermediate right-hand side vector for the Runge-Kutta method.

• void setRK4RHS (Vect< real\_t > &f)

Set intermediate right-hand side vector for the Runge-Kutta method.

• void setInitial (Vect< real\_t > &u)

Set initial condition for a first-oder system of differential equations.

• void setInitial (real\_t u, int i)

Set initial condition for a first-oder system of differential equations.

void setInitial (Vect< real\_t > &u, Vect< real\_t > &v)

Set initial condition for a second-order system of differential equations.

void setInitialRHS (Vect< real\_t > &f)

Set initial RHS for a system of differential equations.

void setInitial (real\_t u, real\_t v)

Set initial condition for a second-order ordinary differential equation.

• void setInitial (real\_t u)

Set initial condition for a first-order ordinary differential equation.

void setInitialRHS (real\_t f)

Set initial right-hand side for a single differential equation.

void setMatrices (DMatrix < real\_t > &A0, DMatrix < real\_t > &A1)

Define matrices for a system of first-order ODEs.

void setMatrices (DMatrix < real\_t > &A0, DMatrix < real\_t > &A1, DMatrix < real\_t > &A2)

Define matrices for a system of second-order ODEs.

• void seODEVectors (Vect< real\_t > &a0, Vect< real\_t > &a1)

Define matrices for an implicit nonlinear system of first-order ODEs.

• void seODEVectors (Vect< real\_t > &a0, Vect< real\_t > &a1, Vect< real\_t > &a2)

Define matrices for an implicit nonlinear system of second-order ODEs.

void setRHS (Vect< real\_t > &b)

Set right-hand side vector for a system of ODE.

void setRHS (real\_t f)

Set right-hand side for a linear ODE.

• void setRHS (string f)

Set right-hand side value for a linear ODE.

void setNewmarkParameters (real\_t beta, real\_t gamma)

 $Define\ parameters\ for\ the\ Newmarxk\ scheme.$ 

void setConstantMatrix ()

*Say that matrix problem is constant.* 

• void setNonConstantMatrix ()

Say that matrix problem is variable.

• void setLinearSolver (Iteration s=DIRECT\_SOLVER, Preconditioner p=DIAG\_PREC)

Set linear solver data.

void setMaxIter (int max\_it)

Set maximal number of iterations.

• void setTolerance (real\_t toler)

Set tolerance value for convergence.

real\_t runOneTimeStep ()

Run one time step.

• void run (bool opt=false)

Run the time stepping procedure.

• size\_t getNbEq () const

Return number of equations.

LinearSolver < real\_t > & getLSolver ()

Return LinearSolver instance.

real\_t getTimeDerivative (int i=1) const

Get time derivative of solution.

void getTimeDerivative (Vect< real\_t > &y) const

Get time derivative of solution (for a system)

• real\_t get () const

Return solution in the case of a scalar equation.

# 7.82.1 Detailed Description

To solve a system of ordinary differential equations.

The class ODESolver enables solving by a numerical scheme a system or ordinary differential equations taking one of the forms:

• A linear system of differential equations of the first-order:

```
A_1(t)u'(t) + A_0(t)u(t) = f(t)
```

• A linear system of differential equations of the second-order:

```
A_2(t)u''(t) + A_1(t)u'(t) + A_0(t)u(t) = f(t)
```

A system of ordinary differential equations of the form:
 u'(t) = f(t,u(t))

The following time integration schemes can be used:

- Forward Euler scheme (value: FORWARD\_EULER) for first-order systems
- Backward Euler scheme (value: BACKWARD\_EULER) for first-order linear systems
- Crank-Nicolson (value: CRANK\_NICOLSON) for first-order linear systems
- Heun (value: *HEUN*) for first-order systems
- 2nd Order Adams-Bashforth (value: AB2) for first-order systems
- 4-th order Runge-Kutta (value: RK4) for first-order systems
- 2nd order Backward Differentiation Formula (value: BDF2) for linear first-order systems
- Newmark (value: NEWMARK) for linear second-order systems with constant matrices

### Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# 7.82.2 Constructor & Destructor Documentation

ODESolver ( TimeScheme s, real\_t  $time\_step$  = theTimeStep, real\_t  $final\_time$  = theFinalTime, size\_t  $nb\_eq = 1$  )

Constructor using time discretization data.

### **Parameters**

in	S	Choice of the scheme: To be chosen in the enumerated variable <i>Scheme</i> (see the presentation of the class)
in	time_step	Value of the time step. This value will be modified if an adaptive method is used. The default value for this parameter if the value given by the global variable theTimeStep
in	in final_time Value of the final time (time starts at 0). The default value for thi is the value given by the global variable theFinalTime	
in	nb_eq	Number of differential equations (size of the system) [Default: 1]

# 7.82.3 Member Function Documentation

void set ( TimeScheme s, real\_t time\_step = theTimeStep, real\_t final\_time = theFinalTime )

Define data of the differential equation or system.

#### **Parameters**

in	S	Choice of the scheme: To be chosen in the enumerated variable <i>Scheme</i> (see the presentation of the class)	
in	time_step	Value of the time step. This value will be modified if an adaptive method is used. The default value for this parameter if the value given by the global variable theTimeStep	
in	final_time	Value of the final time (time starts at 0). The default value for this parameter is the value given by the global variable theFinalTime	

# void setNbEq ( size\_t nb\_eq )

Set the number of equations [Default: 1].

This function is to be used if the default constructor was used

# void setCoef ( real\_t a0, real\_t a1, real\_t a2, real\_t f )

Define coefficients in the case of a scalar differential equation.

This function enables giving coefficients of the differential equation as an algebraic expression of time t (see the function fparse)

in	a0	Coefficient of the 0-th order term
in	a1	Coefficient of the 1-st order term
in	a2	Coefficient of the 2-nd order term

|--|

#### Note

Naturally, the equation is of the first order if a2=0

### void setCoef ( string a0, string a1, string a2, string f )

Define coefficients in the case of a scalar differential equation.

#### Parameters

in	a0	Coefficient of the 0-th order term
in	a1	Coefficient of the 1-st order term
in	a2	Coefficient of the 2-nd order term
in	f	Value of the right-hand side

#### Note

Naturally, the equation if of the first order if a2=0

### void setLinear ( )

Claim that ODE is linear.

Claim that the defined ODE (or system of ODEs) is linear

### void setF ( string f )

Set time derivative, given as an algebraic expression, for a nonlinear ODE.

This function enables prescribing the value of the 1-st derivative for a 1st order ODE or the 2nd one for a 2nd-order ODE. It is to be used for nonlinear ODEs of the form y'(t) = f(t,y(t)) or y''(t) = f(t,y(t),y'(t))

In the case of a system of ODEs, this function can be called once for each equation, given in the order of the unknowns

# Parameters

in	f	Expression of the function

### void setF ( string f, int i )

Set time derivative, given as an algebraic expression, for a nonlinear ODE.

This function enables prescribing the value of the 1-st derivative for a 1st order ODE or the 2nd one for a 2nd-order ODE. It is to be used for nonlinear ODEs of the form y'(t) = f(t,y(t)) or y''(t) = f(t,y(t),y'(t))

This function is to be used for the i-th equation of a system of ODEs

in	f	Expression of the function	
in	i	Index of equation. Must be not larger than the number of equation	

### void setDF ( string df )

Set time derivative of the function defining the ODE.

This function enables prescribing the value of the 1-st derivative for a 1st order ODE or the 2nd one for a 2nd-order ODE. It is to be used for nonlinear ODEs of the form y'(t) = f(t,y(t)) or y''(t) = f(t,y(t),y'(t))

In the case of a system of ODEs, this function can be called once for each equation, given in the order of the unknowns

# void setDF ( string df, int i )

Set time derivative with respect to the unknown of the function defining the ODE.

This function enables prescribing the value of the 1-st derivative for a 1st order ODE or the 2nd one for a 2nd-order ODE. It is to be used for nonlinear ODEs of the form y'(t) = f(t,y(t)) or y''(t) = f(t,y(t),y'(t))

This function is to be used for the i-th equation of a system of ODEs

#### **Parameters**

in	df	Expression of time derivative of the function	
in	i	Index of equation. Must be not larger than the number of equations	

# void setRK4RHS ( real\_t f )

Set intermediate right-hand side vector for the Runge-Kutta method.

#### Parameters

in	f	Value of right-hand side
	J	

### void setRK4RHS ( Vect< real\_t > & f )

Set intermediate right-hand side vector for the Runge-Kutta method.

### **Parameters**

in	f	right-hand side vector
----	---	------------------------

### void setInitial ( Vect< real\_t > & u )

Set initial condition for a first-oder system of differential equations.

in	и	Vector containing initial condition for the unknown	
----	---	---	--

### void setInitial ( real\_t u, int i )

Set initial condition for a first-oder system of differential equations.

#### **Parameters**

in	и	Initial condition for an unknown
in	i	Index of the unknown

# void setInitial ( Vect< real\_t > & u, Vect< real\_t > & v)

Set initial condition for a second-order system of differential equations.

Giving the right-hand side at initial time is somtimes required for high order methods like Runge-Kutta

#### Parameters

in	и	Vector containing initial condition for the unknown
in	v	Vector containing initial condition for the time derivative of the unknown

# void setInitialRHS ( $Vect < real_t > & f$ )

Set initial RHS for a system of differential equations.

Giving the right-hand side at initial time is somtimes required for high order methods like Runge-Kutta

### Parameters

in	f	Vector containing right-hand side at initial time. This vector is helpful for high order
		methods

# void setInitial ( real\_t u, real\_t v )

Set initial condition for a second-order ordinary differential equation.

in	и	Initial condition (unknown) value
in	v	Initial condition (time derivative of the unknown) value

### void setInitial ( real\_t u )

Set initial condition for a first-order ordinary differential equation.

#### **Parameters**

in u	Initial condition (unknown) value
------	-----------------------------------

### void setInitialRHS ( real\_t f )

Set initial right-hand side for a single differential equation.

#### Parameters

in	f	Value of right-hand side at initial time. This value is helpful for high order methods
----	---	--

### void setMatrices ( DMatrix < real\_t > & A0, DMatrix < real\_t > & A1 )

Define matrices for a system of first-order ODEs.

Matrices are given as references to class DMatrix.

#### **Parameters**

in	A0	Reference to matrix in front of the 0-th order term (no time derivative)
in	A1	Reference to matrix in front of the 1-st order term (first time derivative)

### Remarks

This function has to be called at each time step

# void setMatrices ( DMatrix< real\_t > & A0, DMatrix< real\_t > & A1, DMatrix< real\_t > & A2 )

Define matrices for a system of second-order ODEs.

Matrices are given as references to class DMatrix.

### Parameters

in	A0	Reference to matrix in front of the 0-th order term (no time derivative)
in	A1	Reference to matrix in front of the 1-st order term (first time derivative)
in	A2	Reference to matrix in front of the 2-nd order term (second time derivative)

### Remarks

This function has to be called at each time step

### void seODEVectors ( Vect< real\_t > & a0, Vect< real\_t > & a1 )

Define matrices for an implicit nonlinear system of first-order ODEs.

The system has the nonlinear implicit form a1(u)' + a0(u) = 0 Vectors a0, a1 are given as references to class Vect.

#### Parameters

in	a0	Reference to vector in front of the 0-th order term (no time derivative)
in	a1	Reference to vector in front of the 1-st order term (first time derivative)

#### Remarks

This function has to be called at each time step

# void seODEVectors ( Vect< real\_t > & a0, Vect< real\_t > & a1, Vect< real\_t > & a2)

Define matrices for an implicit nonlinear system of second-order ODEs.

The system has the nonlinear implicit form a2(u)'' + a1(u)' + a0(u) = 0 Vectors a0, a1, a2 are given as references to class Vect.

#### **Parameters**

in	a0	Reference to vector in front of the 0-th order term (no time derivative)
in	a1	Reference to vector in front of the 1-st order term (first time derivative)
in	a2	Reference to vector in front of the 2-nd order term (second time derivative)

### Remarks

This function has to be called at each time step

### void setRHS ( Vect< real\_t > & b )

Set right-hand side vector for a system of ODE.

### **Parameters**

in	b	Vect instance containing right-hand side for a linear system of ordinary differential
		equations

# void setRHS ( real\_t f )

Set right-hand side for a linear ODE.

# Parameters

### void setNewmarkParameters ( real\_t beta, real\_t gamma )

Define parameters for the Newmarxk scheme.

in	beta	Parameter beta [Default: 0.25]
in	gamma	Parameter gamma [Default: 0.5]

### void setConstantMatrix ( )

Say that matrix problem is constant.

This is useful if the linear system is solved by a factorization method but has no effect otherwise

#### void setNonConstantMatrix ( )

Say that matrix problem is variable.

This is useful if the linear system is solved by a factorization method but has no effect otherwise

# void setLinearSolver ( Iteration $s = DIRECT\_SOLVER$ , Preconditioner $p = DIAG\_PREC$ )

Set linear solver data.

#### **Parameters**

in	S	Solver identification parameter. To be chosen in the enumeration variable Iteration: DIRECT_SOLVER, CG_SOLVER, CGS_SOLVER, BICG_SOLVER, BICG_STAB_SOLVER, GMRES_SOLVER, QMR_SOLVER [Default: DIRECT_SOLVER]
in	р	Preconditioner identification parameter. To be chosen in the enumeration variable Preconditioner:  IDENT_PREC, DIAG_PREC, ILU_PREC [Default: DIAG_PREC]

### Note

The argument *p* has no effect if the solver is DIRECT\_SOLVER

### void setMaxIter ( int max\_it )

Set maximal number of iterations.

This function is useful for a non linear ODE (or system of ODEs) if an implicit scheme is used

# Parameters

in	max⇔	Maximal number of iterations [Default: 100]
	_it	

# void setTolerance ( real\_t toler )

Set tolerance value for convergence.

This function is useful for a non linear ODE (or system of ODEs) if an implicit scheme is used

	in	toler	Tolerance value [Default: 1.e-8]
--	----	-------	----------------------------------

# real\_t runOneTimeStep ( )

Run one time step.

#### Returns

Value of new time step if this one is updated

### void run ( bool opt = false )

Run the time stepping procedure.

#### Parameters

in	opt	Flag to say if problem matrix is constant while time stepping (true) or not (Default
		value is false)

### Note

This argument is not used if the time stepping scheme is explicit

# real\_t getTimeDerivative ( int i = 1 ) const

Get time derivative of solution.

Return approximate time derivative of solution in the case of a single equation

### Parameters

in	i	Index of component whose time derivative is sought
		I

### Returns

Time derivative of the i-th component of the solution

### Remarks

If we are solving one equation, this parameter is not used.

# void getTimeDerivative ( Vect< real\_t > & y ) const

Get time derivative of solution (for a system)

Get approximate time derivative of solution in the case of an ODE system

out	y	Vector containing time derivative of solution
-----	---	---

# 7.83 OFELIException Class Reference

To handle exceptions in OFELI. Inherits runtime\_error.

### **Public Member Functions**

• OFELIException (const std::string &s)

This form will be used most often in a throw.

• OFELIException ()

Throw with no error message.

# 7.83.1 Detailed Description

To handle exceptions in OFELI.

This class enables using exceptions in programs using OFELI

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

# 7.84 OptSolver Class Reference

To solve an optimization problem with bound constraints.

# **Public Types**

```
    enum OptMethod {
        GRADIENT = 0,
        TRUNCATED_NEWTON = 1,
        SIMULATED_ANNEALING = 2,
        NELDER_MEAD = 3 }
```

 $Choose\ optimization\ algorithm.$ 

### **Public Member Functions**

• OptSolver ()

Default constructor.

• OptSolver (Vect< real\_t > &x)

Constructor using vector of optimization variables.

OptSolver (MyOpt &opt, Vect< real\_t > &x)

Constructor using vector of optimization variables.

• ∼OptSolver ()

Destructor.

• int getNbFctEval () const

Return the total number of function evaluations.

void setOptMethod (OptMethod m)

Choose optimization method.

• void setBC (const Vect< real\_t > &bc)

Prescribe boundary conditions as constraints.

• void setObjective (string exp)

Define the objective function to minimize by an algebraic expression.

• void setGradient (string exp, int i=1)

Define a component of the gradient of the objective function to minimize by an algebraic expression.

• void setOptClass (MyOpt &opt)

Choose user defined optimization class.

void setUpperBound (real\_t ub)

Define upper bound for optimization variable.

void setUpperBounds (Vect< real\_t > &ub)

Define upper bounds for optimization variables.

void setLowerBound (real\_t lb)

Define lower bound for optimization variable.

• void setVerbosity (int verb)

Set verbosity parameter.

void setLowerBounds (Vect< real\_t > &lb)

Define lower bounds for optimization variables.

 void setSAOpt (real\_t rt, int ns, int nt, int &neps, int maxevl, real\_t t, Vect< real\_t > &vm, Vect< real\_t > &xopt, real\_t &fopt)

Set Simulated annealing options.

void setTolerance (real\_t toler)

Set error tolerance.

void setMaxIterations (int n)

Set maximal number of iterations.

• int getNbObjEval () const

Return number of objective function evaluations.

• real\_t getTemperature () const

Return the final temperature.

• int getNbAcc () const

Return the number of accepted objective function evaluations.

• int getNbOutOfBounds () const

Return the total number of trial function evaluations that would have been out of bounds.

real\_t getOptObj () const

Return Optimal value of the objective.

• int run ()

Run the optimization algorithm.

int run (real\_t toler, int max\_it)

Run the optimization algorithm.

real\_t getSolution () const

Return solution in the case of a one variable optimization.

void getSolution (Vect< real\_t > &x) const

Get solution vector.

### **Friends**

• ostream & operator<< (ostream &s, const OptSolver &os)

Output class information.

# 7.84.1 Detailed Description

To solve an optimization problem with bound constraints.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

### 7.84.2 Member Enumeration Documentation

# enum OptMethod

Choose optimization algorithm.

Enumerator

GRADIENT Gradient method

TRUNCATED\_NEWTON Truncated Newton method

SIMULATED\_ANNEALING Simulated annealing global optimization method

NELDER\_MEAD Nelder-Mead global optimization method

### 7.84.3 Constructor & Destructor Documentation

# OptSolver ( Vect< real\_t > & x )

Constructor using vector of optimization variables.

#### **Parameters**

i	n	х	Vector having as size the number of optimization variables. It contains the initial	
			guess for the optimization algorithm.	

### Remarks

After using the member function run, the vector x contains the obtained solution if the optimization procedure was successful

### OptSolver ( MyOpt & opt, Vect < real.t > & x )

Constructor using vector of optimization variables.

in	opt	Reference to instance of user defined optimization class. This class inherits from abstract class MyOpt. It must contain the member function double Objective(Vect <double> &amp;x) which returns the value of the objective for a given solution vector x. The user defined class must contain, if the optimization algorithm requires it the member function Gradient(Vect<double> &amp;x, Vect<double> &amp;g) which stores the gradient of the objective in the vector g for a given optimization vector x. The user defined class must also contain, if the optimization algorithm requires it the member function</double></double></double>
in	x	Vector having as size the number of optimization variables. It contains the initial guess for the optimization algorithm.

#### Remarks

After using the member function run, the vector x contains the obtained solution if the optimization procedure was successful

### 7.84.4 Member Function Documentation

# void setOptMethod ( OptMethod m )

Choose optimization method.

### Parameters

in	т	Enumerated value to choose the optimization algorithm to use. Must be chosen among the enumerated values:  • GRADIENT: Gradient steepest descent method with projection for bounded constrained problems
		• TRUNCATED_NEWTON: The Nash's Truncated Newton Algorithm, due to S.G. Nash (Newton-type Minimization via the Lanczos method, SIAM J. Numer. Anal. 21 (1984) 770-778).
		• SIMULATED_ANNEALING: Global optimization simulated annealing method. See Corana et al.'s article: "Minimizing Multimodal Functions of Continuous Variables with the Simulated Annealing Algorithm" in the September 1987 (vol. 13, no. 3, pp. 262-280) issue of the ACM Transactions on Mathematical Software.
		<ul> <li>NELDER_MEAD: Global optimization Nelder-Mead method due to John Nelder, Roger Mead (A simplex method for function minimization, Computer Journal, Volume 7, 1965, pages 308-313). As implemented by R. ONeill (Algorithm AS 47: Function Minimization Using a Simplex Procedure, Applied Statistics, Volume 20, Number 3, 1971, pages 338-345).</li> </ul>

# void setBC ( const Vect< real\_t > & bc )

Prescribe boundary conditions as constraints.

This member function is useful in the case of optimization problems where the optimization variable vector is the solution of a partial differential equation. For this case, Dirichlet boundary

conditions can be prescribed as constraints for the optimization problem

#### **Parameters**

in	bc	Vector containing the values to impose on degrees of freedom. This vector must
		have been constructed using the Mesh instance.

### Remarks

Only degrees of freedom with positive code are taken into account as prescribed

# void setObjective ( string exp )

Define the objective function to minimize by an algebraic expression.

### Parameters

in   <i>exp</i>   Regular expression defining the objective fund
--

# void setGradient ( string exp, int i = 1 )

Define a component of the gradient of the objective function to minimize by an algebraic expression.

### **Parameters**

in	ехр	Regular expression defining the objective function
in	i	Component of gradient [Default: 1]

# void setOptClass ( MyOpt & opt )

Choose user defined optimization class.

### Parameters

in	opt	Reference to inherited user specified optimization class
----	-----	--

# void setUpperBound ( real\_t ub )

Define upper bound for optimization variable. Case of a one-variable problem

in	иb	Upper bound
	110	opper bound

### void setUpperBounds ( Vect< real\_t > & ub )

Define upper bounds for optimization variables.

#### Parameters

	in	иb	Vector containing upper values for variables	
--	----	----	--	--

#### void setLowerBound ( real\_t lb )

Define lower bound for optimization variable. Case of a one-variable problem

#### Parameters

in <i>lb</i>	Lower value
--------------	-------------

# void setVerbosity ( int verb )

Set verbosity parameter.

# Parameters

in	verb	Verbosity parameter
----	------	---------------------

# void setLowerBounds ( Vect< real\_t > & lb )

Define lower bounds for optimization variables.

### Parameters

in	lb	Vector containing lower values for variables	

void setSAOpt ( real\_t rt, int ns, int nt, int & neps, int maxevl, real\_t t, Vect< real\_t > & vm, Vect< real\_t > & xopt, real\_t & fopt )

Set Simulated annealing options.

### Remarks

This member function is useful only if simulated annealing is used.

in	rt	The temperature reduction factor. The value suggested by Corana et al. is .85. See Goffe et al. for more advice.
in	ns	Number of cycles. After <i>ns*nb_var</i> function evaluations, each element of <i>vm</i> is adjusted so that approximately half of all function evaluations are accepted. The suggested value is 20.

in	nt	Number of iterations before temperature reduction. After $nt*ns*n$ function evaluations, temperature (t) is changed by the factor $rt$ . Value suggested by Corana et al. is $max(100,5*nb\_var)$ . See Goffe et al. for further advice.
in	neps	Number of final function values used to decide upon termination. See eps. Suggested value is 4
in	maxevl	The maximum number of function evaluations. If it is exceeded, the return <i>code</i> =1.
in	t	The initial temperature. See Goffe et al. for advice.
in	vm	The step length vector. On input it should encompass the region of interest given the starting value $x$ . For point $x[i]$ , the next trial point is selected is from $x[i]$ -vm[i] to $x[i]$ +vm[i]. Since $vm$ is adjusted so that about half of all points are accepted, the input value is not very important (i.e. is the value is off, $OptimSA$ adjusts $vm$ to the correct value).
out	xopt	optimal values of optimization variables
out	fopt	Optimal value of objective

### void setTolerance ( real\_t toler )

Set error tolerance.

#### **Parameters**

in	toler	Error tolerance for termination. If the final function values from the last neps
		temperatures differ from the corresponding value at the current temperature by
		less than eps and the final function value at the current temperature differs from
		the current optimal function value by less than toler, execution terminates and
		the value $0$ is returned.

# real\_t getTemperature ( ) const

Return the final temperature.

This function is meaningful only if the Simulated Annealing algorithm is used

# int getNbAcc ( ) const

Return the number of accepted objective function evaluations.

This function is meaningful only if the Simulated Annealing algorithm is used

# int getNbOutOfBounds ( ) const

Return the total number of trial function evaluations that would have been out of bounds. This function is meaningful only if the Simulated Annealing algorithm is used

### int run ( )

Run the optimization algorithm.

This function runs the optimization procedure using default values for parameters. To modify these values, user the function run with arguments

#### int run ( real\_t toler, int max\_it )

Run the optimization algorithm.

#### **Parameters**

in	toler	Tolerance value for convergence testing	
in	max←	Maximal number of iterations to achieve convergence	
	_it		

# real\_t getSolution ( ) const

Return solution in the case of a one variable optimization.

In the case of a one variable problem, the solution value is returned, if the optimization procedure was successful

# void getSolution ( Vect< real\_t > & x ) const

Get solution vector.

The vector *x* contains the solution of the optimization problem. Note that if the constructor using an initial vector was used, the vector will contain the solution once the member function run has beed used (If the optimization procedure was successful)

#### **Parameters**

out x	solution vector
-------	-----------------

# 7.85 Partition Class Reference

To partition a finite element mesh into balanced submeshes.

### **Public Member Functions**

• Partition ()

Default constructor.

• Partition (Mesh &mesh, size\_t n)

Constructor to partition a mesh into submeshes.

Partition (Mesh &mesh, int n, vector< int > &epart)

Constructor using already created submeshes.

• ∼Partition ()

Destructor.

size\_t getNbSubMeshes () const

Return number of submeshes.

size\_t getNbNodes (size\_t i) const

Return number of nodes in given submesh.

size\_t getNbElements (size\_t i) const

Return number of elements in given submesh.

Mesh \* getMesh ()

Return the global Mesh instance.

Mesh \* getMesh (size\_t i)

Return the submesh of label i

• size\_t getNodeLabelInSubMesh (size\_t sm, size\_t label) const

Return node label in subdomain by giving its label in initial mesh.

• size\_t getElementLabelInSubMesh (size\_t sm, size\_t label) const

Return element label in subdomain by giving its label in initial mesh.

• size\_t getNodeLabelInMesh (size\_t sm, size\_t label) const

Return node label in initial mesh by giving its label in submesh.

size\_t getElementLabelInMesh (size\_t sm, size\_t label) const

Return element label in initial mesh by giving its label in submesh.

size\_t getNbInterfaceSides (size\_t sm) const

Return Number of interface sides for a given sub-mesh.

size\_t getSubMesh (size\_t sm, size\_t i) const

Return index of submesh that contains the *i*-th side label in sub-mesh sm

Mesh & getSubMesh (size\_t i) const

Return reference to submesh.

• size\_t getFirstSideLabel (size\_t sm, size\_t i) const

Return i-th side label in a given submesh.

• size\_t getSecondSideLabel (size\_t sm, size\_t i) const

Return side label in the neighbouring submesh corresponding to i-th side label in sub-mesh sm

• int getNbConnectInSubMesh (int n, int s) const

Get number of connected nodes in a submesh.

int getNbConnectOutSubMesh (int n, int s) const

Get number of connected nodes out of a submesh.

• void put (size\_t n, string file) const

Save a submesh in file.

void set (Mesh &mesh, size\_t n)

Set Mesh instance.

## **Friends**

• ostream & operator << (ostream &s, const Partition &p)

Output class information.

#### 7.85.1 Detailed Description

To partition a finite element mesh into balanced submeshes.

Class Partition enables partitioning a given mesh into a given number of submeshes with a minimal connectivity. Partition uses the well known metis library that is included in the OFELI library. A more detailed description of metis can be found in the web site:

http://www.csit.fsu.edu/~burkardt/c\_src/metis/metis.html

Author

Rachid Touzani

Copyright

GNU Lesser Public License

## 7.85.2 Constructor & Destructor Documentation

#### Partition ( Mesh & mesh, size\_t n )

Constructor to partition a mesh into submeshes.

#### **Parameters**

in	mesh	Mesh instance
in	n	Number of submeshes

## Partition ( Mesh & mesh, int n, vector < int > & epart )

Constructor using already created submeshes.

#### Parameters

j	in	mesh	Mesh instance	
j	in	n	Number of submeshes	
j	in	epart	Vector containing for each element its submesh label (Running from 0 to n-1	

### 7.85.3 Member Function Documentation

## size\_t getNodeLabelInSubMesh ( size\_t sm, size\_t label ) const

Return node label in subdomain by giving its label in initial mesh.

### Parameters

in	sm	Label of submesh
in	label	Label of node in initial mesh

## size\_t getNodeLabelInMesh ( size\_t sm, size\_t label ) const

Return node label in initial mesh by giving its label in submesh.

### Parameters

in	sm	Label of submesh
in	label	Node label

## $size_t getSubMesh ( size_t sm, size_t i ) const$

Return index of submesh that contains the i-th side label in sub-mesh sm

#### **Parameters**

in	sm	Submesh index
in	i	Side label

## Returns

Index of submesh

## Mesh& getSubMesh ( size\_t i ) const

Return reference to submesh.

#### Parameters

in	i	Submesh index

#### Returns

Reference to corresponding Mesh instance

## size\_t getFirstSideLabel ( size\_t sm, size\_t i ) const

Return i-th side label in a given submesh.

#### **Parameters**

in	sm	Index of submesh
in	i	Label of side

## size\_t getSecondSideLabel ( size\_t sm, size\_t i ) const

Return side label in the neighbouring submesh corresponding to i-th side label in sub-mesh sm

#### Parameters

in	sm	Label of submesh
in	i	Side label

## int getNbConnectInSubMesh ( int n, int s ) const

Get number of connected nodes in a submesh.

#### Parameters

in	n	Label of node for which connections are counted
in	S	Label of submesh (starting from 0)

## int getNbConnectOutSubMesh ( int n, int s ) const

Get number of connected nodes out of a submesh.

#### **Parameters**

in	n	Label of node for which connections are counted
in	S	Label of submesh (starting from 0)

## void put ( size\_t n, string file ) const

Save a submesh in file.

#### **Parameters**

in	n	Label of submesh
in	file	Name of file in which submesh is saved

## 7.86 Penta6 Class Reference

Defines a 6-node pentahedral finite element using  $P_1$  interpolation in local coordinates (s.x,s.y) and  $Q_1$  isoparametric interpolation in local coordinates (s.x,s.z) and (s.y,s.z). Inheritance diagram for Penta6:



#### **Public Member Functions**

• Penta6 ()

Default Constructor.

• Penta6 (const Element \*element)

Constructor when data of Element el are given.

• ~Penta6 ()

Destructor.

void set (const Element \*el)

Choose element by giving its pointer.

• void setLocal (const Point< real\_t > &s)

*Initialize local point coordinates in element.* 

• vector< Point< real\_t >> DSh () const

Return partial derivatives of shape functions of element nodes.

• real\_t getMaxEdgeLength () const

Return Maximum length of pentahedron edges.

• real\_t getMinEdgeLength () const

Return Mimimum length of pentahedron edges.

## 7.86.1 Detailed Description

Defines a 6-node pentahedral finite element using  $P_1$  interpolation in local coordinates (s.x,s.y) and  $Q_1$  isoparametric interpolation in local coordinates (s.x,s.z) and (s.y,s.z).

The reference element is the cartesian product of the standard reference triangle with the line [-1,1]. The nodes are ordered as follows: Node 1 in reference element is at s=(1,0,0) Node 2 in reference element is at s=(0,1,0) Node 3 in reference element is at s=(0,0,0) Node 4 in reference element is at s=(0,0,0) Node 5 in reference element is at s=(0,0,0) Node 6 in reference element is at s=(0,0,0)

The user must take care to the fact that determinant of jacobian and other quantities depend on the point in the reference element where they are calculated. For this, before any utilization of shape functions or jacobian, function **setLocal()** must be invoked.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

#### 7.86.2 Constructor & Destructor Documentation

Penta6 (const Element \* element)

Constructor when data of Element el are given.

**Parameters** 

in	element	Pointer to Element
----	---------	--------------------

#### 7.86.3 Member Function Documentation

void setLocal ( const Point< real $_{-}$ t > & s )

Initialize local point coordinates in element.

#### **Parameters**

in	S	Point in the reference element This function computes jacobian, shape functions and
		their partial derivatives at s. Other member functions only return these values.

### $vector < Point < real_t > > DSh ( ) const$

Return partial derivatives of shape functions of element nodes.

#### Returns

LocalVect instance of partial derivatives of shape functions e.g. dsh(i).x, dsh(i).y, are partial derivatives of the i-th shape function.

Note

The local point at which the derivatives are computed must be chosen before by using the member function setLocal

# 7.87 PhaseChange Class Reference

This class enables defining phase change laws for a given material.

#### **Public Member Functions**

virtual ~PhaseChange ()

Destructor.

• int E2T (real\_t &H, real\_t &T, real\_t &gamma)

Calculate temperature from enthalpy.

virtual int EnthalpyToTemperature (real\_t &H, real\_t &T, real\_t &gamma)

Virtual function to calculate temperature from enthalpy.

• void setMaterial (Material &m, int code)

Choose Material instance and material code.

• Material & getMaterial () const

Return reference to Material instance.

## 7.87.1 Detailed Description

This class enables defining phase change laws for a given material.

These laws are predefined for a certain number of materials. The user can set himself a specific behavior for his own materials by defining a class that inherits from PhaseChange. The derived class must has at least the member function

int EnthalpyToTemperature(real\_t &H, real\_t &T, real\_t &gamma)

#### 7.87.2 Member Function Documentation

int E2T ( real\_t & H, real\_t & T, real\_t & gamma )

Calculate temperature from enthalpy.

This member function is to be called in any equation class that needs phase change laws.

#### **Parameters**

in	Н	Enthalpy value
out	T	Calculated temperature value
out	gamma	Maximal slope of the curve H -> T

### virtual int EnthalpyToTemperature ( real\_t & H, real\_t & T, real\_t & gamma ) [virtual]

Virtual function to calculate temperature from enthalpy.

This member function must be implemented in any derived class in order to define user's own material laws.

#### **Parameters**

in	Н	Enthalpy value
out	T	Calculated temperature value
out	gamma	Maximal slope of the curve H -> T

# 7.88 Point< T\_> Class Template Reference

Defines a point with arbitrary type coordinates.

#### **Public Member Functions**

```
• Point ()
```

Default constructor.

• Point (T<sub>-</sub> a, T<sub>-</sub> b=T<sub>-</sub>(0), T<sub>-</sub> c=T<sub>-</sub>(0))

Constructor that assigns a, b to x-, y- and z-coordinates respectively.

• Point (const Point < T\_ > &p)

Copy constructor.

• T<sub>-</sub> & operator() (size<sub>-</sub>t i)

Operator (): Non constant version.

const T<sub>-</sub> & operator() (size<sub>-</sub>t i) const

Operator (): Constant version.

• T<sub>-</sub> & operator[] (size<sub>-</sub>t i)

Operator []: Non constant version.

• const T<sub>-</sub> & operator[] (size<sub>-</sub>t i) const

Operator []: Constant version.

• Point $< T_- > & operator += (const Point < T_- > &p)$ 

Operator +=

• Point< T $_->$  & operator=(const Point< T $_->$  &p)

Operator -=

• Point< T\_> & operator= (const T\_ &a)

Operator =

• Point< T<sub>−</sub> > & operator+= (const T<sub>−</sub> &a)

Operator +=

• Point< T\_> & operator== (const T\_ &a)

Operator -=

• Point< T\_> & operator\*= (const T\_ &a)

Operator \*=

• Point< T\_> & operator/= (const T\_ &a)

Operator /=

• bool operator== (const Point< T\_> &p)

Operator ==

• bool operator!= (const Point< T\_> &p)

Operator !=

• double NNorm () const

Return squared euclidean norm of vector.

• double Norm () const

Return norm (length) of vector.

• void Normalize ()

Normalize vector.

• Point< double > Director (const Point< double > &p) const

Return Director (Normalized vector)

• bool isCloseTo (const Point< double > &a, double toler=OFELI\_TOLERANCE) const

Return true if current point is close to instance a (up to tolerance toler)

• T\_ operator, (const Point< T\_ > &p) const

Return Dot (scalar) product of two vectors.

#### **Public Attributes**

• T\_x

First coordinate.

• T\_ **y** 

Second coordinate.

• T<sub>-</sub>z

Third coordinate.

## 7.88.1 Detailed Description

```
template < class T_> class OFELI::Point < T_>
```

Defines a point with arbitrary type coordinates.

Operators = and () are overloaded.

**Template Parameters** 

```
T_{\leftarrow} Data type (double, float, complex<double>, ...)
```

Author

Rachid Touzani

Copyright

GNU Lesser Public License

### 7.88.2 Constructor & Destructor Documentation

Point (
$$T_-a$$
,  $T_-b = T_-(0)$ ,  $T_-c = T_-(0)$ )

Constructor that assigns a, b to x-, y- and z-coordinates respectively.

Default values for b and c are 0

#### 7.88.3 Member Function Documentation

```
T_{-}% operator() ( size_t i )
Operator (): Non constant version.
   Values i = 1, 2, 3 correspond to x, y and z respectively
const T_& operator() ( size_t i ) const
Operator (): Constant version.
   Values i = 1, 2, 3 correspond to x, y and z respectively
T_{-} operator[]( size_t i)
Operator []: Non constant version.
   Values i = 0, 1, 2 correspond to x, y and z respectively
const T_& operator[]( size_t i ) const
Operator []: Constant version.
   Values i = 0, 1, 2 correspond to x, y and z respectively
Point<T_->& operator+= ( const Point< T_->& p )
Operator +=
   Add point p to current instance
Point<T_->& operator= ( const Point< T_-> & p )
Operator -=
   Subtract point p from current instance
Point<T_->& operator= ( const T_- & a )
Operator =
   Assign constant a to current instance coordinates
Point<T_->& operator+= ( const T_-& a )
Operator +=
   Add constant a to current instance coordinates
Point<T_->& operator== ( const T_- & a )
Operator -=
   Subtract constant a from current instance coordinates
Point<T_->& operator*= ( const T_-& a )
Operator *=
   Multiply constant a by current instance coordinates
Point<T_->& operator/= ( const T_- & a )
Operator /=
   Divide current instance coordinates by a
```

#### bool operator== ( const Point $< T_- > & p$ )

Operator ==

Return true if current instance is equal to p, false otherwise.

### bool operator!= ( const Point $< T_- > & p$ )

Operator !=

Return false if current instance is equal to p, true otherwise.

### void Normalize ( )

Normalize vector.

Divide vector components by its 2-norm

#### bool isCloseTo ( const Point < double > & a, double toler = OFELI\_TOLERANCE ) const

Return true if current point is close to instance a (up to tolerance toler)

Default value for toler is the OFELI\_TOLERANCE constant.

#### $T_-$ operator, (const Point< $T_-$ > & p) const

Return Dot (scalar) product of two vectors.

A typical use of this operator is double a = (p,q) where p and q are 2 instances of Point < double >

#### **Parameters**

in *p* Point instance by which the current instance is multiplied

# 7.89 Point2D < T\_ > Class Template Reference

Defines a 2-D point with arbitrary type coordinates.

## **Public Member Functions**

• Point2D ()

Default constructor.

• Point2D (T<sub>-</sub> a, T<sub>-</sub> b=T<sub>-</sub>(0))

Constructor that assigns a, b to x-, y- and y-coordinates respectively.

• Point2D (T\_\*a)

Initialize point coordinates with C-array a.

• Point2D (const Point2D < T\_ > &pt)

Copy constructor.

• Point2D (const Point < T\_ > &pt)

Copy constructor from class Point.

• T<sub>-</sub> & operator() (size<sub>-</sub>t i)

*Operator(): Non constant version.* 

• const T<sub>-</sub> & operator() (size<sub>-</sub>t i) const

Operator(): Constant version.

• T<sub>-</sub> & operator[] (size<sub>-</sub>t i)

```
Operator []: Non constant version.
• const T<sub>-</sub> & operator[] (size<sub>-</sub>t i) const
      Operator [] Constant version.
• Point2D< T_-> & operator= (const Point2D< T_-> &p)
      Operator =
• Point2D< T_-> & operator+= (const Point2D< T_-> &p)
      Operator +=
• Point2D< T_-> & operator= (const Point2D< T_-> &p)
      Operator -=
• Point2D< T_> & operator= (const T_ &a)
      Operator =
• Point2D< T_> & operator+= (const T_ &a)
      Operator +=
• Point2D< T_- > & operator= (const T_- &a)
      Operator -=
• Point2D< T_-> & operator*= (const T_- &a)
      Operator *=
• Point2D< T_-> & operator/= (const T_- &a)
      Operator /=
• bool operator== (const Point2D < T_ > &p)
      Operator ==
• bool operator!= (const Point2D < T_ > &p)
      Operator !=

    real_t CrossProduct (const Point2D< real_t > &lp, const Point2D< real_t > &rp)

      Return Cross product of two vectors lp and rp

    real_t NNorm () const

      Return squared norm (length) of vector.
• real_t Norm () const
      Return norm (length) of vector.
• Point2D< real_t > Director (const Point2D< real_t > &p) const
      Return Director (Normalized vector)

    bool isCloseTo (const Point2D < real_t > &a, real_t toler=OFELI_TOLERANCE) const
```

Return true if current point is close to instance a (up to tolerance toler)

#### **Public Attributes**

• T\_x

First coordinate of point.

• T<sub>-</sub> y

Second coordinate of point.

## 7.89.1 Detailed Description

```
template<class T_> class OFELI::Point2D< T_>
```

Defines a 2-D point with arbitrary type coordinates. Operators = and () are overloaded. The actual **Template Parameters** 

```
T_{\leftarrow} Data type (double, float, complex<double>, ...)
```

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

#### 7.89.2 Constructor & Destructor Documentation

```
Point2D ( T_-a, T_-b = T_-(0) )
```

Constructor that assigns a, b to x-, y- and y-coordinates respectively. Default value for b is 0

#### 7.89.3 Member Function Documentation

```
T_{\&} operator() ( size_t i )
```

Operator(): Non constant version.

Values i = 1,2 correspond to x and y respectively

#### const T\_& operator() ( size\_t i ) const

Operator() : Constant version.

Values i=1,2 correspond to x and y respectively

## $T_{\&}$ operator[]( size\_t i)

Operator[]: Non constant version.

Values i=0,1 correspond to x and y respectively

## const T\_& operator[] ( size\_t i ) const

Operator[] Constant version.

Values i=0,1 correspond to x and y respectively

#### Point2D<T $_->$ & operator= ( const Point2D< T $_->$ & p )

Operator =

Assign point p to current instance

#### Point2D<T $_->$ & operator+= ( const Point2D< T $_->$ & p )

Operator +=

Add point p to current instance

## Point2D<T $_->$ & operator== ( const Point2D< T $_->$ & p )

Operator -=

Subtract point p from current instance

#### Point2D<T $_->$ & operator= ( const T $_-$ & a )

Operator =

Assign constant a to current instance coordinates

### Point2D<T $_->$ & operator+= ( const T $_-$ & a )

Operator +=

Add constant a to current instance coordinates

## Point2D<T $_->$ & operator== ( const T $_-$ & a )

Operator -=

Subtract constant a from current instance coordinates

#### Point2D<T $_>$ & operator\*= ( const T $_$ & a )

Operator \*=

Multiply constant a by current instance coordinates

#### Point2D<T $_->$ & operator/= ( const T $_-$ & a )

Operator /=

Divide current instance coordinates by a

### bool operator== ( const Point2D < T<sub>-</sub> > & p )

Operator ==

Return true if current instance is equal to p, false otherwise.

#### bool operator!= ( const Point2D< T $_->$ & p )

Operator !=

Return false if current instance is equal to p, true otherwise.

# 7.90 Polygon Class Reference

To store and treat a polygonal figure. Inheritance diagram for Polygon:



## **Public Member Functions**

• polygon ()

Default constructor.

• Polygon (const Vect< Point< real\_t >> &v, int code=1)

Constructor.

• void setVertices (const Vect< Point< real\_t >> &v)

Assign vertices of polygon.

- real\_t getSignedDistance (const Point< real\_t > &p) const Return signed distance of a given point from the current polygon.
- Polygon & operator+= (Point< real\_t > a)

*Operator* +=.

• Polygon & operator+= (real\_t a)

*Operator* \*=.

### **Additional Inherited Members**

## 7.90.1 Detailed Description

To store and treat a polygonal figure.

#### 7.90.2 Constructor & Destructor Documentation

Polygon ( const Vect< Point< real t > & v, int code = 1 )

Constructor.

## Parameters

in	υ	Vect instance containing list of coordinates of polygon vertices
in	code	Code to assign to the generated domain (Default value = 1)

#### 7.90.3 Member Function Documentation

void setVertices ( const Vect< Point< real\_t >> & v )

Assign vertices of polygon.

#### Parameters

in   v   Vector containing vertices coordinates in counter clockwise of	order
---	-------

## real\_t getSignedDistance ( const Point< real\_t > & p ) const [virtual]

Return signed distance of a given point from the current polygon.

The computed distance is negative if p lies in the polygon, negative if it is outside, and 0 on its boundary

#### **Parameters**

#### **Parameters**

	in	p	Point <double> instance</double>
--	----	---	----------------------------------

Reimplemented from Figure.

#### Polygon& operator+= ( Point< real\_t > a )

Operator +=.

Translate polygon by a vector a

### Polygon& operator+= ( real\_t a )

Operator \*=.

Scale polygon by a factor a

# 7.91 Prec< T $_->$ Class Template Reference

To set a preconditioner.

#### **Public Member Functions**

• Prec ()

Default constructor.

• Prec (int type)

Constructor that chooses preconditioner.

Prec (const SpMatrix < T<sub>-</sub> > &A, int type=DIAG\_PREC)

Constructor using matrix of the linear system to precondition.

• Prec (const Matrix < T\_ > \*A, int type=DIAG\_PREC)

Constructor using matrix of the linear system to precondition.

• ~Prec ()

Destructor.

• void setType (int type)

Define preconditioner type.

void setMatrix (const Matrix < T<sub>-</sub> > \*A)

*Define pointer to matrix for preconditioning (if this one is abstract)* 

• void setMatrix (const SpMatrix < T\_ > &A)

Define the matrix for preconditioning.

• void solve (Vect<  $T_-> &x$ ) const

Solve a linear system with preconditioning matrix.

• void solve (const Vect<  $T_->$  &b, Vect<  $T_->$  &x) const

Solve a linear system with preconditioning matrix.

• void TransSolve (Vect< T\_> &x) const

Solve a linear system with transposed preconditioning matrix.

• void TransSolve (const Vect< T $_->$  &b, Vect< T $_->$  &x) const

Solve a linear system with transposed preconditioning matrix.

• T\_ & getPivot (size\_t i) const

Return i-th pivot of preconditioning matrix.

## 7.91.1 Detailed Description

template<class T\_> class OFELI::Prec< T\_>

To set a preconditioner.

The preconditioner type is chosen in the constructor

**Template Parameters** 

$< T \leftarrow$	Data type (real_t, float, complex <real_t>,)</real_t>	
_>		

Author

Rachid Touzani

Copyright

GNU Lesser Public License

## 7.91.2 Constructor & Destructor Documentation

## Prec (int type)

Constructor that chooses preconditioner.

### Parameters

type	Preconditioner type:
	<ul> <li>IDENT_PREC: Identity preconditioner (No preconditioning)</li> </ul>
	DIAG_PREC: Diagonal preconditioner
	<ul> <li>DILU_PREC: Diagonal Incomplete factorization preconditioner</li> </ul>
	ILU_PREC: Incomplete factorization preconditioner
	SSOR_PREC: SSOR (Symmetric Successive Over Relaxation) preconditioner
	type

## Prec ( const SpMatrix< $T_-$ > & A, int type = DIAG\_PREC )

Constructor using matrix of the linear system to precondition.

#### **Parameters**

in	A	Matrix to precondition

OFELI's Reference Guide 515

## Parameters

in	type	Preconditioner type:
		IDENT_PREC: Identity preconditioner (No preconditioning)
		DIAG_PREC: Diagonal preconditioner
		DILU_PREC: Diagonal Incomplete factorization preconditioner
		ILU_PREC: Incomplete factorization preconditioner
		SSOR_PREC: SSOR (Symmetric Successive Over Relaxation) preconditioner

## Prec ( const Matrix < T<sub>-</sub> > \* A, int type = DIAG\_PREC )

Constructor using matrix of the linear system to precondition.

#### Parameters

in	A	Pointer to abstract Matrix class to precondition	
in	type	Preconditioner type:	
		<ul> <li>IDENT_PREC: Identity preconditioner (No preconditioning)</li> </ul>	
		DIAG_PREC: Diagonal preconditioner	
		DILU_PREC: Diagonal Incomplete factorization preconditioner	
		ILU_PREC: Incomplete factorization preconditioner	
		SSOR_PREC: SSOR (Symmetric Successive Over Relaxation) preconditioner	

## 7.91.3 Member Function Documentation

## void setType ( int type )

Define preconditioner type.

### Parameters

in	type	Preconditioner type:
		IDENT_PREC: Identity preconditioner (No preconditioning)
		DIAG_PREC: Diagonal preconditioner
		DILU_PREC: Diagonal Incomplete factorization preconditioner
		ILU_PREC: Incomplete factorization preconditioner
		SSOR_PREC: SSOR (Symmetric Successive Over Relaxation) preconditioner

#### 7.91. PREC < T\_ > CLASS TEMPLATE REFERENCE CHAPTER 7. CLASS DOCUMENTATION

#### void setMatrix ( const Matrix $< T_- > *A$ )

Define pointer to matrix for preconditioning (if this one is abstract)

#### **Parameters**

in A	Matrix to precondition
------	------------------------

## void setMatrix ( const SpMatrix $< T_- > & A$ )

Define the matrix for preconditioning.

#### **Parameters**

in	Α	Matrix to precondition (instance of class SpMatrix)	]
----	---	---	---

#### void solve ( Vect< T $_->$ & x ) const

Solve a linear system with preconditioning matrix.

#### **Parameters**

	in,out	x	Right-hand side on input and solution on output.	
--	--------	---	--	--

## void solve ( const Vect< $T_-$ > & b, Vect< $T_-$ > & x ) const

Solve a linear system with preconditioning matrix.

### Parameters

in	b	Right-hand side
out	x	Solution vector

## void TransSolve ( Vect< $T_- > \& x$ ) const

Solve a linear system with transposed preconditioning matrix.

#### Parameters

in,out	х	Right-hand side in input and solution in output.
--------	---	--

## void TransSolve ( const Vect< $T_- > \& b$ , Vect< $T_- > \& x$ ) const

Solve a linear system with transposed preconditioning matrix.

#### **Parameters**

in	b	Right-hand side vector
out	x	Solution vector

# 7.92 Prescription Class Reference

To prescribe various types of data by an algebraic expression. Data may consist in boundary conditions, forces, tractions, fluxes, initial condition. All these data types can be defined through an enumerated variable.

#### **Public Member Functions**

• Prescription ()

Default constructor.

• Prescription (Mesh &mesh, const std::string &file)

Constructor that gives an instance of class Mesh and the data file name.

• ∼Prescription ()

Destructor.

• int get (int type, Vect< real\_t > &v, real\_t time=0, size\_t dof=0)

## 7.92.1 Detailed Description

To prescribe various types of data by an algebraic expression. Data may consist in boundary conditions, forces, tractions, fluxes, initial condition. All these data types can be defined through an enumerated variable.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

#### 7.92.2 Constructor & Destructor Documentation

Prescription ( Mesh & mesh, const std::string & file )

Constructor that gives an instance of class Mesh and the data file name. It reads parameters in Prescription Format from this file.

#### Parameters

in	mesh	Mesh instance
in	file	Name of Prescription file

## 7.92.3 Member Function Documentation

int get ( int type, Vect< real\_t > & v, real\_t time = 0, size\_t dof = 0 )

Read data in the given file and stores in a Vect instance for a chosen DOF. The input value type determines the type of data to read.

#### **Parameters**

in	type	Type of data to seek. To choose among the enumerated values:	
		• BOUNDARY_CONDITION: Read values for (Dirichlet) boundary conditions	
		<ul> <li>BOUNDARY_FORCE: Read values for boundary force (Neumann boundary condition).</li> <li>The values TRACTION and FLUX have the same effect.</li> </ul>	
		BODY_FORCE: Read values for body (or volume) forces.  The value SOURCE has the same effect.	
		<ul> <li>POINT_FORCE: Read values for pointwise forces</li> </ul>	
		INITIAL_FIELD: Read values for initial solution	
		SOLUTION: Read values for a solution vector	
in,out	v	Vect instance that is instantiated on input and filled on output	
in	time	Value of time for which data is read [Default: 0].	
in	dof	DOF to store (Default is 0: All DOFs are chosen).	

# 7.93 Quad4 Class Reference

Defines a 4-node quadrilateral finite element using  $\mathbb{Q}_1$  isoparametric interpolation. Inheritance diagram for Quad4:



## **Public Member Functions**

• Quad4 ()

Default Constructor.

• Quad4 (const Element \*element)

Constructor when data of Element el are given.

• Quad4 (const Side \*side)

Constructor when data of Side sd are given.

• ~Quad4 ()

Destructor.

• void set (const Element \*el)

Choose element by giving its pointer.

• void set (const Side \*sd)

Choose side by giving its pointer.

• void setLocal (const Point < real\_t > &s)

Initialize local point coordinates in element.

void atGauss (int n, std::vector< real\_t > &sh, std::vector< Point< real\_t >> &dsh, std
 ::vector< real\_t > &w)

Calculate shape functions and their partial derivatives and integration weights.

• Point< real\_t > Grad (const LocalVect< real\_t, 4 > &u, const Point< real\_t > &s)

Return gradient of a function defined at element nodes.

• real\_t getMaxEdgeLength () const

Return maximal edge length of quadrilateral.

• real\_t getMinEdgeLength () const

Return minimal edge length of quadrilateral.

## 7.93.1 Detailed Description

Defines a 4-node quadrilateral finite element using  $Q_1$  isoparametric interpolation.

The reference element is the square [-1,1]x[-1,1]. The user must take care to the fact that determinant of jacobian and other quantities depend on the point in the reference element where they are calculated. For this, before any utilization of shape functions or jacobian, function **set** $\leftarrow$ **Local()** must be invoked.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

### 7.93.2 Constructor & Destructor Documentation

Quad4 ( const Element \* element )

Constructor when data of Element el are given.

Parameters

in	element	Pointer to Element

### Quad4 ( const Side \* side )

Constructor when data of Side sd are given.

Parameters

in	side	Pointer to Side

## 7.93.3 Member Function Documentation

void setLocal ( const Point < real\_t > & s )

Initialize local point coordinates in element.

#### **Parameters**

in	S	Point in the reference element This function computes jacobian, shape functions and
		their partial derivatives at s. Other member functions only return these values.

void at Gauss ( int n, std::vector< real\_t > & sh, std::vector< Point< real\_t > & dsh, std::vector< real\_t > & w )

Calculate shape functions and their partial derivatives and integration weights.

#### **Parameters**

in	n	Number of Gauss-Legendre integration points in each direction
in	sh	Vector of shape functions at Gauss points
in	dsh	Vector of shape function derivatives at Gauss points
in	w	Weights of integration formula at Gauss points

## Point<real\_t> Grad ( const LocalVect< real\_t, 4 > & u, const Point< real\_t > & s)

Return gradient of a function defined at element nodes.

### Parameters

in	и	Vector of values at nodes	
in	S	Local coordinates (in [-1,1]*[-1,1]) of point where the gradient is evaluated	

#### Returns

Value of gradient

#### Note

If the derivatives of shape functions were not computed before calling this function (by calling setLocal), this function will compute them

## 7.94 Reconstruction Class Reference

To perform various reconstruction operations.

## **Public Member Functions**

• Reconstruction ()

Default constructor.

• Reconstruction (const Mesh &ms)

Constructor using a refrence to a Mesh instance.

• ~Reconstruction ()

Destructor.

void setMesh (Mesh &ms)

Provide Mesh instance.

• void P0toP1 (const Vect< real\_t > &u, Vect< real\_t > &v)

Smooth an elementwise field to obtain a nodewise field by  $L^2$  projection.

• void DP1toP1 (const Vect< real\_t > &u, Vect< real\_t > &v)

Smooth an Discontinuous P1 field to obtain a nodewise (Continuous  $P_1$ ) field by  $L^2$  projection.

## 7.94.1 Detailed Description

To perform various reconstruction operations.

This class enables various reconstruction operations like smoothing, projections, ...

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

## 7.94.2 Member Function Documentation

void P0toP1 ( const Vect< real\_t > & u, Vect< real\_t > & v)

Smooth an elementwise field to obtain a nodewise field by L<sup>2</sup> projection.

#### **Parameters**

in	и	Vect instance that contains field to smooth
out	v	Vect instance that contains on output smoothed field

#### void DP1toP1 ( const Vect< real\_t > & $u_t$ Vect< real\_t > & v )

Smooth an Discontinuous P1 field to obtain a nodewise (Continuous P<sub>1</sub>) field by L<sup>2</sup> projection.

#### **Parameters**

in	и	Vect instance that contains field to smooth
out	v	Vect instance that contains on output smoothed field

Warning

This function is valid for  $P_1$  triangles (2-D) only.

## 7.95 Rectangle Class Reference

To store and treat a rectangular figure. Inheritance diagram for Rectangle:



### **Public Member Functions**

• Rectangle ()

Default constructor.

- Rectangle (const Point < real\_t > &bbm, const Point < real\_t > &bbM, int code=1)
   Constructor.
- void setBoundingBox (const Point < real\_t > &bbm, const Point < real\_t > &bbM)
   Assign bounding box of the rectangle.
- Point< real\_t > getBoundingBox1 () const

Return first point of bounding box.

Point< real\_t > getBoundingBox2 () const

Return second point of bounding box.

- real\_t getSignedDistance (const Point < real\_t > &p) const
  - Return signed distance of a given point from the current rectangle.
- Rectangle & operator+= (Point< real\_t > a)

*Operator* +=.

• Rectangle & operator+= (real\_t a)

Operator \*=.

## **Additional Inherited Members**

## 7.95.1 Detailed Description

To store and treat a rectangular figure.

### 7.95.2 Constructor & Destructor Documentation

Rectangle (const Point < real\_t > & bbm, const Point < real\_t > & bbM, int code = 1)

Constructor.

#### **Parameters**

in	bbm	Left Bottom point of rectangle
in	bbM	Right Top point of rectangle
in	code	Code to assign to rectangle

## 7.95.3 Member Function Documentation

void setBoundingBox ( const Point< real\_t > & bbm, const Point< real\_t > & bbM)

Assign bounding box of the rectangle.

#### **Parameters**

in	bbm	Left Bottom point
in	bbM	Right Top point

### real\_t getSignedDistance ( const Point < real\_t > & p ) const [virtual]

Return signed distance of a given point from the current rectangle.

The computed distance is negative if p lies in the rectangle, negative if it is outside, and 0 on its boundary

#### **Parameters**

Reimplemented from Figure.

### Rectangle& operator+= ( Point< real\_t > a )

Operator +=.

Translate rectangle by a vector a

## Rectangle & operator += ( real\_t a )

Operator \*=.

Scale rectangle by a factor a

## 7.96 Side Class Reference

To store and treat finite element sides (edges in 2-D or faces in 3-D)

## **Public Types**

```
    enum SideType {
        INTERNAL_SIDE = 0,
        EXTERNAL_BOUNDARY = 1,
        INTERNAL_BOUNDARY = 2 }
```

## **Public Member Functions**

• Side ()

Default Constructor.

• Side (size\_t label, const string &shape)

Constructor initializing side label and shape.

• Side (size\_t label, int shape)

Constructor initializing side label and shape.

• Side (const Side &sd)

Copy constructor.

• ~Side ()

Destructor.

• void Add (Node \*node)

*Insert a node at end of list of nodes of side.* 

• void Add (Edge \*edge)

Insert an edge at end of list of edges of side.

• void setLabel (size\_t i)

Define label of side.

void setFirstDOF (size\_t n)

Define First DOF.

void setNbDOF (size\_t nb\_dof)

Set number of degrees of freedom (DOF).

void DOF (size\_t i, size\_t dof)

Define label of DOF.

void setDOF (size\_t &first\_dof, size\_t nb\_dof)

Define number of DOF.

• void setCode (size\_t dof, int code)

Assign code to a DOF.

void setCode (const string &exp, int code, size\_t dof=1)

Define code by a boolean algebraic expression invoking coordinates of side nodes.

• void Replace (size\_t label, Node \*node)

Replace a node at a given local label.

• void Add (Element \*el)

Set pointer to neighbor element.

• void set (Element \*el, size\_t i)

Set pointer to neighbor element.

void setNode (size\_t i, Node \*node)

Assign a node given by its pointer as the i-th node of side.

void setOnBoundary ()

Say that the side is on the boundary.

• int getShape () const

Return side's shape.

• size\_t getLabel () const

Return label of side.

• size\_t n () const

Return label of side.

• size\_t getNbNodes () const

Return number of side nodes.

• size\_t getNbVertices () const

Return number of side vertices.

• size\_t getNbEq () const

Return number of side equations.

• size\_t getNbDOF () const

Return number of DOF.

• int getCode (size\_t dof=1) const

Return code for a given DOF of node.

• size\_t getDOF (size\_t i) const

Return label of i-th dof.

• size\_t getFirstDOF () const

Return label of first dof of node.

• Node \* getPtrNode (size\_t i) const

Return pointer to node of local label i.

Node \* operator() (size\_t i) const

Operator ().

• size\_t getNodeLabel (size\_t i) const

Return global label of node with given local label.

• Element \* getNeighborElement (size\_t i) const

Return pointer to i-th side neighboring element.

Element \* getOtherNeighborElement (Element \*el) const

Return pointer to other neighboring element than given one.

• Point< real\_t > getNormal () const

Return normal vector to side.

Point< real\_t > getUnitNormal () const

Return unit normal vector to side.

• int isOnBoundary () const

Boundary side or not.

• int isReferenced ()

Say if side has a nonzero code or not.

• real\_t getMeasure () const

Return measure of side.

• size\_t Contains (const Node \*nd) const

Say if a given node belongs to current side.

void setActive (bool opt=true)

Set side is active (default) or not if argument is false

• bool isActive () const

Return true or false whether side is active or not.

• int getLevel () const

Return side level Side level increases when side is refined (starting from 0). If the level is 0, then the element has no father.

• void setChild (Side \*sd)

Assign side as child of current one and assign current side as father.

Side \* getParent () const

Return pointer to parent side Return null if no parent.

• Side \* getChild (size\_t i) const

Return pointer to i-th child side Returns null pointer is no childs.

• size\_t getNbChilds () const

Return number of children of side.

## 7.96.1 Detailed Description

To store and treat finite element sides (edges in 2-D or faces in 3-D)

Defines a side of a finite element mesh. The sides are given in particular by their shapes and a list of nodes. Each node can be accessed by the member function **getPtrNode()**. The string defining the element shape must be chosen according to the following list:

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

#### 7.96.2 Member Enumeration Documentation

## enum SideType

To select side type (boundary side or not).

Enumerator

INTERNAL\_SIDE Internal sideEXTERNAL\_BOUNDARY Side on external boundaryINTERNAL\_BOUNDARY Side on internal boundary

## 7.96.3 Constructor & Destructor Documentation

Side ( size\_t label, const string & shape )

Constructor initializing side label and shape.

#### Parameters

in	label	Label to assign to side.
in	shape	Shape of side (See class description).

### Side ( size\_t label, int shape )

Constructor initializing side label and shape.

#### **Parameters**

in	label	to assign to side.
in	shape	of side (See enum ElementShape in Mesh).

## 7.96.4 Member Function Documentation

void DOF ( size\_t i, size\_t dof )

Define label of DOF.

#### Parameters

in	i	DOF index
in	dof	Its label

## void setDOF ( size\_t & first\_dof, size\_t nb\_dof )

Define number of DOF.

#### **Parameters**

in,out	first_dof	Label of the first DOF in input that is actualized
in	nb_dof	Number of DOF

## void setCode ( size\_t dof, int code )

Assign code to a DOF.

#### Parameters

in	dof	DOF to which code is assigned
in	code	Code to assign

## void setCode ( const string & exp, int code, size\_t dof = 1 )

Define code by a boolean algebraic expression invoking coordinates of side nodes.

### Parameters

in	ехр	Boolean algebraic expression as required by fparser
in	code	Code to assign to node if the algebraic expression is true
in	dof	Degree of Freedom for which code is assigned [Default: 1]

## void Add ( Element \* el )

Set pointer to neighbor element.

## Parameters

in	el	Pointer to element to add as a neigbor element
----	----	--

#### Remarks

This function adds the pointer el only if this one is not a null pointer

#### void set ( Element \* el, size\_t i )

Set pointer to neighbor element.

#### **Parameters**

in	el	Pointer to element to set as a neighbor element
in	i	Local number of neighbor element

### Remarks

This function differs from the Add by the fact that the local label of neighbor element is given

## int getCode ( $size_t dof = 1$ ) const

Return code for a given DOF of node.

#### **Parameters**

in	dof	Local label of degree of freedom. [Default: 1]	]
----	-----	--	---

## Node\* operator() ( size\_t i ) const

Operator ().

Return pointer to node of local label i.

## Element\* getNeighborElement ( size\_t i ) const

Return pointer to i-th side neighboring element.

#### Parameters

in	i	Local label of neighbor element (must be equal to 1 or 2).
----	---	--

## Element\* getOtherNeighborElement ( Element \* el ) const

Return pointer to other neighboring element than given one.

## Parameters

in	el	Pointer to a given neighbor element

#### Remarks

If the side is on the boundary this function returns null pointer

### Point<real\_t> getNormal ( ) const

Return normal vector to side.

The normal vector is oriented from the first neighbor element to the second one.

#### Warning

The norm of this vector is equal to the measure of the side (length of the edge in 2-D and area of the face in 3-D), and To get the unit normal, use rather the member function get← UnitNormal.

#### Point<real\_t> getUnitNormal ( ) const

Return unit normal vector to side.

The unit normal vector is oriented from the first neighbor element to the second one.

#### Remarks

The norm of this vector is equal to one.

#### int isOnBoundary ( ) const

Boundary side or not.

Returns 1 or -1 if side is on boundary Depending on whether the first or the second neighbor element is defined Returns 0 if side is an inner one

#### Remarks

This member function is valid only if member function Mesh::getAllSides() or Mesh::get← BoundarySides() has been called before.

#### real\_t getMeasure ( ) const

Return measure of side.

This member function returns length or area of side. In case of quadrilaterals it returns determinant of Jacobian of mapping between reference and actual side

#### size\_t Contains ( const Node \* nd ) const

Say if a given node belongs to current side.

#### Parameters

in	nd	Pointer to searched node
	1000	1 officer to bear crica floate

#### Returns

index (local label) of node if found, 0 if not

#### void setChild ( Side \* sd )

Assign side as child of current one and assign current side as father.

This function is principally used when refining is invoked (e.g. for mesh adaption)

#### **Parameters**

in	sd	Pointer to side to assign
----	----	---------------------------

## 7.97 SideList Class Reference

Class to construct a list of sides having some common properties.

#### **Public Member Functions**

SideList (Mesh &ms)

Constructor using a Mesh instance.

• ∼SideList ()

Destructor.

• void selectCode (int code, int dof=1)

Select sides having a given code for a given degree of freedom.

• void unselectCode (int code, int dof=1)

Unselect sides having a given code for a given degree of freedom.

• size\_t getNbSides () const

Return number of selected sides.

• void top ()

Reset list of sides at its top position (Non constant version)

• void top () const

Reset list of sides at its top position (Constant version)

• Side \* get ()

Return pointer to current side and move to next one (Non constant version)

• Side \* get () const

Return pointer to current side and move to next one (Constant version)

## 7.97.1 Detailed Description

Class to construct a list of sides having some common properties.

This class enables choosing multiple selection criteria by using function select... However, the intersection of these properties must be empty.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

## 7.97.2 Member Function Documentation

void selectCode ( int code, int dof = 1 )

Select sides having a given code for a given degree of freedom.

#### **Parameters**

in	code	Code that sides share
in	dof	Degree of Freedom label [Default: 1]

## void unselectCode ( int code, int dof = 1 )

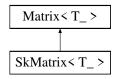
Unselect sides having a given code for a given degree of freedom.

#### **Parameters**

in	code	Code of sides to exclude
in	dof	Degree of Freedom label [Default: 1]

# 7.98 SkMatrix $< T_{-} >$ Class Template Reference

To handle square matrices in skyline storage format. Inheritance diagram for SkMatrix<  $T_->$ :



## **Public Member Functions**

• SkMatrix ()

Default constructor.

• SkMatrix (size\_t size, int is\_diagonal=false)

Constructor that initializes a dense symmetric matrix.

• SkMatrix (Mesh &mesh, size\_t dof=0, int is\_diagonal=false)

Constructor using mesh to initialize skyline structure of matrix.

• SkMatrix (const Vect< size\_t > &ColHt)

Constructor that initializes skyline structure of matrix using vector of column heights.

• SkMatrix (const SkMatrix < T\_ > &m)

 $Copy\ Constructor.$ 

• ~SkMatrix ()

Destructor.

• void setMesh (Mesh &mesh, size\_t dof=0)

Determine mesh graph and initialize matrix.

• void setSkyline (Mesh &mesh)

Determine matrix structure.

• void setDiag ()

Store diagonal entries in a separate internal vector.

• void setDOF (size\_t i)

```
Choose DOF to activate.
• void set (size_t i, size_t j, const T_ &val)
      Assign a value to an entry ofthe matrix.
• void Axpy (T<sub>-</sub> a, const SkMatrix < T<sub>-</sub> > &m)
      Add to matrix the product of a matrix by a scalar.
• void Axpy (T_a, const Matrix < T_s > *m)
      Add to matrix the product of a matrix by a scalar.
• void MultAdd (const Vect< T_> &x, Vect< T_> &y) const
      Multiply matrix by vector x and add to y.
• void TMultAdd (const Vect< T_-> &x, Vect< T_-> &y) const
      Multiply transpose of matrix by vector x and add to y.

    void MultAdd (T<sub>-</sub> a, const Vect< T<sub>-</sub> > &x, Vect< T<sub>-</sub> > &y) const

      Multiply matrix by a vector and add to another one.
• void Mult (const Vect< T_-> &x, Vect< T_-> &y) const
      Multiply matrix by vector x and save in y.
• void TMult (const Vect< T_-> &x, Vect< T_-> &y) const
      Multiply transpose of matrix by vector x and save in y.
• void add (size_t i, size_t j, const T_ &val)
      Add a constant value to an entry of the matrix.

    size_t getColHeight (size_t i) const

      Return column height.
• T_ operator() (size_t i, size_t j) const
      Operator () (Constant version).
• T<sub>-</sub> & operator() (size<sub>-</sub>t i, size<sub>-</sub>t j)
      Operator () (Non constant version).

    void DiagPrescribe (Mesh &mesh, Vect< T<sub>-</sub> > &b, const Vect< T<sub>-</sub> > &u, int flag=0)

      Impose an essential boundary condition.
• void DiagPrescribe (Vect< T_> &b, const Vect< T_> &u, int flag=0)
      Impose an essential boundary condition using the Mesh instance provided by the constructor.
• SkMatrix< T<sub>-</sub> > & operator= (const SkMatrix< T<sub>-</sub> > &m)
      Operator =.
• SkMatrix< T<sub>-</sub> > & operator= (const T<sub>-</sub> &x)
      Operator =.
• SkMatrix< T<sub>-</sub> > & operator+= (const SkMatrix< T<sub>-</sub> > &m)
      Operator +=.
• SkMatrix< T_- > & operator += (const T_- & x)
      Operator +=.
• SkMatrix< T_> & operator*= (const T_ &x)
      Operator *=.
• int setLU ()
      Factorize the matrix (LU factorization)
```

• int solve (Vect< T\_> &b, bool fact=true)

Solve linear system.

• int solve (const Vect<  $T_->$  &b, Vect<  $T_->$  &x, bool fact=true)

Solve linear system.

• T<sub>-</sub> \* **get** () const

Return C-Array.

• T\_get (size\_t i, size\_t j) const

Return entry (i, j) of matrix if this one is stored, 0 else.

## 7.98.1 Detailed Description

template < class T\_> class OFELI::SkMatrix < T\_>

To handle square matrices in skyline storage format.

This template class allows storing and manipulating a matrix in skyline storage format. The matrix entries are stored in 2 vectors column by column as in the following example:

#### **Template Parameters**

```
T \leftarrow  Data type (double, float, complex<double>, ...)
```

#### Author

Rachid Touzani

## Copyright

GNU Lesser Public License

#### 7.98.2 Constructor & Destructor Documentation

#### SkMatrix ( )

Default constructor.

Initializes a zero-dimension matrix

#### SkMatrix ( size\_t size, int is\_diagonal = false )

Constructor that initializes a dense symmetric matrix. Normally, for a dense matrix this is not the right class.

### Parameters

in	size	Number of matrix rows (and columns).
in	is_diagonal	Boolean to select if the matrix is diagonal or not [Default: false]

## SkMatrix ( Mesh & mesh, size\_t dof = 0, int is\_diagonal = false )

Constructor using mesh to initialize skyline structure of matrix.

OFELI's Reference Guide

#### **Parameters**

in	mesh	Mesh instance for which matrix graph is determined.
in	dof	Option parameter, with default value 0. dof=1 means that only one degree of freedom for each node (or element or side) is taken to determine matrix structure. The value dof=0 means that matrix structure is determined using all DOFs.
in	is_diagonal	Boolean argument to say is the matrix is actually a diagonal matrix or not.

# SkMatrix ( const Vect< size $_{-}$ t > & ColHt )

Constructor that initializes skyline structure of matrix using vector of column heights.

# Parameters

in	ColHt	Vect instance that contains rows lengths of matrix.
----	-------	---

# 7.98.3 Member Function Documentation

void setMesh ( Mesh & mesh, size\_t dof = 0 )

Determine mesh graph and initialize matrix.

This member function is called by constructor with the same arguments

# Parameters

in	mesh	Mesh instance for which matrix graph is determined.
in	dof	Option parameter, with default value 0. dof=1 means that only one degree of freedom for each node (or element or side) is taken to determine matrix structure. The value dof=0 means that matrix structure is determined using all DOFs.

# void setSkyline ( Mesh & mesh )

Determine matrix structure.

This member function calculates matrix structure using a Mesh instance.

# Parameters

in	mesh	Mesh instance

#### void setDOF ( size\_t i )

Choose DOF to activate.

This function is available only if variable dof is equal to 1 in the constructor

# 7.98. SKMATRIX< T\_> CLASS TEMPLATE REFERENCEPTER 7. CLASS DOCUMENTATION

#### Parameters

in	i	Index of the DOF
----	---	------------------

# void set ( size\_t i, size\_t j, const T\_ & val ) [virtual]

Assign a value to an entry of the matrix.

# Parameters

in	i	Row index (starting at i=1)
in	j	Column index (starting at i=1)
in	val	Value to assign to entry a(i,j)

Implements Matrix  $< T_- >$ .

# void Axpy ( $T_- a$ , const SkMatrix $< T_- > \& m$ )

Add to matrix the product of a matrix by a scalar.

# Parameters

in	а	Scalar to premultiply
in	m	Matrix by which a is multiplied. The result is added to current instance

# void Axpy ( $T_-a$ , const Matrix $< T_- > * m$ ) [virtual]

Add to matrix the product of a matrix by a scalar.

#### Parameters

in	а	Scalar to premultiply
in	m	Matrix by which a is multiplied. The result is added to current instance

Implements Matrix  $< T_- >$ .

# void MultAdd ( const Vect< $T_- > \& x$ , Vect< $T_- > \& y$ ) const [virtual]

Multiply matrix by vector x and add to y.

#### **Parameters**

in	x	Vector to multiply by matrix
in,out	y	Vector to add to the result. y contains on output the result.

Implements Matrix  $< T_- >$ .

# void TMultAdd ( const Vect< T $_->$ & x, Vect< T $_->$ & y ) const

Multiply transpose of matrix by vector x and add to y.

#### Parameters

in	x	Vector to multiply by matrix
in,out	y	Vector to add to the result. y contains on output the result.

# void MultAdd ( $T_-a$ , const Vect< $T_- > \& x$ , Vect< $T_- > \& y$ ) const [virtual]

Multiply matrix by a vector and add to another one.

#### Parameters

in	а	Constant to multiply by matrix
in	x	Vector to multiply by matrix
in,out	у	Vector to add to the result. y contains on output the result.

Implements Matrix $< T_->$ .

# void Mult ( const Vect< $T_- > \& x$ , Vect< $T_- > \& y$ ) const [virtual]

Multiply matrix by vector x and save in y.

# Parameters

i	n	х	Vector to multiply by matrix
c	out	y	Vector that contains on output the result.

Implements Matrix  $< T_- >$ .

# void TMult ( const Vect< $T_- > & x$ , Vect< $T_- > & y$ ) const [virtual]

Multiply transpose of matrix by vector x and save in y.

# Parameters

in	x	Vector to multiply by matrix
out	y	Vector that contains on output the result.

Implements Matrix  $< T_- >$ .

# void add ( size\_t i, size\_t j, const T\_ & val ) [virtual]

Add a constant value to an entry of the matrix.

in	i	Row index

# 7.98. SKMATRIX < T\_ > CLASS TEMPLATE REFERENCEPTER 7. CLASS DOCUMENTATION

#### **Parameters**

in	j	Column index
in	val	Constant value to add to a(i,j)

Implements Matrix  $< T_- >$ .

# size\_t getColHeight ( size\_t i ) const

Return column height.

Column height at entry i is returned.

# T\_operator() ( size\_t i, size\_t j ) const [virtual]

Operator () (Constant version).

#### **Parameters**

in	i	Row index
in	j	Column index

Implements Matrix  $< T_- >$ .

# T\_& operator() ( size\_t i, size\_t j ) [virtual]

Operator () (Non constant version).

# Parameters

in	i	Row index
in	j	Column index

Implements Matrix  $< T_- >$ .

# void DiagPrescribe (Mesh & mesh, Vect $< T_- > \& b$ , const Vect $< T_- > \& u$ , int flag = 0)

Impose an essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. It can be modified by member function **setPenal**(..).

in	mesh	Mesh instance from which information is extracted.
in	b	Vect instance that contains right-hand side.
in	и	Vect instance that conatins imposed valued at DOFs where they are to be imposed.
in	flag	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

# void DiagPrescribe ( Vect< T $_->$ & b, const Vect< T $_->$ & u, int flag=0)

Impose an essential boundary condition using the Mesh instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. It can be modified by member function **setPenal**(..).

#### **Parameters**

in	b	Vect instance that contains right-hand side.
in	и	Vect instance that conatins imposed valued at DOFs where they are to be imposed.
in	flag	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

# SkMatrix<T $_->$ & operator= ( const SkMatrix< T $_->$ & m )

Operator =.

Copy matrix m to current matrix instance.

# SkMatrix<T $_->$ & operator= ( const T $_-$ & x )

Operator =.

define the matrix as a diagonal one with all diagonal entries equal to x.

# SkMatrix<T $_->$ & operator+= ( const SkMatrix< T $_->$ & m )

Operator +=.

Add matrix m to current matrix instance.

#### SkMatrix<T $_->$ & operator+= ( const T $_-$ & x )

Operator +=.

Add constant value x to matrix entries.

# SkMatrix<T $_->$ & operator\*= ( const T $_-$ & x )

Operator \*=.

Premultiply matrix entries by constant value x.

#### int setLU ( )

Factorize the matrix (LU factorization)

LU factorization of the matrix is realized. Note that since this is an in place factorization, the contents of the matrix are modified.

#### Returns

- 0 if factorization was normally performed,
- n if the n-th pivot is null.

#### Remarks

A flag in this class indicates after factorization that this one has been realized, so that, if the member function solve is called after this no further factorization is done.

# int solve ( Vect< $T_-$ > & b, bool fact = true ) [virtual]

Solve linear system.

The linear system having the current instance as a matrix is solved by using the LU decomposition. Solution is thus realized after a factorization step and a forward/backward substitution step. The factorization step is realized only if this was not already done.

Note that this function modifies the matrix contents is a factorization is performed. Naturally, if the the matrix has been modified after using this function, the user has to refactorize it using the function setLU. This is because the class has no non-expensive way to detect if the matrix has been modified. The function setLU realizes the factorization step only.

#### **Parameters**

in,out	b	Vect instance that contains right-hand side on input and solution on output.
in	fact	Set true if matrix is to be factorized (Default value), false if not

#### Returns

- 0 if solution was normally performed,
- n if the n-th pivot is null.

Implements Matrix  $< T_- >$ .

#### int solve ( const Vect< T $_->$ & b, Vect< T $_->$ & x, bool fact = true ) [virtual]

Solve linear system.

The linear system having the current instance as a matrix is solved by using the LU decomposition. Solution is thus realized after a factorization step and a forward/backward substitution step. The factorization step is realized only if this was not already done.

Note that this function modifies the matrix contents is a factorization is performed. Naturally, if the the matrix has been modified after using this function, the user has to refactorize it using the function setLU. This is because the class has no non-expensive way to detect if the matrix has been modified. The function setLU realizes the factorization step only.

#### Parameters

in	b	Vect instance that contains right-hand side.
out	x	Vect instance that contains solution
in	fact	Set true if matrix is to be factorized (Default value), false if not

#### Returns

- 0 if solution was normally performed,
- n if the n-th pivot is null.

Implements Matrix  $< T_- >$ .

# T\_\* get ( ) const

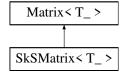
Return C-Array.

Skyline of matrix is stored row by row.

# 7.99 SkSMatrix< T $_->$ Class Template Reference

To handle symmetric matrices in skyline storage format.

Inheritance diagram for SkSMatrix< T\_>:



# **Public Member Functions**

• SkSMatrix ()

Default constructor.

• SkSMatrix (size\_t size, int is\_diagonal=false)

Constructor that initializes a dense symmetric matrix.

• SkSMatrix (Mesh &mesh, size\_t dof=0, int is\_diagonal=false)

Constructor using mesh to initialize skyline structure of matrix.

• SkSMatrix (const Vect< size\_t > &ColHt)

Constructor that initializes skyline structure of matrix using vector of column height.

• SkSMatrix (const Vect< size\_t > &I, const Vect< size\_t > &J, int opt=1)

Constructor for a square matrix using non zero row and column indices.

• SkSMatrix (const Vect< size\_t > &I, const Vect< size\_t > &J, const Vect< T\_ > &a, int opt=1) Constructor for a square matrix using non zero row and column indices.

• SkSMatrix (const SkSMatrix < T\_ > &m)

Copy Constructor.

• ~SkSMatrix ()

Destructor.

• void setMesh (Mesh &mesh, size\_t dof=0)

Determine mesh graph and initialize matrix.

• void setSkyline (Mesh &mesh)

Determine matrix structure.

void setDiag ()

Store diagonal entries in a separate internal vector.

• void set (size\_t i, size\_t j, const T\_ &val)

Assign a value to an entry of the matrix.

• void Axpy  $(T_- a, const SkSMatrix < T_- > &m)$ 

*Add to matrix the product of a matrix by a scalar.* 

• void Axpy (T<sub>-</sub> a, const Matrix< T<sub>-</sub> > \*m)

Add to matrix the product of a matrix by a scalar.

void MultAdd (const Vect< T<sub>-</sub> > &x, Vect< T<sub>-</sub> > &y) const

Multiply matrix by vector x and add to y.

```
• void MultAdd (T_-a, const Vect< T_- > &x, Vect< T_- > &y) const
      Multiply matrix by vector \mathbf{a} * \mathbf{x} and add to \mathbf{y}.
• void Mult (const Vect< T_-> &x, Vect< T_-> &y) const
      Multiply matrix by vector x and save in y
• void TMult (const Vect< T_-> &x, Vect< T_-> &y) const
      Multiply transpose of matrix by vector x and save in y.
• void add (size_t i, size_t j, const T_ &val)
      Add a constant to an entry of the matrix.
• size_t getColHeight (size_t i) const
      Return column height.
• Vect< T_> getColumn (size_t j) const
      Get j-th column vector.
• Vect< T_> getRow (size_t i) const
      Get i-th row vector.
• T_ & operator() (size_t i, size_t j)
      Operator () (Non constant version).
• T_ operator() (size_t i, size_t j) const
      Operator () (Constant version).
• SkSMatrix< T_> & operator= (const SkSMatrix< T_> &m)
      Operator =.
• SkSMatrix< T_> & operator= (const T_ &x)
      Operator = .
• SkSMatrix< T<sub>-</sub> > & operator+= (const SkSMatrix< T<sub>-</sub> > &m)
      Operator +=.
• SkSMatrix< T_- > & operator*= (const T_- &x)
      Operator *=.
• int setLDLt ()
      Factorize matrix (LDLt (Crout) factorization).
• int solveLDLt (const Vect< T_-> &b, Vect< T_-> &x)
      Solve a linear system using the LDLt (Crout) factorization.
• int solve (Vect< T_> &b, bool fact=true)
      Solve linear system.
• int solve (const Vect< T_-> &b, Vect< T_-> &x, bool fact=true)
      Solve linear system.
• T_* get () const
      Return C-Array.
• void set (size_t i, T_x)
      Assign a value to the i-th entry of C-array containing matrix.
```

• T<sub>\_</sub> get (size\_t i, size\_t j) const

Return entry (i, j) of matrix if this one is stored, 0 else.

# 7.99.1 Detailed Description

template < class T\_> class OFELI::SkSMatrix < T\_>

To handle symmetric matrices in skyline storage format.

This template class allows storing and manipulating a symmetric matrix in skyline storage format.

The matrix entries are stored column by column as in the following example:

# **Template Parameters**

```
T_{\leftarrow} Data type (double, float, complex<double>, ...)
```

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# 7.99.2 Constructor & Destructor Documentation

# SkSMatrix ( )

Default constructor.

Initializes a zero-dimension matrix

# SkSMatrix ( size\_t size, int is\_diagonal = false )

Constructor that initializes a dense symmetric matrix.

Normally, for a dense matrix this is not the right class.

### Parameters

in	size	Number of matrix rows (and columns).
in	is_diagonal	Boolean to select if the matrix is diagonal or not [Default: false]

# SkSMatrix ( Mesh & mesh, size\_t dof = 0, int is\_diagonal = false )

Constructor using mesh to initialize skyline structure of matrix.

#### Parameters

in	mesh	Mesh instance for which matrix graph is determined.
in	dof	Option parameter, with default value 0.
		dof=1 means that only one degree of freedom for each node (or element or side) is taken to determine matrix structure. The value dof=0 means that matrix structure is determined using all DOFs.
in	is_diagonal	Boolean argument to say is the matrix is actually a diagonal matrix or not.

# SkSMatrix ( const Vect< size\_t > & ColHt )

Constructor that initializes skyline structure of matrix using vector of column height.

# Parameters

in	ColHt	Vect instance that contains rows lengths of matrix.
----	-------	---

# SkSMatrix ( const Vect< size\_t > & I, const Vect< size\_t > & J, int opt = 1)

Constructor for a square matrix using non zero row and column indices.

#### Parameters

in	I	Vector containing row indices	
in	J	Vector containing column indices	
in	opt	Flag indicating if vectors I and J are cleaned and ordered (opt=1) or not (opt=0). In the latter case, these vectors can contain the same contents more than once and are not necessarily ordered.	

# SkSMatrix ( const Vect< size\_t > & I, const Vect< size\_t > & J, const Vect< $T_-$ > & a, int opt = 1)

Constructor for a square matrix using non zero row and column indices.

in	I	Vector containing row indices		
in	J	Vector containing column indices		
in	а	Vector containing matrix entries in the same order than the one given by I and J		
in	opt	Flag indicating if vectors I and J are cleaned and ordered (opt=1) or not (opt=0). In the latter case, these vectors can contain the same contents more than once and are not necessarily ordered		

# 7.99.3 Member Function Documentation

void setMesh ( Mesh & mesh, size\_t dof = 0 )

Determine mesh graph and initialize matrix.

This member function is called by constructor with the same arguments

#### **Parameters**

in	mesh	Mesh instance for which matrix graph is determined.	
in	dof	Option parameter, with default value 0.	
		dof=1 means that only one degree of freedom for each node (or element or side)	
		is taken to determine matrix structure. The value dof=0 means that matrix	
		structure is determined using all DOFs.	

# void setSkyline ( Mesh & mesh )

Determine matrix structure.

This member function calculates matrix structure using Mesh instance mesh.

# void set ( size\_t i, size\_t j, const T\_ & val ) [virtual]

Assign a value to an entry of the matrix.

#### **Parameters**

in	i	Row index	
in	j	Column index	
in	val	Value to assign to a(i,j)	

Implements Matrix  $< T_- >$ .

# void Axpy ( $T_-a$ , const SkSMatrix< $T_- > \& m$ )

Add to matrix the product of a matrix by a scalar.

# Parameters

i	n	а	Scalar to premultiply	
i	n	m	Matrix by which a is multiplied. The result is added to current instance	

# void Axpy ( $T_-a$ , const Matrix< $T_- > * m$ ) [virtual]

Add to matrix the product of a matrix by a scalar.

in a Scalar to premultiply		Scalar to premultiply
in	m	Pointer to Matrix by which a is multiplied. The result is added to current instance

Implements Matrix  $< T_- >$ .

void MultAdd ( const Vect<  $T_- > \& x$ , Vect<  $T_- > \& y$  ) const [virtual]

Multiply matrix by vector x and add to y.

#### **Parameters**

in	x	Vector to multiply by matrix	
in,out	y	Vector to add to the result. y contains on output the result	

Implements Matrix  $< T_- >$ .

void MultAdd (  $T_-a$ , const Vect<  $T_- > \& x$ , Vect<  $T_- > \& y$  ) const [virtual]

Multiply matrix by vector a\*x and add to y.

#### **Parameters**

in	а	Constant to multiply by matrix	
in	x	Vector to multiply by matrix	
in,out	y	Vector to add to the result. y contains on output the resul	

Implements Matrix  $< T_- >$ .

void Mult ( const Vect<  $T_-> & x$ , Vect<  $T_-> & y$  ) const [virtual]

Multiply matrix by vector x and save in y

#### Parameters

in	x	Vector to multiply by matrix	
out	y	Vector that contains on output the result.	

Implements Matrix $< T_- >$ .

void TMult ( const Vect<  $T_- > & x$ , Vect<  $T_- > & y$  ) const [virtual]

Multiply transpose of matrix by vector x and save in y.

#### Parameters

in	x	Vector to multiply by matrix	
out	y	Vector that contains on output the result.	

Implements Matrix  $< T_- >$ .

void add ( size\_t i, size\_t j, const T\_ & val ) [virtual]

Add a constant to an entry of the matrix.

#### **Parameters**

in	i	Row index	
in	j	Column index	
in	val	Constant value to add to a(i,j)	

Implements Matrix  $< T_- >$ .

# size\_t getColHeight ( size\_t i ) const

Return column height.

Column height at entry i is returned.

# $T_{\infty}$ operator() ( size\_t i, size\_t j ) [virtual]

Operator () (Non constant version).

#### **Parameters**

in	i	Row index
in	j	Column index

# Warning

To modify a value of an entry of the matrix it is safer not to modify both lower and upper triangles. Otherwise, wrong values will be assigned. If not sure, use the member functions set or add.

Implements Matrix $< T_- >$ .

# T\_operator() ( size\_t i, size\_t j ) const [virtual]

Operator () (Constant version).

#### **Parameters**

in	i	Row index
in	j	Column index

Implements Matrix  $< T_- >$ .

# SkSMatrix<T $_->$ & operator= ( const SkSMatrix< T $_->$ & m )

Operator =.

Copy matrix m to current matrix instance.

# SkSMatrix<T $_->$ & operator= ( const T $_-$ & x )

Operator =.

define the matrix as a diagonal one with all diagonal entries equal to x.

### SkSMatrix<T $_->$ & operator+= ( const SkSMatrix< T $_->$ & m )

Operator +=.

Add matrix m to current matrix instance.

# SkSMatrix<T $_->$ & operator\*= ( const T $_-$ & x )

Operator \*=.

Premultiply matrix entries by constant value x.

#### int setLDLt ( )

Factorize matrix (LDLt (Crout) factorization).

#### Returns

- 0 if factorization was normally performed
- n if the n-th pivot is null

### int solveLDLt ( const Vect< T $_->$ & b, Vect< T $_->$ & x )

Solve a linear system using the LDLt (Crout) factorization.

This function solves a linear system. The LDLt factorization is performed if this was not already done using the function setLU.

#### **Parameters**

ĺ	in	b	Vect instance that contains right-hand side
	out	x	Vect instance that contains solution

#### Returns

- 0 if solution was normally performed,
- n if the n-th pivot is null

Solution is performed only is factorization has previouly been invoked.

# int solve ( Vect< T $_->$ & b, bool fact = true ) [virtual]

Solve linear system.

The linear system having the current instance as a matrix is solved by using the LDLt decomposition. Solution is thus realized after a factorization step and a forward/backward substitution step. The factorization step is realized only if this was not already done.

Note that this function modifies the matrix contents is a factorization is performed. Naturally, if the the matrix has been modified after using this function, the user has to refactorize it using the function setLU. This is because the class has no non-expensive way to detect if the matrix has been modified. The function setLDLt realizes the factorization step only.

in,out	b	Vect instance that contains right-hand side on input and solution on output.
in	fact	Set true if matrix is to be factorized (Default value), false if not

#### Returns

- 0 if solution was normally performed,
- n if the n-th pivot is null.

Implements Matrix $< T_- >$ .

int solve ( const Vect< T $_->$  & b, Vect< T $_->$  & x, bool fact = true ) [virtual]

Solve linear system.

The linear system having the current instance as a matrix is solved by using the LDLt decomposition. Solution is thus realized after a factorization step and a forward/backward substitution step. The factorization step is realized only if this was not already done.

Note that this function modifies the matrix contents is a factorization is performed. Naturally, if the the matrix has been modified after using this function, the user has to refactorize it using the function setLDLt. This is because the class has no non-expensive way to detect if the matrix has been modified. The function setLDLt realizes the factorization step only.

#### **Parameters**

in	b	Vect instance that contains right-hand side.
out	x	Vect instance that contains solution
in	fact	Set true if matrix is to be factorized (Default value), false if not

#### Returns

- 0 if solution was normally performed,
- n if the n-th pivot is null.

Implements Matrix  $< T_- >$ .

# T\_\* get ( ) const

Return C-Array.

Skyline of matrix is stored row by row.

# 7.100 Sphere Class Reference

To store and treat a sphere.

Inheritance diagram for Sphere:



# **Public Member Functions**

• Sphere ()

Default construcor.

• Sphere (const Point< real\_t > &c, real\_t r, int code=1)

Constructor.

• void setRadius (real\_t r)

Assign radius of sphere.

real\_t getRadius () const

Return radius of sphere.

• void setCenter (const Point< real\_t > &c)

Assign coordinates of center of sphere.

Point< real\_t > getCenter () const

Return coordinates of center of sphere.

• real\_t getSignedDistance (const Point< real\_t > &p) const

Return signed distance of a given point from the current sphere.

• Sphere & operator+= (Point< real\_t > a)

Operator +=.

• Sphere & operator+= (real\_t a)

Operator \*=.

# **Additional Inherited Members**

# 7.100.1 Detailed Description

To store and treat a sphere.

# 7.100.2 Constructor & Destructor Documentation

Sphere ( const Point< real\_t > & c, real\_t r, int code = 1 )

Constructor.

#### **Parameters**

in	С	Coordinates of center of sphere
in	r	Radius
in	code	Code to assign to the generated sphere [Default: 1]

# 7.100.3 Member Function Documentation

real\_t getSignedDistance ( const Point< real\_t > & p ) const [virtual]

Return signed distance of a given point from the current sphere.

The computed distance is negative if p lies in the ball, positive if it is outside, and 0 on the sphere

#### **Parameters**

in	р	Point <double> instance</double>
----	---	----------------------------------

Reimplemented from Figure.

### Sphere& operator+= ( Point< real\_t > a )

Operator +=.

Translate sphere by a vector a

# Sphere& operator+= $( real_t a )$

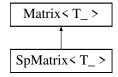
Operator \*=.

Scale sphere by a factor a

# 7.101 SpMatrix< T $_->$ Class Template Reference

To handle matrices in sparse storage format.

Inheritance diagram for SpMatrix< T $_->$ :



### **Public Member Functions**

• SpMatrix ()

Default constructor.

• SpMatrix (size\_t nr, size\_t nc)

Constructor that initializes current instance as a dense matrix.

• SpMatrix (size\_t size, int is\_diagonal=false)

Constructor that initializes current instance as a dense matrix.

SpMatrix (Mesh &mesh, size\_t dof=0, int is\_diagonal=false)

Constructor using a Mesh instance.

• SpMatrix (const Vect< RC > &I, int opt=1)

Constructor for a square matrix using non zero row and column indices.

• SpMatrix (const Vect< RC > &I, const Vect< T<sub>-</sub> > &a, int opt=1)

Constructor for a square matrix using non zero row and column indices.

SpMatrix (size\_t nr, size\_t nc, const vector < size\_t > &row\_ptr, const vector < size\_t > &col←
 \_ind)

Constructor for a rectangle matrix.

SpMatrix (size\_t nr, size\_t nc, const vector < size\_t > &row\_ptr, const vector < size\_t > &col

 ind, const vector < T<sub>-</sub> > &a)

Constructor for a rectangle matrix.

SpMatrix (const vector < size\_t > &row\_ptr, const vector < size\_t > &col\_ind)

Constructor for a rectangle matrix.

SpMatrix (const vector < size\_t > &row\_ptr, const vector < size\_t > &col\_ind, const vector < T\_ > &a)

Constructor for a rectangle matrix.

• SpMatrix (const SpMatrix &m)

Copy constructor.

• ~SpMatrix ()

Destructor.

• void Identity ()

Define matrix as identity.

• void Dense ()

Define matrix as a dense one.

• void Diagonal ()

Define matrix as a diagonal one.

• void Diagonal (const T<sub>-</sub> &a)

Define matrix as a diagonal one with diagonal entries equal to a

• void Laplace1D (size\_t n, real\_t h)

*Sets the matrix as the one for the Laplace equation in 1-D.* 

void Laplace2D (size\_t nx, size\_t ny)

*Sets the matrix as the one for the Laplace equation in 2-D.* 

void setMesh (Mesh &mesh, size\_t dof=0)

Determine mesh graph and initialize matrix.

• void setOneDOF ()

Activate 1-DOF per node option.

• void setSides ()

Activate Sides option.

• void setDiag ()

Store diagonal entries in a separate internal vector.

• void DiagPrescribe (Mesh &mesh, Vect< T\_> &b, const Vect< T\_> &u)

Impose by a diagonal method an essential boundary condition.

void DiagPrescribe (Vect< T<sub>-</sub> > &b, const Vect< T<sub>-</sub> > &u)

Impose by a diagonal method an essential boundary condition using the Mesh instance provided by the constructor.

void setSize (size\_t size)

*Set size of matrix (case where it's a square matrix).* 

• void setSize (size\_t nr, size\_t nc)

Set size (number of rows) of matrix.

• void setGraph (const Vect< RC > &I, int opt=1)

Set graph of matrix by giving a vector of its nonzero entries.

• Vect< T\_> getRow (size\_t i) const

Get i-th row vector.

• Vect< T\_> getColumn (size\_t j) const

Get j-th column vector.

• T<sub>-</sub> & operator() (size\_t i, size\_t j)

Operator () (Non constant version)

• T\_ operator() (size\_t i, size\_t j) const

Operator () (Constant version)

• T\_ operator() (size\_t i) const

*Operator () with one argument (Constant version)* 

• T\_operator[] (size\_t i) const

Operator [] (Constant version).

• Vect<  $T_->$  operator\* (const Vect<  $T_->$  &x) const

Operator \* to multiply matrix by a vector.

• SpMatrix< T<sub>-</sub> > & operator\*= (const T<sub>-</sub> &a)

*Operator* \*= to premultiply matrix by a constant.

void getMesh (Mesh &mesh)

Get mesh instance whose reference will be stored in current instance of SpMatrix.

• void Mult (const Vect<  $T_-> &x$ , Vect<  $T_-> &y$ ) const

Multiply matrix by vector and save in another one.

• void MultAdd (const Vect< T\_> &x, Vect< T\_> &y) const

Multiply matrix by vector x and add to y.

• void MultAdd ( $T_-$  a, const Vect<  $T_-$  > &x, Vect<  $T_-$  > &y) const

Multiply matrix by vector  $\mathbf{a} * \mathbf{x}$  and add to  $\mathbf{y}$ .

• void TMult (const Vect<  $T_-> &x$ , Vect<  $T_-> &y$ ) const

Multiply transpose of matrix by vector x and save in y.

• void Axpy (T<sub>-</sub> a, const SpMatrix < T<sub>-</sub> > &m)

Add to matrix the product of a matrix by a scalar.

• void Axpy (T<sub>-</sub> a, const Matrix < T<sub>-</sub> > \*m)

Add to matrix the product of a matrix by a scalar.

• void set (size\_t i, size\_t j, const T\_ &val)

Assign a value to an entry of the matrix.

• void add (size\_t i, size\_t j, const T\_ &val)

Add a value to an entry of the matrix.

• void operator= (const T<sub>-</sub> &x)

Operator =.

• size\_t getColInd (size\_t i) const

Return storage information.

• size\_t getRowPtr (size\_t i) const

Return Row pointer at position i.

• int solve (const Vect< T\_> &b, Vect< T\_> &x, bool fact=false)

Solve the linear system of equations.

• void setSolver (Iteration solver=CG\_SOLVER, Preconditioner prec=DIAG\_PREC, int max ← it=1000, real\_t toler=1.e-8)

Choose solver and preconditioner for an iterative procedure.

• void clear ()

brief Set all matrix entries to zero

• T\_\* get () const

Return C-Array.

• T\_get (size\_t i, size\_t j) const

Return entry (i, j) of matrix if this one is stored, 0 otherwise.

# **Friends**

template<class TT\_>
 ostream & operator<< (ostream &s, const SpMatrix< TT\_> &A)

# 7.101.1 Detailed Description

```
template < class T_> class OFELI::SpMatrix < T_>
```

To handle matrices in sparse storage format.

This template class enables storing and manipulating a sparse matrix, i.e. only nonzero terms are stored. Internally, the matrix is stored as a vector instance and uses for the definition of its graph a Vect<size\_t> instance row\_ptr and a Vect<size\_t> instance col\_ind that contains respectively addresses of first element of each row and column indices.

To illustrate this, consider the matrix

```
1 2 0
3 4 0
0 5 0
```

Such a matrix is stored in the vector<real\_t> instance  $\{1,2,3,4,5\}$ . The vectors row\_ptr and col\_ind are respectively:  $\{0,2,4,5\}$ ,  $\{1,2,1,2,2\}$ 

When the library eigen is used in conjunction with OFELI, the class uses the sparse matrix class of eigen and enables then access to specific solvers (see class LinearSolver)

**Template Parameters** 

```
T \leftarrow  Data type (double, float, complex<double>, ...)
```

Author

Rachid Touzani

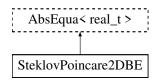
Copyright

GNU Lesser Public License

# 7.102 SteklovPoincare2DBE Class Reference

Solver of the Steklov Poincare problem in 2-D geometries using piecewie constant boundary elemen.

Inheritance diagram for SteklovPoincare2DBE:



# **Public Member Functions**

• SteklovPoincare2DBE ()

Default Constructor.

• SteklovPoincare2DBE (Mesh &ms)

Constructor using mesh data.

• SteklovPoincare2DBE (Mesh &ms, Vect< real\_t > &u)

Constructor that solves the Steklov Poincare problem.

• ~SteklovPoincare2DBE ()

Destructor.

void setMesh (Mesh &ms)

set Mesh instance

• void setExterior ()

Choose domain of the Laplace equation as exterior one.

• int run ()

Solve Setklov-Poincare problem.

# 7.102.1 Detailed Description

Solver of the Steklov Poincare problem in 2-D geometries using piecewie constant boundary elemen.

SteklovPoincare2DBE solves the Steklov Poincare problem in 2-D: Given the trace of a harmonic function on the boundary of a given (inner or outer) domain, this class computes the normal derivative of the function. The normal is considered as oriented out of the bounded (inner) domain in both inner and outer configurations. The numerical approximation uses piecewise constant (P<sub>0</sub>) approximation on edges of the boundary. Solution is obtained from the GMRES iterative solver without preconditioning. The given data is the vector (instance of class Vect) of piecewise constant values of the harmonic function on the boundary and the returned solution is piecewise constant value of the normal derivative considered either as a Vect instance.

Note

Although the mesh of the inner domain is not necessary to solve the problem, this one must be provided in order to calculate the outward normal.

Author

Rachid Touzani

Copyright

**GNU Lesser Public License** 

# 7.102.2 Constructor & Destructor Documentation

SteklovPoincare2DBE ( Mesh & ms )

Constructor using mesh data.

**Parameters** 

in ms Reference to Mesh instance	
----------------------------------	--

# SteklovPoincare2DBE ( Mesh & ms, Vect< real\_t > & u )

Constructor that solves the Steklov Poincare problem.

This constructor calls member function setMesh and Solve.

#### **Parameters**

ir	ms	Reference to mesh instance.
ir	и	Reference to solution vector. It contains the solution (normal derivative on
		boundary, once problem is solved

# 7.102.3 Member Function Documentation

void setMesh ( Mesh & ms )

set Mesh instance

#### **Parameters**

#### void setExterior ( )

Choose domain of the Laplace equation as exterior one.

By default the domain where the Laplace equation is considered is the interior domain, *i.e.* bounded. This function chooses the exterior of a bounded domain

#### int run ( )

Solve Setklov-Poincare problem.

This member function builds and solves the Steklov-Poincare equation.

# 7.103 Tabulation Class Reference

To read and manipulate tabulated functions.

# **Public Member Functions**

• Tabulation ()

Default constructor.

• Tabulation (string file)

Constructor using file name.

• ∼Tabulation ()

Destructor.

• void setFile (string file)

Set file name.

• real\_t getValue (string funct, real\_t v)

Return the calculated value of the function.

• real\_t getDerivative (string funct, real\_t v)

Return the derivative of the function at a given point.

• real\_t getValue (string funct, real\_t v1, real\_t v2)

Return the calculated value of the function.

• real\_t getValue (string funct, real\_t v1, real\_t v2, real\_t v3)

Return the calculated value of the function.

# 7.103.1 Detailed Description

To read and manipulate tabulated functions.

This class enables reading a tabulated function of one to three variables and calculating the value of the function using piecewise multilinear interpolation.

The file defining the function is an XML file where any function is introduced via the tag " $\leftarrow$  Function".

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# 7.104 Tetra4 Class Reference

Defines a three-dimensional 4-node tetrahedral finite element using P<sub>1</sub> interpolation. Inheritance diagram for Tetra4:



# **Public Member Functions**

• Tetra4 ()

Default Constructor.

• Tetra4 (const Element \*el)

Constructor when data of Element el are given.

• ~Tetra4 ()

Destructor.

• void set (const Element \*el)

Choose element by giving its pointer.

• real\_t Sh (size\_t i, Point < real\_t > s) const

Calculate shape function of node i at a given point s.

• real\_t getVolume () const

Return volume of element.

• Point< real\_t > getRefCoord (const Point< real\_t > &x) const

Return reference coordinates of a point x in element.

bool isIn (const Point < real\_t > &x)

Check whether point x is in current tetrahedron or not.

• real\_t getInterpolate (const Point< real\_t > &x, const LocalVect< real\_t, 4 > &v)

Return interpolated value at point of coordinate x

• Point< real\_t > EdgeSh (size\_t k, Point< real\_t > s)

Return edge shape function.

Point< real\_t > CurlEdgeSh (size\_t k)

Return curl of edge shape function.

real\_t getMaxEdgeLength () const

Return maximal edge length of tetrahedron.

• real\_t getMinEdgeLength () const

Return minimal edge length of tetrahedron.

• std::vector< Point< real\_t >> DSh () const

Calculate partial derivatives of shape functions at element nodes.

# 7.104.1 Detailed Description

Defines a three-dimensional 4-node tetrahedral finite element using  $P_1$  interpolation. The reference element is the right tetrahedron with four unit edges interpolation.

#### Author

Rachid Touzani

# Copyright

**GNU Lesser Public License** 

# 7.104.2 Member Function Documentation

# real\_t Sh ( size\_t i, Point< real\_t > s ) const

Calculate shape function of node i at a given point s. s is a point in the reference tetrahedron.

# Point<real $_t>$ EdgeSh ( size $_tk$ , Point< real $_t>s$ )

Return edge shape function.

#### **Parameters**

in	k	Local edge number for which the edge shape function is computed
in	s	Local coordinates in element

# Remarks

Element edges are ordered as follows: Edge k has end vertices k and k+1

# Point<real $_t>$ CurlEdgeSh ( size $_tk$ )

Return curl of edge shape function.

in	k	Local edge number for which the curl of the edge shape function is computed

#### Remarks

Element edges are ordered as follows: Edge k has end vertices k and k+1

# $std::vector < Point < real_t > > DSh ( ) const$

Calculate partial derivatives of shape functions at element nodes.

#### Returns

Vector of partial derivatives of shape functions  $e.g. \, dsh[i-1].x$ , dsh[i-1].y, are partial derivatives of the i-th shape function

# 7.105 Timer Class Reference

To handle elapsed time counting.

# **Public Member Functions**

• Timer ()

Default constructor.

• ~Timer ()

Destructor.

• bool Started () const

Say if time counter has started.

• void Start ()

Start (or resume) time counting.

• void Stop ()

Stop time counting.

• void Clear ()

Clear time value (Set to zero)

• real\_t get () const

Return elapsed time (in seconds)

real\_t getTime () const

Return elapsed time (in seconds)

# 7.105.1 Detailed Description

To handle elapsed time counting.

This class is to be used when testing program performances. A normal usage of the class is, once an instance is constructed, to use alternatively, Start, Stop and Resume. Elapsed time can be obtained once the member function Stop is called.

# Author

Rachid Touzani

# Copyright

GNU Lesser Public License

# 7.105.2 Member Function Documentation

#### bool Started ( ) const

Say if time counter has started.

Return true if time has started, false if not

#### void Start ( )

Start (or resume) time counting.

This member function is to be used to start or resume time counting

### void Stop ( )

Stop time counting.

This function interrupts time counting. This one can be resumed by the function Start

### real\_t getTime ( ) const

Return elapsed time (in seconds) Identical to get

# 7.106 TimeStepping Class Reference

To solve time stepping problems, i.e. systems of linear ordinary differential equations of the form  $[A2]\{y''\} + [A1]\{y'\} + [A0]\{y\} = \{b\}.$ 

# **Public Member Functions**

• TimeStepping ()

Default constructor.

• TimeStepping (TimeScheme s, real\_t time\_step=theTimeStep, real\_t final\_time=theFinal← Time)

Constructor using time discretization data.

• ∼TimeStepping ()

Destructor.

void set (TimeScheme s, real\_t time\_step=theTimeStep, real\_t final\_time=theFinalTime)

Define data of the differential equation or system.

void setLinearSolver (LinearSolver < real\_t > &ls)

Set reference to LinearSolver instance.

• void setPDE (AbsEqua < real\_t > &eq, bool nl=false)

Define partial differential equation to solve.

void setRK4RHS (Vect< real\_t > &f)

Set intermediate right-hand side vector for the Runge-Kutta method.

void setRK3\_TVDRHS (Vect< real\_t > &f)

Set intermediate right-hand side vector for the TVD Runge-Kutta 3 method.

void setInitial (Vect< real\_t > &u)

Set initial condition for the system of differential equations.

void setInitial (Vect< real\_t > &u, Vect< real\_t > &v)

Set initial condition for a system of differential equations.

void setInitialRHS (Vect< real\_t > &f)

Set initial RHS for a system of differential equations when the used scheme requires it.

void setRHS (Vect< real\_t > &b)

Set right-hand side vector.

• void setBC (Vect< real\_t > &u)

Set vector containing boundary condition to enforce.

• void setNewmarkParameters (real\_t beta, real\_t gamma)

Define parameters for the Newmark scheme.

void setConstantMatrix ()

Say that matrix problem is constant.

void setNonConstantMatrix ()

Say that matrix problem is variable.

• void setLinearSolver (Iteration s=DIRECT\_SOLVER, Preconditioner p=DIAG\_PREC)

Set linear solver data.

void setNLTerm0 (Vect< real\_t > &a0, Matrix< real\_t > &A0)

Set vectors defining a nonlinear first order system of ODEs.

• void setNLTerm (Vect< real\_t > &a0, Vect< real\_t > &a1, Vect< real\_t > &a2)

Set vectors defining a nonlinear second order system of ODEs.

• void setVerbose (int v=0)

*Set verbosity parameter:* 

real\_t runOneTimeStep ()

Run one time step.

void run (bool opt=false)

Run the time stepping procedure.

• void Assembly (const Element &el, real\_t \*b, real\_t \*A0, real\_t \*A1, real\_t \*A2=nullptr)

Assemble element arrays into global matrix and right-hand side.

• void SAssembly (const Side &sd, real\_t \*b, real\_t \*A=nullptr)

Assemble side arrays into global matrix and right-hand side.

LinearSolver < real\_t > & getLSolver ()

Return LinearSolver instance.

# 7.106.1 Detailed Description

To solve time stepping problems, i.e. systems of linear ordinary differential equations of the form  $[A2]\{y''\} + [A1]\{y'\} + [A0]\{y\} = \{b\}.$ 

Author

Rachid Touzani

# Copyright

GNU Lesser Public License

#### Features:

- The system may be first or second order (first and/or second order time derivatives
- The following time integration schemes can be used:

– For first order systems: The following schemes are implemented Forward Euler (value  $\leftarrow$ 

: FORWARD\_EULER)

Backward Euler (value: BACKWARD\_EULER) Crank-Nicolson (value: CRANK\_NICOLSON)

Heun (value: HEUN)

2nd Order Adams-Bashforth (value: *AB2*) 4-th order Runge-Kutta (value: *RK4*)

2nd order Backward Differentiation Formula (value: BDF2)

For second order systems: The following schemes are implemented Newmark (value ← : NEWMARK)

# 7.106.2 Constructor & Destructor Documentation

TimeStepping ( TimeScheme s, real\_t time\_step = theTimeStep, real\_t final\_time = theFinalTime )

Constructor using time discretization data.

#### **Parameters**

in	S	Choice of the scheme: To be chosen in the enumerated variable <i>TimeScheme</i> (see the presentation of the class)
in	time_step	Value of the time step. This value will be modified if an adaptive method is used. The default value for this parameter if the value given by the global variable theTimeStep
in	final_time	Value of the final time (time starts at 0). The default value for this parameter is the value given by the global variable theFinalTime

### 7.106.3 Member Function Documentation

void set ( TimeScheme s, real\_t time\_step = theTimeStep, real\_t final\_time = theFinalTime )

Define data of the differential equation or system.

#### Parameters

in	S	Choice of the scheme: To be chosen in the enumerated variable <i>TimeScheme</i> (see the presentation of the class)
in	time_step	Value of the time step. This value will be modified if an adaptive method is used. The default value for this parameter if the value given by the global variable theTimeStep
in	final_time	Value of the final time (time starts at 0). The default value for this parameter is the value given by the global variable theFinalTime

void setLinearSolver ( LinearSolver < real\_t > & ls )

Set reference to LinearSolver instance.

#### Parameters

in <i>ls</i> Reference to LinearSolver i	instance
--	----------

# void setPDE ( AbsEqua < real\_t > & eq, bool nl = false )

Define partial differential equation to solve.

The used equation class must have been constructed using the Mesh instance

#### **Parameters**

in	eq	Reference to equation instance
in	nl	Toggle to say if the considered equation is linear [Default: 0] or not

# void setRK4RHS ( Vect< real\_t > & f )

Set intermediate right-hand side vector for the Runge-Kutta method.

#### Parameters

in $f$	Vector containing the RHS
--------	---------------------------

# void setRK3\_TVDRHS ( Vect< real\_t > & f )

Set intermediate right-hand side vector for the TVD Runge-Kutta 3 method.

#### **Parameters**

in	f	Vector containing the RHS
----	---	---------------------------

# void setInitial ( Vect< real\_t > & u )

Set initial condition for the system of differential equations.

# Parameters

in	и	Vector containing initial condition for the unknown

#### Remarks

If a second-order differential equation is to be solved, use the the same function with two initial vectors (one for the unknown, the second for its time derivative)

# void setInitial ( Vect< real\_t > & u, Vect< real\_t > & v)

Set initial condition for a system of differential equations.

#### **Parameters**

in	и	Vector containing initial condition for the unknown
in	v	Vector containing initial condition for the time derivative of the unknown

#### Note

This function can be used to provide solution at previous time step if a restarting procedure is used.

This member function is to be used only in the case of a second order system

# void setInitialRHS ( Vect< real\_t > & f )

Set initial RHS for a system of differential equations when the used scheme requires it.

Giving the right-hand side at initial time is somtimes required for high order methods like Runge-Kutta

#### **Parameters**

in	f	Vector containing right-hand side at initial time. This vector is helpful for high order
		methods

#### Note

This function can be used to provide solution at previous time step if a restarting procedure is used.

# void setNewmarkParameters ( real\_t beta, real\_t gamma )

Define parameters for the Newmark scheme.

# Parameters

in	beta	Parameter beta [Default: 0.25]
in	gamma	Parameter gamma [Default: 0.5]

#### void setConstantMatrix( )

Say that matrix problem is constant.

This is useful if the linear system is solved by a factorization method but has no effect otherwise

### void setNonConstantMatrix ( )

Say that matrix problem is variable.

This is useful if the linear system is solved by a factorization method but has no effect otherwise

# void setLinearSolver ( Iteration $s = DIRECT\_SOLVER$ , Preconditioner $p = DIAG\_PREC$ )

Set linear solver data.

#### Parameters

in	S	Solver identification parameter. To be chosen in the enumeration variable Iteration: DIRECT_SOLVER, CG_SOLVER, CGS_SOLVER, BICG_SOLVER, BICG_STAB_SOLVER, GMRES_SOLVER, QMR_SOLVER [Default: DIRECT_SOLVER]
in	р	Preconditioner identification parameter. To be chosen in the enumeration variable Preconditioner:  IDENT_PREC, DIAG_PREC, ILU_PREC [Default: DIAG_PREC]

#### Note

The argument *p* has no effect if the solver is DIRECT\_SOLVER

# void setNLTerm0 ( Vect< real\_t > & a0, Matrix< real\_t > & A0)

Set vectors defining a nonlinear first order system of ODEs. The ODE system has the form a1(u)' + a0(u) = 0

#### **Parameters**

in	a0	Reference to Vect instance defining the 0-th order term
in	A0	Reference to Matrix instance

# void setNLTerm ( Vect< real\_t > & a0, Vect< real\_t > & a1, Vect< real\_t > & a2 )

Set vectors defining a nonlinear second order system of ODEs. The ODE system has the form a2(u)'' + a1(u)' + a0(u) = 0

# Parameters

in	a0	Reference to Vect instance defining the 0-th order term
in	a1	Reference to Vect instance defining the first order term
in	a2	Reference to Vect instance defining the second order term

# void setVerbose ( int v = 0 )

Set verbosity parameter:

- = 0, No output
- = 1, Print step label and time value
- = 2, Print step label, time value, time step and integration scheme

# real\_t runOneTimeStep ( )

Run one time step.

# Returns

Value of new time step if this one is updated

# void run ( bool opt = false )

Run the time stepping procedure.

#### **Parameters**

in	opt	Flag to say if problem matrix is constant while time stepping (true) or not (Default
		value is false)

# Note

This argument is not used if the time stepping scheme is explicit

# void Assembly (const Element & el, real\_t \* b, real\_t \* A0, real\_t \* A1, real\_t \* A2 = nullptr)

Assemble element arrays into global matrix and right-hand side.

This member function is to be called from finite element equation classes

### Parameters

in	el	Reference to Element class	
in	b	Pointer to element right-hand side	
in	A0	Pointer to matrix of 0-th order term (involving no time derivative)	
in	A1	Pointer to matrix of first order term (involving time first derivative)	
in	A2	Pointer to matrix of second order term (involving time second derivative) [Default: nullptr]	

# void SAssembly ( const Side & sd, real\_t \* b, real\_t \* A = nullptr )

Assemble side arrays into global matrix and right-hand side.

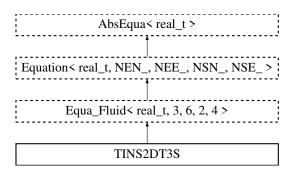
This member function is to be called from finite element equation classes

in	sd	Reference to Side class	
in	b	Pointer to side right-hand side	
in	Α	Pointer to matrix [Default: nullptr]	

# 7.107 TINS2DT3S Class Reference

Builds finite element arrays for transient incompressible fluid flow using Navier-Stokes equations in 2-D domains. Numerical approximation uses stabilized 3-node triangle finite elements for velocity and pressure. 2nd-order projection scheme is used for time integration.

Inheritance diagram for TINS2DT3S:



# **Public Member Functions**

• TINS2DT3S ()

Default Constructor.

• TINS2DT3S (Mesh &mesh)

Constructor using mesh.

TINS2DT3S (Mesh &mesh, Vect< real\_t > &u)

Constructor using mesh and velocity.

• ~TINS2DT3S ()

Destructor.

void setInput (EqDataType opt, Vect< real\_t > &u)

Set equation input data.

• int runOneTimeStep ()

Run one time step.

• int run ()

Run (in the case of one step run)

# 7.107.1 Detailed Description

Builds finite element arrays for transient incompressible fluid flow using Navier-Stokes equations in 2-D domains. Numerical approximation uses stabilized 3-node triangle finite elements for velocity and pressure. 2nd-order projection scheme is used for time integration.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# 7.107.2 Constructor & Destructor Documentation

TINS2DT3S ( Mesh & mesh )

Constructor using mesh.

#### Parameters

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

# TINS2DT3S ( Mesh & mesh, Vect < real\_t > & u )

Constructor using mesh and velocity.

#### **Parameters**

in	mesh	Mesh instance	
in,out	и	Vect instance containing initial velocity. This vector is updated during	
		computations and will therefore contain velocity at each time step	

# 7.107.3 Member Function Documentation

void setInput ( EqDataType opt, Vect< real\_t > & u )

Set equation input data.

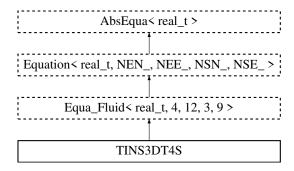
#### Parameters

in	opt	Parameter to select type of input (enumerated values)	
		INITIAL_FIELD: Initial temperature	
		BOUNDARY_CONDITION: Boundary condition (Dirichlet)	
		SOURCE: Body force applied to fluid	
		TRACTION: Heat flux (Neumann boundary condition)	
		VELOCITY_FIELD: Velocity vector (for the convection term)	
in	и	Vector containing input data (Vect instance)	

# 7.108 TINS3DT4S Class Reference

Builds finite element arrays for transient incompressible fluid flow using Navier-Stokes equations in 3-D domains. Numerical approximation uses stabilized 4-node tatrahedral finite elements for velocity and pressure. 2nd-order projection scheme is used for time integration.

Inheritance diagram for TINS3DT4S:



# **Public Member Functions**

• TINS3DT4S ()

Default Constructor.

• TINS3DT4S (Mesh &ms)

Constructor using mesh.

• TINS3DT4S (Mesh &ms, Vect< real\_t > &u)

Constructor using mesh and velocity.

• ~TINS3DT4S ()

Destructor.

void setInput (EqDataType opt, Vect< real\_t > &u)

Set equation input data.

• int runOneTimeStep ()

Run one time step.

# 7.108.1 Detailed Description

Builds finite element arrays for transient incompressible fluid flow using Navier-Stokes equations in 3-D domains. Numerical approximation uses stabilized 4-node tatrahedral finite elements for velocity and pressure. 2nd-order projection scheme is used for time integration.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# 7.108.2 Constructor & Destructor Documentation

TINS3DT4S ( Mesh & ms )

Constructor using mesh.

in	ms	Mesh instance

# TINS3DT4S ( Mesh & ms, Vect< real\_t > & u )

Constructor using mesh and velocity.

# Parameters

in	ms	Mesh instance	
in,out	и	ect instance containing initial velocity. This vector is updated during	
		computations and will therefore contain velocity at each time step	

# 7.108.3 Member Function Documentation

void setInput ( EqDataType opt, Vect< real\_t > & u )

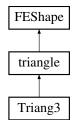
Set equation input data.

# Parameters

in	opt	Parameter to select type of input (enumerated values)	
		INITIAL_FIELD: Initial temperature	
		BOUNDARY_CONDITION_DATA: Boundary condition (Dirichlet)	
		SOURCE_DATA: Heat source	
		FLUX_DATA: Heat flux (Neumann boundary condition). NOT IMPLEMENTED	
		VELOCITY_FIELD: Velocity vector (for the convection term)	
in	и	Vector containing input data (Vect instance)	

# 7.109 Triang3 Class Reference

Defines a 3-Node (P<sub>1</sub>) triangle. Inheritance diagram for Triang3:



# **Public Member Functions**

- Triang3 ()
  - Default Constructor.
- Triang3 (const Element \*el)

Constructor for an element.

• Triang3 (const Side \*sd)

Constructor for a side.

• ~Triang3 ()

Destructor.

• void set (const Element \*el)

Choose element by giving its pointer.

void set (const Side \*sd)

Choose side by giving its pointer.

• real\_t Sh (size\_t i, Point < real\_t > s) const

Calculate shape function of node at a given point.

• std::vector< Point< real\_t >> DSh () const

Return partial derivatives of shape functions of element nodes.

• real\_t getInterpolate (const Point< real\_t > &x, const LocalVect< real\_t, 3 > &v)

Return interpolated value at point of coordinate x

• real\_t check () const

Check element area and number of nodes.

• Point< real\_t > Grad (const LocalVect< real\_t, 3 > &u) const

Return constant gradient vector in triangle.

real\_t getMaxEdgeLength () const

Return maximal edge length of triangle.

real\_t getMinEdgeLength () const

Return minimal edge length of triangle.

# 7.109.1 Detailed Description

Defines a 3-Node  $(P_1)$  triangle.

The reference element is the rectangle triangle with two unit edges.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# 7.109.2 Constructor & Destructor Documentation

Triang3 (const Element \*el)

Constructor for an element.

The constructed triangle is an element in a 2-D mesh.

Triang3 (const Side \* sd)

Constructor for a side.

The constructed triangle is a side in a 3-D mesh.

# 7.109.3 Member Function Documentation

real\_t Sh ( size\_t i, Point< real\_t > s ) const

Calculate shape function of node at a given point.

### **Parameters**

in	i	Label (local) of node	
in	S	Natural coordinates of node where to evaluate	

# $std::vector < Point < real_t > > DSh ( ) const$

Return partial derivatives of shape functions of element nodes.

### Returns

Vector of partial derivatives of shape functions  $e.g. \, dsh[i-1].x$ , dsh[i-1].y, are partial derivatives of the i-th shape function.

# real\_t check ( ) const

Check element area and number of nodes.

### Returns

- > 0: m is the length
- = 0: zero length (=> Error)

# Point<real\_t> Grad ( const LocalVect< real\_t, 3 > & u ) const

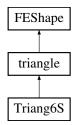
Return constant gradient vector in triangle.

# Parameters

in	и	Local vector for which the gradient is evaluated

# 7.110 Triang6S Class Reference

Defines a 6-Node straight triangular finite element using P<sub>2</sub> interpolation. Inheritance diagram for Triang6S:



# **Public Member Functions**

• Triang6S ()

Default Constructor.

• Triang6S (const Element \*el)

Constructor for an element.

• ∼Triang6S ()

Destructor.

• void Sh (real\_t s, real\_t t, real\_t \*sh) const

Calculate shape functions.

• Point< real\_t > getCenter () const

Return coordinates of center of element.

real\_t getMaxEdgeLength () const

Return maximal edge length of triangle.

real\_t getMinEdgeLength () const

Return minimal edge length of triangle.

void setLocal (real\_t s, real\_t t)

*Initialize local point coordinates in element.* 

• void atMidEdges (std::vector< Point< real\_t >> &dsh, std::vector< real\_t > &w)

Compute partial derivatives of shape functions at mid edges of triangles.

• std::vector< Point< real\_t >> DSh () const

Return partial derivatives of shape functions of element nodes.

# 7.110.1 Detailed Description

Defines a 6-Node straight triangular finite element using P<sub>2</sub> interpolation.

The reference element is the rectangle triangle with two unit edges.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# 7.110.2 Constructor & Destructor Documentation

Triang6S (const Element \* el )

Constructor for an element.

The constructed triangle is an element in a 2-D mesh.

**Parameters** 

in *el* Pointer to Element instance

### 7.110.3 Member Function Documentation

void Sh ( real\_t s, real\_t t, real\_t \*sh ) const

Calculate shape functions.

#### **Parameters**

in	S	Local first coordinate of the point where the gradient of the shape functions are evaluated
in	t	Local second coordinate of the point where the gradient of the shape functions are evaluated
out	sh	Array of of shape functions at (s,t)

### void setLocal ( real\_t s, real\_t t )

Initialize local point coordinates in element.

#### **Parameters**

in	S	Local first coordinate of the point where the gradient of the shape functions are evaluated	
in	t	Local second coordinate of the point where the gradient of the shape functions are evaluated	

# void atMidEdges ( std::vector< Point< real\_t >> & dsh, std::vector< real\_t > & w)

Compute partial derivatives of shape functions at mid edges of triangles.

This member function can be called for integrations using partial derivatives of shape functions and approximated by midedge integration formula

#### **Parameters**

out	dsh	Vector containing partial derivatives of shape functions
out	w	Vector containing weights for the integration formula

# $std::vector < Point < real_t > > DSh ( ) const$

Return partial derivatives of shape functions of element nodes.

#### Returns

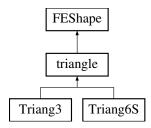
LocalVect instance of partial derivatives of shape functions e.g. dsh(i).x, dsh(i).y, are partial derivatives of the i-th shape function.

# Note

The local point at which the derivatives are computed must be chosen before by using the member function setLocal

# 7.111 triangle Class Reference

Defines a triangle. The reference element is the rectangle triangle with two unit edges. Inheritance diagram for triangle:



# **Public Member Functions**

• triangle ()

Default Constructor.

• triangle (const Element \*el)

Constructor for an element.

• triangle (const Side \*sd)

Constructor for a side.

• virtual ~triangle ()

Destructor.

real\_t getArea ()

Return element area.

• Point< real\_t > getCenter () const

Return coordinates of center of element.

Point< real\_t > getCircumcenter () const

Return coordinates of circumcenter of element.

• real\_t getCircumRadius () const

Return radius of circumscribed circle of triangle.

real\_t getInRadius () const

Return radius of inscribed circle of triangle.

• Point< real\_t > getRefCoord (const Point< real\_t > &x) const

Return reference coordinates of a point x in element.

• real\_t getMaxEdgeLength () const

Return maximal edge length of triangle.

• real\_t getMinEdgeLength () const

Return minimal edge length of triangle.

• bool isIn (const Point< real\_t > &x) const

Check whether point x is in current triangle or not.

• bool isStrictlyIn (const Point< real\_t > &x) const

Check whether point x is strictly in current triangle (not on the boundary) or not.

# 7.111.1 Detailed Description

Defines a triangle. The reference element is the rectangle triangle with two unit edges.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# 7.111.2 Constructor & Destructor Documentation

### triangle (const Element \*el)

Constructor for an element.

The constructed triangle is an element in a 2-D mesh.

# triangle ( const Side \*sd )

Constructor for a side.

The constructed triangle is a side in a 3-D mesh.

# 7.112 Triangle Class Reference

To store and treat a triangle.

Inheritance diagram for Triangle:



# **Public Member Functions**

• Triangle ()

Default constructor.

Triangle (const Point< real\_t > &v1, const Point< real\_t > &v2, const Point< real\_t > &v3, int code=1)

Constructor with vertices and code.

• void setVertex1 (const Point < real\_t > &v)

Assign first vertex of triangle.

void setVertex2 (const Point< real\_t > &v)

Assign second vertex of triangle.

• void setVertex3 (const Point< real\_t > &v)

Assign third vertex of triangle.

real\_t getSignedDistance (const Point< real\_t > &p) const

Return signed distance of a given point from the current triangle.

• Triangle & operator+= (Point< real\_t > a)

Operator +=.

• Triangle & operator+= (real\_t a)

Operator \*=.

# **Additional Inherited Members**

# 7.112.1 Detailed Description

To store and treat a triangle.

# 7.112.2 Constructor & Destructor Documentation

# Triangle ( )

Default constructor.

Constructs a unit triangle with vertices (0,0), (1,0) and (0,1)

Triangle ( const Point< real\_t > & v1, const Point< real\_t > & v2, const Point< real\_t > & v3, int code = 1)

Constructor with vertices and code.

#### **Parameters**

in	v1	Coordinates of first vertex of triangle	
in	<i>v</i> 2	Coordinates of second vertex of triangle	
in	v3	Coordinates of third vertex of triangle	
in	code	Code to assign to the generated figure [Default: 1]	

#### Remarks

Vertices must be given in couterclockwise order

# 7.112.3 Member Function Documentation

real\_t getSignedDistance ( const Point< real\_t > & p ) const [virtual]

Return signed distance of a given point from the current triangle.

The computed distance is negative if p lies in the triangle, positive if it is outside, and 0 on its boundary

### Parameters

in	p	Point <double> instance</double>
TII	P	romic double > mstance

Reimplemented from Figure.

Triangle& operator+= ( Point< real\_t > a )

Operator +=.

Translate triangle by a vector a

Triangle& operator+=  $( real_t a )$ 

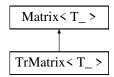
Operator \*=.

Scale triangle by a factor a

# 7.113 TrMatrix $< T_- >$ Class Template Reference

To handle tridiagonal matrices.

Inheritance diagram for TrMatrix< T $_->$ :



### **Public Member Functions**

• TrMatrix ()

Default constructor.

• TrMatrix (size\_t size)

Constructor for a tridiagonal matrix with size rows.

• TrMatrix (const TrMatrix &m)

Copy Constructor.

~TrMatrix ()

Destructor.

• void Identity ()

Define matrix as identity matrix.

• void Diagonal ()

Define matrix as a diagonal one.

• void Diagonal (const T<sub>-</sub> &a)

Define matrix as a diagona one and assign value a to all diagonal entries.

• void Laplace1D (real\_t h)

Define matrix as the one of 3-point finite difference discretization of the second derivative.

• void setSize (size\_t size)

Set size (number of rows) of matrix.

void MultAdd (const Vect< T<sub>-</sub> > &x, Vect< T<sub>-</sub> > &y) const

Multiply matrix by vector x and add result to y.

• void MultAdd (T\_a, const Vect< T\_> &x, Vect< T\_> &y) const

Multiply matrix by vector  $\mathbf{a} * \mathbf{x}$  and add result to  $\mathbf{y}$ .

• void Mult (const Vect<  $T_-> &x$ , Vect<  $T_-> &y$ ) const

Multiply matrix by vector **x** and save result in y.

• void TMult (const Vect<  $T_-$  > &x, Vect<  $T_-$  > &y) const

Multiply transpose of matrix by vector x and save result in y.

• void Axpy (T<sub>-</sub> a, const TrMatrix < T<sub>-</sub> > &m)

Add to matrix the product of a matrix by a scalar.

• void Axpy (T<sub>-</sub> a, const Matrix < T<sub>-</sub> > \*m)

Add to matrix the product of a matrix by a scalar.

• void set (size\_t i, size\_t j, const T\_ &val)

Assign constant val to an entry (i, j) of the matrix.

void add (size\_t i, size\_t j, const T\_ &val)

Add constant val value to an entry (i, j) of the matrix.

• T\_ operator() (size\_t i, size\_t j) const

Operator () (Constant version).

• T<sub>-</sub> & operator() (size<sub>-</sub>t i, size<sub>-</sub>t j)

Operator () (Non constant version).

• TrMatrix< T<sub>-</sub> > & operator= (const TrMatrix< T<sub>-</sub> > &m)

Operator = .

• TrMatrix< T\_> & operator= (const T\_ &x)

 $Operator = Assign \ matrix \ to \ identity \ times \ x.$ 

• TrMatrix< T<sub>-</sub> > & operator∗= (const T<sub>-</sub> &x)

Operator \*=.

• int solve (Vect< T\_> &b, bool fact=true)

Solve a linear system with current matrix (forward and back substitution).

• int solve (const Vect< T\_> &b, Vect< T\_> &x, bool fact=false)

Solve a linear system with current matrix (forward and back substitution).

• T<sub>-</sub> \* get () const

Return C-Array.

• T\_get (size\_t i, size\_t j) const

Return entry (i, j) of matrix.

# 7.113.1 Detailed Description

```
template<class T_>
class OFELI::TrMatrix< T_>
```

To handle tridiagonal matrices.

This class enables storing and manipulating tridiagonal matrices. The template parameter is the type of matrix entries. Any matrix entry can be accessed by the () operator: For instance, if A is an instance of this class, A(i,j) stands for the entry at the i-th row and j-th column, i and j starting from 1. If is difference from i-1, i or i+1, the returned value is 0. Entries of A can be assigned a value by the same operator. Only nonzero entries can be assigned.

**Template Parameters** 

```
T \leftarrow  Data type (double, float, complex<double>, ...)
```

Author

Rachid Touzani

Copyright

GNU Lesser Public License

# 7.114 Vect< T\_> Class Template Reference

To handle general purpose vectors.

Inherits vector $< T_- >$ .

# **Public Member Functions**

• Vect ()

Default Constructor. Initialize a zero-length vector.

• Vect (size\_t n)

Constructor setting vector size.

• Vect (size\_t nx, size\_t ny)

Constructor of a 2-D index vector.

Vect (size\_t nx, size\_t ny, size\_t nz)

Constructor of a 3-D index vector.

• Vect (size\_t n, T\_ \*x)

Create an instance of class *Vect* as an image of a C/C++ array.

• Vect (Grid &g)

Constructor with a Grid instance.

• Vect (Mesh &m, int nb\_dof=0, int dof\_type=NODE\_FIELD)

Constructor with a mesh instance.

Vect (Mesh &m, string name, real\_t t=0.0, int nb\_dof=0, int dof\_type=NODE\_FIELD)

Constructor with a mesh instance giving name and time for vector.

• Vect (const Element \*el, const Vect< T\_> &v)

Constructor of an element vector.

• Vect (const Side \*sd, const Vect< T\_> &v)

Constructor of a side vector.

• Vect (const Vect< T $_->$  &v, const Vect< T $_->$  &bc)

Constructor using boundary conditions.

• Vect (const Vect< T\_> &v, size\_t nb\_dof, size\_t first\_dof)

Constructor to select some components of a given vector.

• Vect (const Vect  $< T_- > &v$ )

Copy constructor.

• Vect (const Vect< T\_> &v, size\_t n)

Constructor to select one component from a given 2 or 3-component vector.

• Vect (size\_t d, const Vect< T\_> &v, const string &name=""")

Constructor that extracts some degrees of freedom (components) from given instance of Vect.

• ~Vect ()

Destructor.

void set (const T<sub>-</sub> \*v, size<sub>-</sub>t n)

Initialize vector with a c-array.

• void select (const Vect< T\_> &v, size\_t nb\_dof=0, size\_t first\_dof=1)

Initialize vector with another Vect instance.

• void set (const string &exp, size\_t dof=1)

Initialize vector with an algebraic expression.

• void set (const string &exp, const Vect< real\_t > &x)

Initialize vector with an algebraic expression.

• void set (Mesh &ms, const string &exp, size\_t dof=1)

Initialize vector with an algebraic expression with providing mesh data.

• void set (const Vect< real\_t > &x, const string &exp)

Initialize vector with an algebraic expression.

• void setMesh (Mesh &m, size\_t nb\_dof=0, size\_t dof\_type=NODE\_FIELD)

Define mesh class to size vector.

• size\_t size () const

Return vector (global) size.

• void setSize (size\_t nx, size\_t ny=1, size\_t nz=1)

Set vector size (for 1-D, 2-D or 3-D cases)

• void resize (size\_t n)

Set vector size.

• void resize (size\_t n, T\_ v)

Set vector size and initialize to a constant value.

• void setDOFType (int dof\_type)

Set DOF type of vector.

• void setDG (int degree=1)

Set Discontinuous Galerkin type vector.

• bool isGrid () const

Say if vector is constructed for a grid.

• size\_t getNbDOF () const

Return vector number of degrees of freedom.

size\_t getNb () const

Return vector number of entities (nodes, elements or sides)

• Mesh & getMesh () const

Return Mesh instance.

• bool WithMesh () const

Return true if vector contains a Mesh pointer, false if not.

- int getDOFType () const
- void setTime (real\_t t)

Set time value for vector.

• real\_t getTime () const

Get time value for vector.

• void setName (string name)

Set name of vector.

• string getName () const

Get name of vector.

• real\_t Norm (NormType t) const

Compute a norm of vector.

• real\_t getNorm1 () const

Calculate 1-norm of vector.

real\_t getNorm2 () const

 $Calculate\ 2\hbox{-}norm\ (Euclidean\ norm)\ of\ vector.$ 

• real\_t getNormMax () const

Calculate Max-norm (Infinite norm) of vector.

• real\_t getWNorm1 () const

Calculate weighted 1-norm of vector The wighted 1-norm is the 1-Norm of the vector divided by its size.

real\_t getWNorm2 () const

Calculate weighted 2-norm of vector.

• T<sub>-</sub> getMin () const

Calculate Min value of vector entries.

• T<sub>-</sub> getMax () const

Calculate Max value of vector entries.

• size\_t getNx () const

Return number of grid points in the x-direction if grid indexing is set.

size\_t getNy () const

Return number of grid points in the y-direction if grid indexing is set.

• size\_t getNz () const

Return number of grid points in the z-direction if grid indexing is set.

void setIJK (const string &exp)

Assign a given function (given by an interpretable algebraic expression) of indices components of vector.

void setNodeBC (Mesh &m, int code, T\_ val, size\_t dof)

Assign a given value to components of vector with given code.

• void setNodeBC (Mesh &m, int code, T\_ val)

Assign a given value to components of vector with given code.

• void setSideBC (Mesh &m, int code, T\_ val, size\_t dof)

 $Assign\ a\ given\ value\ to\ components\ of\ vector\ corresponding\ to\ sides\ with\ given\ code.$ 

void setNodeBC (Mesh &m, int code, const string &exp, size\_t dof)

Assign a given function (given by an interpretable algebraic expression) to components of vector with given code

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

Assign a given function (given by an interpretable algebraic expression) to components of vector with given code.

• void setSideBC (Mesh &m, int code, const string &exp, size\_t dof)

Assign a given function (given by an interpretable algebraic expression) to components of vector corresponding to sides with given code.

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

Assign a given function (given by an interpretable algebraic expression) to components of vector corresponding to sides with given code.

• void setNodeBC (int code, T\_ val, size\_t dof)

Assign a given value to components of vector with given code.

void setNodeBC (int code, T<sub>-</sub> val)

Assign a given value to components of vector with given code.

void setNodeBC (int code, const string &exp, size\_t dof)

Assign a given function (given by an interpretable algebraic expression) to components of vector with given code.

void setNodeBC (int code, const string &exp)

Assign a given function (given by an interpretable algebraic expression) to components of vector with given code.

• void setSideBC (int code, const string &exp, size\_t dof)

Assign a given function (given by an interpre<table algebraic expression) to components of vector with given code.

• void setSideBC (int code, const string &exp)

Assign a given function (given by an interpre<table algebraic expression) to components of vector with given code.

• void setSideBC (int code, T\_ val, size\_t dof)

Assign a given value to components of vector with given code.

• void setSideBC (int code, T\_ val)

Assign a given value to components of vector with given code.

• void removeBC (const Mesh &ms, const Vect< T\_> &v, int dof=0)

Remove boundary conditions.

• void removeBC (const Vect< T\_> &v, int dof=0)

Remove boundary conditions.

• void transferBC (const Vect< T<sub>-</sub> > &bc, int dof=0)

*Transfer boundary conditions to the vector.* 

• void insertBC (Mesh &m, const Vect<  $T_-$  > &v, const Vect<  $T_-$  > &bc, int dof=0)

Insert boundary conditions.

• void insertBC (Mesh &m, const Vect< T<sub>-</sub> > &v, int dof=0)

Insert boundary conditions.

• void insertBC (const Vect< T\_> &v, const Vect< T\_> &bc, int dof=0)

*Insert boundary conditions.* 

• void insertBC (const Vect< T\_> &v, int dof=0)

Insert boundary conditions.

• void Assembly (const Element &el, const Vect< T\_> &b)

Assembly of element vector into current instance.

• void Assembly (const Element &el, const T<sub>-</sub> \*b)

Assembly of element vector (as C-array) into Vect instance.

• void Assembly (const Side &sd, const Vect< T\_> &b)

Assembly of side vector into Vect instance.

• void Assembly (const Side &sd, const T<sub>-</sub> \*b)

Assembly of side vector (as C-array) into Vect instance.

• void getGradient (class Vect< T\_> &v)

Evaluate the discrete Gradient vector of the current vector.

• void getGradient (Vect< Point< T\_> > &v)

Evaluate the discrete Gradient vector of the current vector.

• void getCurl (Vect< T\_> &v)

Evaluate the discrete curl vector of the current vector.

• void getCurl (Vect< Point< T\_> > &v)

Evaluate the discrete curl vector of the current vector.

• void getSCurl (Vect< T\_> &v)

Evaluate the discrete scalar curl in 2-D of the current vector.

• void getDivergence (Vect< T<sub>-</sub> > &v)

*Evaluate the discrete Divergence of the current vector.* 

real\_t getAverage (const Element &el, int type) const

Return average value of vector in a given element.

• Vect< T $_->$  & MultAdd (const Vect< T $_->$  &x, const T $_-$  &a)

Multiply by a constant then add to a vector.

• void Axpy  $(T_a, const Vect < T_s > x)$ 

Add to vector the product of a vector by a scalar.

void set (size\_t i, T\_ val)

Assign a value to an entry for a 1-D vector.

• void set (size\_t i, size\_t j, T\_ val)

Assign a value to an entry for a 2-D vector.

void set (size\_t i, size\_t j, size\_t k, T\_ val)

Assign a value to an entry for a 3-D vector.

• void add (size\_t i, T\_ val)

Add a value to an entry for a 1-index vector.

• void add (size\_t i, size\_t j, T\_ val)

Add a value to an entry for a 2-index vector.

• void add (size\_t i, size\_t j, size\_t k, T\_ val)

```
Assign a value to an entry for a 3-index vector.
```

• void clear ()

Clear vector: Set all its elements to zero.

• T<sub>-</sub> & operator() (size<sub>-</sub>t i)

Operator () (Non constant version)

• T\_operator() (size\_t i) const

Operator () (Constant version)

• T<sub>-</sub> & operator() (size<sub>-</sub>t i, size<sub>-</sub>t j)

Operator () with 2-D indexing (Non constant version, case of a grid vector).

• T\_ operator() (size\_t i, size\_t j) const

*Operator () with 2-D indexing (Constant version).* 

• T<sub>-</sub> & operator() (size<sub>-</sub>t i, size<sub>-</sub>t j, size<sub>-</sub>t k)

Operator () with 3-D indexing (Non constant version).

• T\_ operator() (size\_t i, size\_t j, size\_t k) const

Operator () with 3-D indexing (Constant version).

• Vect< T $_->$  & operator= (const Vect< T $_->$  &v)

Operator = between vectors.

• void operator= (string s)

Operator =

• void setUniform (T\_vmin, T\_vmax, size\_t n)

*Initialize vector entries by setting extremal values and interval.* 

• Vect< T<sub>−</sub> > & operator= (const T<sub>−</sub> &a)

Operator =

• Vect< T $_->$  & operator+= (const Vect< T $_->$  &v)

Operator +=

• Vect< T\_> & operator+= (const T\_ &a)

Operator +=

• Vect< T $_->$  & operator= (const Vect< T $_->$  &v)

Operator -=

• Vect< T\_> & operator== (const T\_ &a)

Operator -=

• Vect< T\_> & operator\*= (const T\_ &a)

Operator \*=

• Vect< T\_> & operator/= (const T\_ &a)

Operator /=

• void push\_back (const T\_ &v)

Add an entry to the vector.

• const Mesh & getMeshPtr () const

Return reference to Mesh instance.

• T\_ operator, (const Vect< T\_ > &v) const

Return Dot (scalar) product of two vectors.

• Vect< complex\_t > getFFT ()

Compute FFT transform of vector.

• Vect< complex\_t > getInvFFT ()

Compute Inverse FFT transform of vector.

# 7.114.1 Detailed Description

```
template<class T_> class OFELI::Vect< T_>
```

To handle general purpose vectors.

Author

Rachid Touzani

Copyright

GNU Lesser Public License

This template class enables defining and manipulating vectors of various data types. It inherits from the class std::vector An instance of class Vect can be:

- A simple vector of given size
- A vector with up to three indices, *i.e.*, an entry of the vector can be a(i), a(i,j) or a(i,j,k). This feature is useful, for instance, in the case of a structured grid
- A vector associate to a finite element mesh. In this case, a constructor uses a reference to the Mesh instance. The size of the vector is by default equal to the number of nodes x the number of degrees of freedom by node. If the degrees of freedom are supported by elements or sides, then the vector is sized accordingly

Operators =, [] and () are overloaded so that one can write for instance:

```
Vect<real_t> u(10), v(10);
v = -1.0;
u = v;
u(3) = -2.0;
```

to set vector  $\mathbf{v}$  entries to -1, copy vector  $\mathbf{v}$  into vector  $\mathbf{u}$  and assign third entry of  $\mathbf{v}$  to -2. Note that entries of  $\mathbf{v}$  are here  $\mathbf{v}(1)$ ,  $\mathbf{v}(2)$ , ...,  $\mathbf{v}(10)$ , *i.e.* vector entries start at index 1.

**Template Parameters** 

```
T \leftarrow  Data type (real_t, float, complex<real_t>, ...)
```

# 7.114.2 Constructor & Destructor Documentation

Vect (  $size_t n$  )

Constructor setting vector size.

in	n	Size of vector

# Vect ( size\_t nx, size\_t ny )

Constructor of a 2-D index vector.

This constructor can be used for instance for a 2-D grid vector

### Parameters

in	пх	Size for the first index
in	ny	Size for the second index

# Remarks

The size of resulting vector is nx\*ny

# Vect ( size\_t nx, size\_t ny, size\_t nz )

Constructor of a 3-D index vector.

This constructor can be used for instance for a 3-D grid vector

#### **Parameters**

in	nx	Size for the first index
in	пу	Size for the second index
in	nz	Size for the third index

# Remarks

The size of resulting vector is nx\*ny\*nz

# Vect ( size\_t n, $T_- * x$ )

Create an instance of class Vect as an image of a C/C++ array.

# Parameters

in	n	Dimension of vector to construct
in	x	C-array to copy

# Vect ( Grid & g )

Constructor with a **Grid** instance.

The constructed vector has as size the total number of grid nodes

in	8	Grid instance

# Vect ( Mesh & m, int $nb\_dof = 0$ , int $dof\_type = NODE\_FIELD$ )

Constructor with a mesh instance.

### Parameters

i	n m	n	Mesh instance
i	n n	ıb_dof	Number of degrees of freedom per node, element or side If nb_dof is set to 0 (default value) the constructor picks this number from the Mesh instance
i	n de	lof_type	Type of degrees of freedom. To be given among the enumerated values: NODE_FIELD, ELEMENT_FIELD, SIDE_FIELD or EDGE_FIELD (Default: NODE_FIELD)

# Vect ( Mesh & m, string name, real\_t t = 0.0, int $nb\_dof = 0$ , int $dof\_type = NODE\_FIELD$ )

Constructor with a mesh instance giving name and time for vector.

### Parameters

in	m	Mesh instance
in	name	Name of the vector
in	t	Time value for the vector
in	nb_dof	Number of degrees of freedom per node, element or side If nb_dof is set to 0 the constructor picks this number from the Mesh instance
in	dof_type	Type of degrees of freedom. To be given among the enumerated values: NODE_FIELD, ELEMENT_FIELD, SIDE_FIELD or EDGE_FIELD (Default: NODE_FIELD)

# Vect ( const Element \* el, const Vect< $T_- > \& v$ )

Constructor of an element vector.

The constructed vector has local numbering of nodes

# Parameters

in	el	Pointer to Element to localize
in	v	Global vector to localize

# Vect ( const Side \* sd, const Vect< $T_- > \& v$ )

Constructor of a side vector.

The constructed vector has local numbering of nodes

in	sd	Pointer to Side to localize
in	v	Global vector to localize

### Vect ( const Vect< $T_-$ > & v, const Vect< $T_-$ > & bc )

Constructor using boundary conditions.

Boundary condition values contained in bc are reported to vector v

### **Parameters**

in	v	Vect instance to update
in	bc	Vect instance containing imposed valued at desired DOF

# Vect ( const Vect< $T_-$ > & v, size\_t $nb\_dof$ , size\_t $first\_dof$ )

Constructor to select some components of a given vector.

#### **Parameters**

in	v	Vect instance to extract from
in	nb_dof	Number of DOF to extract
in	first_dof	First DOF to extract For instance, a choice first_dof=2 and nb_dof=1 means that the second DOF of each node is copied in the vector

# Vect ( const Vect< $T_-$ > & v, size\_t n)

Constructor to select one component from a given 2 or 3-component vector.

### Parameters

in	v	Vect instance to extract from
in	n	Component to extract (must be $> 1$ and $< 4$ or).

# Vect ( size\_t d, const Vect< $T_-$ > & v, const string & name = " ")

Constructor that extracts some degrees of freedom (components) from given instance of Vect.

This constructor enables constructing a subvector of a given Vect instance. It selects a given list of degrees of freedom and put it according to a given order in the instance to construct.

in	d	Integer number giving the list of degrees of freedom. This number is made of n digits where n is the number of degrees of freedom. Let us give an example: Assume that the instance v has 3 DOF by entity (node, element or side). The choice d=201 means that the constructed instance has 2 DOF where the first DOF is the third one of v, and the second DOF is the first one of f v. Consequently, no digit can be larger than the number of DOF the constructed instance. In this example, a choice d=103 would produce an error message.
in	v	Vect instance from which extraction is performed.
in	name	Name to assign to vector instance (Default value is " ").

# Warning

Don't give zeros as first digits for the argument d. The number is in this case interpreted as octal!!

# 7.114.3 Member Function Documentation

void set ( const  $T_- * v$ , size\_t n )

Initialize vector with a c-array.

#### **Parameters**

in	v	c-array (pointer) to initialize Vect
in	n	size of array

# void select ( const Vect< $T_- > & v$ , size\_t $nb\_dof = 0$ , size\_t $first\_dof = 1$ )

Initialize vector with another Vect instance.

### Parameters

in	v	Vect instance to extract from	
in	nb_dof	Number of DOF per node, element or side (By default, 0: Number of degrees	
	-	of freedom extracted from the Mesh instance)	
in	first_dof	First DOF to extract (Default: 1) For instance, a choice first_dof=2 and	
		nb_dof=1 means that the second DOF of each node is copied in the vector	

# void set ( const string & exp, size\_t dof = 1 )

Initialize vector with an algebraic expression.

This function is to be used is a Mesh instance is associated to the vector

#### **Parameters**

in	ехр	Regular algebraic expression that defines a function of x, y, z which are coordinates of nodes and t which is the time value.
in	dof	Degree of freedom to which the value is assigned [Default: 1]

# Warning

If the time variable t is involved in the expression, the time value associated to the vector instance must be defined (Default value is 0) either by using the appropriate constructor or by the member function setTime.

# void set ( const string & exp, const Vect< real\_t > & x )

Initialize vector with an algebraic expression.

This function can be used for instance in 1-D

### Parameters

in	exp	Regular algebraic expression that defines a function of x which are values of
		vector. This expression must use the variable x as coordinate of vector.

# Warning

If the time variable t is involved in the expression, the time value associated to the vector instance must be defined (Default value is 0) either by using the appropriate constructor or by the member function setTime.

# Parameters

	in	х	Vector that defines coordinates
--	----	---	---------------------------------

# void set ( Mesh & ms, const string & exp, size\_t dof = 1 )

Initialize vector with an algebraic expression with providing mesh data.

#### **Parameters**

in	ms	Mesh instance	
in	exp	Regular algebraic expression that defines a function of x, y and z which are coordinates of nodes.	
in	dof	Degree of freedom to which the value is assigned [Default: 1]	

# void set ( const Vect< real\_t > & x, const string & exp )

Initialize vector with an algebraic expression.

### Parameters

in	x	Vect instance that contains coordinates of points
in	exp	Regular algebraic expression that defines a function of x and i which are coordinates. Consider for instance that we want to initialize the Vect instance with the values $v[i] = exp(1+x[i])$ ; then, we use this member function as follows v.set("exp("1+x",x);

# void setMesh ( Mesh & m, size\_t $nb\_dof = 0$ , size\_t $dof\_type = NODE\_FIELD$ )

Define mesh class to size vector.

in	m	Mesh instance
in	nb_dof	Number of degrees of freedom per node, element or side If nb_dof is set to 0
		the constructor picks this number from the Mesh instance

#### **Parameters**

in	dof_type	Parameter to precise the type of degrees of freedom. To be chosen among the
		enumerated values: NODE_FIELD, ELEMENT_FIELD, SIDE_FIELD, EDGE_FIELD
		[Default: NODE_FIELD]

# void setSize ( size\_t nx, size\_t ny = 1, size\_t nz = 1 )

Set vector size (for 1-D, 2-D or 3-D cases)

This function allocates memory for the vector but does not initialize its components

# Parameters

in	nx	Number of grid points in x-direction
in	пу	Number of grid points in y-direction [Default: 1]
in	nz	Number of grid points in z-direction [Default: 1]

### void resize ( $size_t n$ )

Set vector size.

This function allocates memory for the vector but does not initialize its components

### **Parameters**

in $n$ Size of vector
-----------------------

# void resize ( size\_t n, $T_-v$ )

Set vector size and initialize to a constant value.

This function allocates memory for the vector

# Parameters

in	n	Size of vector
in	v	Value to assign to vector entries

# void setDOFType ( int dof\_type )

Set DOF type of vector.

The DOF type combined with number of DOF per component enable determining the size of vector

in	dof_type	Type of degrees of freedom. Value to be chosen among the enumerated
		values: NODE_FIELD, ELEMENT_FIELD, SIDE_FIELD or EDGE_FIELD

# void setDG ( int degree = 1 )

Set Discontinuous Galerkin type vector.

When the vector is associated to a mesh, this one is sized differently if the DG method is used.

### **Parameters**

in	degree	Polynomial degree of the DG method [Default: 1]
----	--------	---

# bool isGrid ( ) const

Say if vector is constructed for a grid.

Vectors constructed for grids are defined with the help of a Grid instance

#### Returns

true if vector is constructed with a Grid instance

# bool WithMesh ( ) const

Return true if vector contains a Mesh pointer, false if not.

A Vect instance can be constructed using mesh information

# int getDOFType ( ) const

Return DOF type of vector

### Returns

dof\_type Type of degrees of freedom. Value among the enumerated values: NODE\_FIELD, ELEMENT\_FIELD, SIDE\_FIELD or EDGE\_FIELD

# real\_t Norm ( NormType t ) const

Compute a norm of vector.

#### **Parameters**

i	n	t	Norm type to compute: To choose among enumerate values: NORM1: 1-norm WNORM1: Weighted 1-norm (Discrete L1-norm) NORM2: 2-norm WNORM2:
			Weighted 2-norm (Discrete L2-norm) NORM_MAX: max norm (Infinity norm)

### Returns

Value of norm

# Warning

This function is available for real valued vectors only

### real\_t getNorm1 ( ) const

Calculate 1-norm of vector.

# 7.114. VECT< T\_> CLASS TEMPLATE REFERENCICHAPTER 7. CLASS DOCUMENTATION

### Remarks

This function is available only if the template parameter is double or complex<double>

# real\_t getNorm2 ( ) const

Calculate 2-norm (Euclidean norm) of vector.

#### Remarks

This function is available only if the template parameter is double or complex<double>

# real\_t getNormMax ( ) const

Calculate Max-norm (Infinite norm) of vector.

#### Remarks

This function is available only if the template parameter is double or complex<double>

# real\_t getWNorm2 ( ) const

Calculate weighted 2-norm of vector.

The weighted 2-norm is the 2-Norm of the vector divided by the square root of its size

# void setIJK ( const string & exp )

Assign a given function (given by an interpretable algebraic expression) of indices components of vector.

This function enable assigning a value to vector entries as function of indices

# Parameters

in   exp   Regular algebraic expres	ion to assign. It must involve the variables $\mathtt{i}$ , $\mathtt{j}$ and/or $\mathtt{k}$ .
-------------------------------------	--

# void setNodeBC ( Mesh & m, int code, T\_val, size\_t dof )

Assign a given value to components of vector with given code. Vector components are assumed nodewise

### Parameters

in	m	Mesh instance
in	code	The value is assigned if the node has this code
in	val	Value to assign
in	dof	Degree of freedom to assign

# void setNodeBC ( Mesh & m, int code, T\_val )

Assign a given value to components of vector with given code.

Vector components are assumed nodewise. Here all dofs of nodes with given code will be assigned

#### **Parameters**

in	m	Mesh instance
in	code	The value is assigned if the node has this code
in	val	Value to assign

# void setSideBC ( Mesh & m, int code, T\_val, size\_t dof )

Assign a given value to components of vector corresponding to sides with given code. Vector components are assumed nodewise

# Parameters

in	m	Instance of mesh
in	code	Code for which nodes will be assigned prescribed value
in	val	Value to prescribe
in	dof	Degree of Freedom for which the value is assigned [default: 1]

# void setNodeBC ( Mesh & m, int code, const string & exp, size\_t dof )

Assign a given function (given by an interpretable algebraic expression) to components of vector with given code.

Vector components are assumed nodewise

### Parameters

in	m	Instance of mesh
in	code	Code for which nodes will be assigned prescribed value
in	ехр	Regular algebraic expression to prescribe
in	dof	Degree of Freedom for which the value is assigned

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

Assign a given function (given by an interpretable algebraic expression) to components of vector with given code.

Vector components are assumed nodewise. Case of 1-DOF problem

in	m	Instance of mesh
in	code	Code for which nodes will be assigned prescribed value
in	ехр	Regular algebraic expression to prescribe

# void setSideBC ( Mesh & m, int code, const string & exp, size\_t dof )

Assign a given function (given by an interpretable algebraic expression) to components of vector corresponding to sides with given code.

Vector components are assumed nodewise

#### **Parameters**

in	m	Instance of mesh
in	code	Code for which nodes will be assigned prescribed value
in	exp	Regular algebraic expression to prescribe
in	dof	Degree of Freedom for which the value is assigned

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

Assign a given function (given by an interpretable algebraic expression) to components of vector corresponding to sides with given code.

Vector components are assumed nodewise. Case of 1-DOF problem

#### **Parameters**

in	m	Instance of mesh
in	code	Code for which nodes will be assigned prescribed value
in	ехр	Regular algebraic expression to prescribe

# void setNodeBC ( int code, T\_val, size\_t dof )

Assign a given value to components of vector with given code.

Vector components are assumed nodewise

### Parameters

in	code	Code for which nodes will be assigned prescribed value
in	val	Value to prescribe
in	dof	Degree of Freedom for which the value is assigned [default: 1]

# void setNodeBC ( int code, T\_val )

Assign a given value to components of vector with given code. Vector components are assumed nodewise. Concerns 1-DOF problems

in	code	Code for which nodes will be assigned prescribed value
in	val	Value to prescribe

# void setNodeBC ( int code, const string & exp, size\_t dof )

Assign a given function (given by an interpretable algebraic expression) to components of vector with given code.

Vector components are assumed nodewise

#### **Parameters**

in	code	Code for which nodes will be assigned prescribed value
in	ехр	Regular algebraic expression to prescribe
in	dof	Degree of Freedom for which the value is assigned [default: 1]

# Warning

This member function is to be used in the case where a constructor with a Mesh has been used

# void setNodeBC ( int code, const string & exp )

Assign a given function (given by an interpretable algebraic expression) to components of vector with given code.

Vector components are assumed nodewise. Concerns 1-DOF problems

# Parameters

in	code	Code for which nodes will be assigned prescribed value
in	exp	Regular algebraic expression to prescribe

### Warning

This member function is to be used in the case where a constructor with a Mesh has been used

# void setSideBC ( int code, const string & exp, size\_t dof )

Assign a given function (given by an interpre<table algebraic expression) to components of vector with given code.

Vector components are assumed nodewise

### Parameters

	in	code	Code for which nodes will be assigned prescribed value
	in	ехр	Regular algebraic expression to prescribe
Ī	in	dof	Degree of Freedom for which the value is assigned

# Warning

This member function is to be used in the case where a constructor with a Mesh has been used

# void setSideBC ( int code, const string & exp )

Assign a given function (given by an interpre<table algebraic expression) to components of vector with given code.

Vector components are assumed nodewise. Case of 1-DOF problem

#### **Parameters**

in	code	Code for which nodes will be assigned prescribed value
in	exp	Regular algebraic expression to prescribe

# Warning

This member function is to be used in the case where a constructor with a Mesh has been used

# void setSideBC ( int code, T\_val, size\_t dof )

Assign a given value to components of vector with given code.

Vector components are assumed nodewise

# Parameters

in	code	Code for which nodes will be assigned prescribed value
in	val	Value to prescribe
in	dof	Degree of Freedom for which the value is assigned

# Warning

This member function is to be used in the case where a constructor with a Mesh has been used

# void setSideBC ( int code, T\_val )

Assign a given value to components of vector with given code.

Vector components are assumed nodewise. Concerns 1-DOF problems

in	code	Code for which nodes will be assigned prescribed value
in	val	Value to prescribe

# Warning

This member function is to be used in the case where a constructor with a Mesh has been used

# void removeBC (const Mesh & ms, const Vect< T $_->$ & v, int dof = 0)

Remove boundary conditions.

This member function copies to current vector a vector where only non imposed DOF are retained.

#### **Parameters**

in	ms	Mesh instance
in	v	Vector (Vect instance to copy from)
in	dof	Parameter to say if all degrees of freedom are concerned (=0, Default) or if only one degree of freedom (dof) is inserted into vector v which has only one degree of freedom

# void removeBC ( const Vect< $T_- > & v$ , int dof = 0)

Remove boundary conditions.

This member function copies to current vector a vector where only non imposed DOF are retained.

# Parameters

in	v	Vector (Vect instance to copy from)
in	dof	Parameter to say if all degrees of freedom are concerned [Default: 0] or if only one degree of freedom (dof) is inserted into vector v which has only one degree of freedom.

# Warning

This member function is to be used in the case where a constructor with a Mesh has been used

# void transferBC ( const Vect< $T_- > \& bc$ , int dof = 0)

Transfer boundary conditions to the vector.

in	bc	Vect instance from which imposed degrees of freedom are copied to current
		instance
in	dof	Parameter to say if all degrees of freedom are concerned (=0, Default) or if only one degree of freedom (dof) is inserted into vector v which has only one degree of freedom.

# void insertBC ( Mesh & m, const Vect< $T_-$ > & v, const Vect< $T_-$ > & bc, int dof = 0 )

Insert boundary conditions.

# Parameters

in	m	Mesh instance.
in	v	Vect instance from which free degrees of freedom are copied to current instance.
in	bc	Vect instance from which imposed degrees of freedom are copied to current instance.
in	dof	Parameter to say if all degrees of freedom are concerned (=0, Default) or if only one degree of freedom (dof) is inserted into vector v which has only one degree of freedom by node or side

# void insertBC ( Mesh & m, const Vect< $T_- > \& v$ , int dof = 0 )

Insert boundary conditions.

DOF with imposed boundary conditions are set to zero.

### **Parameters**

in	m	Mesh instance.
in	v	Vect instance from which free degrees of freedom are copied to current instance.
in	dof	Parameter to say if all degrees of freedom are concerned (=0, Default) or if only one degree of freedom (dof) is inserted into vector v which has only one degree of freedom by node or side

# void insertBC ( const Vect< $T_-$ > & v, const Vect< $T_-$ > & bc, int dof = 0)

Insert boundary conditions.

### Parameters

in	v	Vect instance from which free degrees of freedom are copied to current instance.
in	bc	Vect instance from which imposed degrees of freedom are copied to current
		instance.
in	dof	Parameter to say if all degrees of freedom are concerned (=0, Default) or if only one degree of freedom (dof) is inserted into vector v which has only one degree of freedom by node or side

# void insertBC ( const Vect< $T_-$ > & v, int dof = 0 )

Insert boundary conditions.

DOF with imposed boundary conditions are set to zero.

in	v	Vect instance from which free degrees of freedom are copied to current instance.

#### **Parameters**

in	dof	Parameter to say if all degrees of freedom are concerned (=0, Default) or if only
		one degree of freedom (dof) is inserted into vector v which has only one degree of
		freedom by node or side

# Warning

This member function is to be used in the case where a constructor with a Mesh has been used

# void Assembly (const Element & el, const Vect< $T_-$ > & b)

Assembly of element vector into current instance.

### **Parameters**

in	el	Reference to Element instance
in	b	Local vector to assemble (Instance of class Vect)

# void Assembly (const Element & el, const $T_- * b$ )

Assembly of element vector (as C-array) into Vect instance.

# Parameters

in	el	Reference to Element instance
in	b	Local vector to assemble (C-Array)

# void Assembly ( const Side & sd, const Vect< $T_- > & b$ )

Assembly of side vector into Vect instance.

# Parameters

in	sd	Reference to Side instance
in	b	Local vector to assemble (Instance of class Vect)

# void Assembly (const Side & sd, const $T_- * b$ )

Assembly of side vector (as C-array) into Vect instance.

in	sd	Reference to Side instance
in	b	Local vector to assemble (C-Array)

# void getGradient ( class Vect< T $_->$ & v )

Evaluate the discrete Gradient vector of the current vector.

The resulting gradient is stored in a Vect instance. This function handles node vectors assuming  $P_1$  approximation. The gradient is then a constant vector for each element.

#### **Parameters**

in	v	Vect instance that contains the gradient, where v(n,1), v(n,2) and v(n,3) are
		respectively the x and y and z derivatives at element n.

# void getGradient ( Vect< Point< T\_-> > & v )

Evaluate the discrete Gradient vector of the current vector.

The resulting gradient is stored in an Vect instance. This function handles node vectors assuming P<sub>1</sub> approximation. The gradient is then a constant vector for each element.

#### **Parameters**

in	v	Vect instance that contains the gradient, where $v(n,1).x, v(n,2).y$ and $v(n,3).z$
		are respectively the x and y and z derivatives at element n.

# void getCurl ( Vect< T $_->$ & v )

Evaluate the discrete curl vector of the current vector.

The resulting curl is stored in a Vect instance. This function handles node vectors assuming  $P_1$  approximation. The curl is then a constant vector for each element.

#### **Parameters**

in	v	Vect instance that contains the curl, where $v(n,1)$ , $v(n,2)$ and $v(n,3)$ are
		respectively the x and y and z curl components at element n.

# void getCurl ( Vect< Point< $T_- >$ & v )

Evaluate the discrete curl vector of the current vector.

The resulting curl is stored in a Vect instance. This function handles node vectors assuming  $P_1$  approximation. The curl is then a constant vector for each element.

## Parameters

in	v	Vect instance that contains the curl, where $v(n,1).x, v(n,2).y$ and $v(n,3).z$ are
		respectively the x and y and z curl components at element n.

### void getSCurl ( Vect< $T_-$ > & v )

Evaluate the discrete scalar curl in 2-D of the current vector.

The resulting curl is stored in a Vect instance. This function handles node vectors assuming

# CHAPTER 7. CLASS DOCUMENTATION.114. VECT< T\_> CLASS TEMPLATE REFERENCE

P<sub>1</sub> approximation. The curl is then a constant vector for each element.

### **Parameters**

in	v	Vect instance that contains the scalar curl.
----	---	--

# void getDivergence ( Vect< $T_- > \& v$ )

Evaluate the discrete Divergence of the current vector.

The resulting divergence is stored in a  $\frac{\text{Vect}}{\text{Instance}}$ . This function handles node vectors assuming  $P_1$  approximation. The divergence is then a constant vector for each element.

# Parameters

in	v	Vect instance that contains the divergence.
----	---	---

# real\_t getAverage ( const Element & el, int type ) const

Return average value of vector in a given element.

#### **Parameters**

in	el	Element instance
in	type	Type of element. This is to be chosen among enumerated values: LINE2, TRIANG3, QUAD4 TETRA4, HEXA8, PENTA6

# Vect<T $_->$ & MultAdd ( const Vect< T $_->$ & x, const T $_-$ & a )

Multiply by a constant then add to a vector.

# Parameters

in	x	Vect instance to add
in	а	Constant to multiply before adding

# void Axpy ( $T_- a$ , const Vect< $T_- > \& x$ )

Add to vector the product of a vector by a scalar.

in	а	Scalar to premultiply
in	x	Vect instance by which a is multiplied. The result is added to current instance

# 7.114. $VECT < T_- > CLASS TEMPLATE REFERENCICHAPTER 7$ . CLASS DOCUMENTATION

# void set ( size\_t i, T\_val )

Assign a value to an entry for a 1-D vector.

# Parameters

in	i	Rank index in vector (starts at 1)
in	val	Value to assign

# void set ( size\_t i, size\_t j, T\_val )

Assign a value to an entry for a 2-D vector.

### Parameters

in	i	First index in vector (starts at 1)
in	j	Second index in vector (starts at 1)
in	val	Value to assign

# void set ( size\_t i, size\_t j, size\_t k, T\_val )

Assign a value to an entry for a 3-D vector.

# Parameters

in	i	First index in vector (starts at 1)
in	j	Second index in vector (starts at 1)
in	k	Third index in vector (starts at 1)
in	val	Value to assign

# void add ( size\_t i, T\_val )

Add a value to an entry for a 1-index vector.

### **Parameters**

in	i	Rank index in vector (starts at 1)
in	val	Value to assign

# void add ( size\_t i, size\_t j, T\_val )

Add a value to an entry for a 2-index vector.

in <i>i</i> First index in vector (starts at	1)
--	----

# Parameters

in	j	Second index in vector (starts at 1)
in	val	Value to assign

# void add ( size\_t i, size\_t j, size\_t k, T\_val )

Assign a value to an entry for a 3-index vector.

### Parameters

in	i	First index in vector (starts at 1)
in	j	Second index in vector (starts at 1)
in	k	Third index in vector (starts at 1)
in	val	Value to assign

# $T_{-}$ % operator() ( size\_t i )

Operator () (Non constant version)

### Parameters

in	i	Rank index in vector (starts at 1)
		• v(i) starts at v(1) to v(size())
		• v(i) is the same element as v[i-1]

# $T_-$ operator() ( size\_t i ) const

Operator () (Constant version)

# Parameters

in	i	Rank index in vector (starts at 1)
		• v(i) starts at v(1) to v(size())
		• v(i) is the same element as v[i-1]

# $T_{-}$ operator() ( size\_t i, size\_t j )

Operator () with 2-D indexing (Non constant version, case of a grid vector).

in	i	first index in vector (Number of vector components in the x-grid)
----	---	---

# 7.114. $VECT < T_- > CLASS TEMPLATE REFERENCICHAPTER 7$ . CLASS DOCUMENTATION

### Parameters

in	j	second index in vector (Number of vector components in the y-grid) v(i,j) starts at
		v(1,1) to v(getNx(),getNy())

# $T_{-}$ operator() ( size\_t i, size\_t j ) const

Operator () with 2-D indexing (Constant version).

### **Parameters**

in	i	first index in vector (Number of vector components in the x-grid)
in	j	second index in vector (Number of vector components in the y-grid) $v(i,j)$ starts at $v(1,1)$ to $v(getNx(),getNy())$

# $T_{\infty}$ operator() ( size\_t i, size\_t j, size\_t k )

Operator () with 3-D indexing (Non constant version).

### Parameters

in	i	first index in vector (Number of vector components in the x-grid)
in	j	second index in vector (Number of vector components in the y-grid)
in	k	third index in vector (Number of vector components in the z-grid) v(i,j,k) starts at v(1,1,1) to v(getNx(),getNy(),getNz())

# $T_-$ operator() ( size\_t i, size\_t j, size\_t k ) const

Operator () with 3-D indexing (Constant version).

# Parameters

in	i	first index in vector (Number of vector components in the x-grid)
in	j	second index in vector (Number of vector components in the y-grid)
in	k	third index in vector (Number of vector components in the z-grid) v(i,j,k) starts at v(1,1,1) to v(getNx(),getNy(),getNz())

# void operator= ( string s )

# Operator =

Assign an algebraic expression to vector entries. This operator has the same effect as the member function set(s)

in	S	String defining the algebraic expression as a function of coordinates and time
----	---	--

# CHAPTER 7. CLASS DOCUMENTATION 7.114. VECT < T $_-$ > CLASS TEMPLATE REFERENCE

#### Warning

A Mesh instance must has been introduced before (e.g. by a constructor)

## void setUniform ( T\_vmin, T\_vmax, size\_t n )

Initialize vector entries by setting extremal values and interval.

#### Parameters

in	vmin	Minimal value to assign to the first entry
in	vmax	Maximal value to assign to the lase entry
in	n	Number of points (including extremities)

#### Remarks

The vector has a size of n. It is sized in this function

## Vect<T $_->$ & operator= ( const T $_-$ & a )

Operator =

Assign a constant to vector entries

#### Parameters

in   a   Value to set	
-----------------------	--

## Vect<T $_->$ & operator+= ( const Vect< T $_->$ & v )

Operator +=

Add vector x to current vector instance.

## Parameters

in	v	Vect instance to add to instance
----	---	----------------------------------

# Vect<T $_->$ & operator+= ( const T $_-$ & a )

Operator +=

Add a constant to current vector entries.

#### Parameters

in	а	Value to add to vector entries
----	---	--------------------------------

# Vect<T $_->$ & operator== ( const Vect< T $_->$ & v )

Operator -=

## 7.114. $VECT < T_- > CLASS TEMPLATE REFERENCICHAPTER 7$ . CLASS DOCUMENTATION

#### **Parameters**

in	v	Vect instance to subtract from
----	---	--------------------------------

## Vect<T $_->$ & operator== ( const T $_-$ & a )

Operator -=

Subtract constant from vector entries.

#### Parameters

in	а	Value to subtract from
----	---	------------------------

#### Vect<T $_->$ & operator\*= ( const T $_-$ & a )

Operator \*=

#### **Parameters**

in	а	Value to multiply by
----	---	----------------------

## Vect<T $_->$ & operator/= ( const T $_-$ & a )

Operator /=

# Parameters

in	а	Value to divide by
----	---	--------------------

# void push\_back ( const T\_ & v )

Add an entry to the vector.

This function is an overload of the member function push\_back of the parent class vector. It adjusts in addition some vector parameters

#### Parameters

in	. 7	,	Entry value to add

## $T_-$ operator, ( const Vect< $T_-$ > & v ) const

Return Dot (scalar) product of two vectors.

A typical use of this operator is double a = (v,w) where v and w are 2 instances of Vect<double>

## Parameters

in	v	Vect instance by which the current instance is multiplied

## Vect<complex\_t> getFFT ( )

Compute FFT transform of vector.

This member function computes the FFT (Fast Fourier Transform) of the vector contained in the instance and returns it

#### Returns

Vect<complex<double> > instance containing the FFT

#### Remarks

The size of Vect instance must be a power of two and must not exceed the value of  $2^{\wedge}MA \leftarrow X$ . FFT\_SIZE (This value is set in the header "constants.h")

The Vect instance can be either a Vect<double> or Vec<complex<double> >

#### Vect<complex\_t> getInvFFT ( )

Compute Inverse FFT transform of vector.

This member function computes the inverse FFT (Fast Fourier Transform) of the vector contained in the instance and returns it

#### Returns

Vect<complex<double> > instance containing the FFT

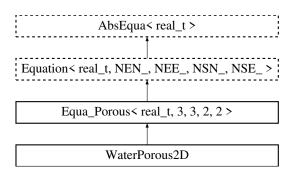
#### Remarks

The size of Vect instance must be a power of two and must not exceed the value of  $2^{\wedge}MA \leftarrow X$ . FFT\_SIZE (This value is set in the header "constants.h")

The Vect instance can be either a Vect<double> or Vec<complex<double> >

# 7.115 WaterPorous2D Class Reference

To solve water flow equations in porous media (1-D) Inheritance diagram for WaterPorous2D:



## **Public Member Functions**

• WaterPorous2D ()

 $Default\ Constructor.$ 

• WaterPorous2D (Mesh &ms)

Constructor.

~WaterPorous2D ()

Destructor.

• void setCoef (real\_t cw, real\_t phi, real\_t rho, real\_t Kx, real\_t Ky, real\_t mu)

Set constant coefficients.

• void Mass ()

Add mass term contribution the element matrix.

• void Mobility ()

Add mobility term contribution the element matrix.

void BodyRHS (const Vect< real\_t > &bf)

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

void BoundaryRHS (const Vect< real\_t > &sf)

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

## **Additional Inherited Members**

## 7.115.1 Detailed Description

To solve water flow equations in porous media (1-D)

To solve water flow equations in porous media (2-D)

Class WaterPorous2D solves the fluid flow equations of water or any incompressible or slightly compressible fluid in a porous medium in two-dimensional configurations.

Porous media flows are modelled here by the Darcy law. The water, or any other fluid is considered as slightly compressible, i.e., its compressibility coefficient is constant.

Space discretization uses the  $P_1$  (2-Node line) finite element method. Time integration uses class TimeStepping that provides various well known time integration schemes.

Class WaterPorous2D solves the fluid flow equations of water or any incompressible or slightly compressible fluid in a porous medium in two-dimensional configurations.

Porous media flows are modelled here by the Darcy law. The water, or any other fluid is considered as slightly compressible, i.e., its compressibility coefficient is constant.

Space discretization uses the P<sub>1</sub> (3-Node triangle) finite element method. Time integration uses class TimeStepping that provides various well known time integration schemes.

## 7.115.2 Constructor & Destructor Documentation

#### WaterPorous2D ( )

Default Constructor.

Constructs an empty equation.

#### WaterPorous2D ( Mesh & ms )

Constructor.

This constructor uses mesh and reservoir information

#### Parameters

in ms	Mesh instance
-------	---------------

## 7.115.3 Member Function Documentation

void setCoef ( real\_t cw, real\_t phi, real\_t rho, real\_t Kx, real\_t Ky, real\_t mu )

Set constant coefficients.

#### **Parameters**

in	сw	Compressibility coefficient
in	phi	Porosity
in	rho	Density
in	Kx	x-Absolute permeability
in	Ку	y-Absolute permeability
in	ти	Viscosity

# ${f void\ BodyRHS}$ ( ${f const\ Vect}{<}\,{f real\_t}>\&\,bf$ ) [virtual]

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

#### Parameters

	in	bf	Vector containing source at nodes.	
--	----	----	------------------------------------	--

Reimplemented from Equa\_Porous< real\_t, 3, 3, 2, 2 >.

# void BoundaryRHS ( const Vect< real t > & sf ) [virtual]

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

#### Parameters

in s	sf Ve	ctor containing source at nodes.
------	-------	----------------------------------

Reimplemented from Equa\_Porous< real\_t, 3, 3, 2, 2 >.

# Index

	A 1 A 11
A OFFILE 4.40	AxbAssembly
OFELI, 149	OFELI::Equation, 328
AB2	AxialForce
OFELI, 147	OFELI::Beam3DL2, 206
ADAMS_BASHFORTH	Axpy
OFELI, 147	OFELI::BMatrix, 214
AUX_INPUT_FIELD_1	OFELI::DMatrix, 248
OFELI, 147	OFELI::DSMatrix, 265
AUX_INPUT_FIELD_2	OFELI::Matrix, 425
OFELI, 147	OFELI::SkMatrix, 537
AUX_INPUT_FIELD_3	OFELI::SkSMatrix, 546
OFELI, 147	OFELI::Vect, 604
AUX_INPUT_FIELD_4	OFELI, 160, 161, 163, 164
OFELI, 147	Utilities, 67
AbsEqua $< T >$ , 197	
ActiveElementLoop	b
Finite Element Mesh, 89	OFELI, 149
Add	BACKWARD_EULER
	OFELI, 147
OFELI::Element, 296–298	BAND
OFELI::Mesh, 440, 441, 445	OFELI, 148
OFELI::Side, 528	ВСТуре
add	OFELI, 148
OFELI::DMatrix, 247	BDF2
OFELI::DSMatrix, 263	OFELI, 147
OFELI::SkMatrix, 538	BICG_SOLVER
OFELI::SkSMatrix, 547	OFELI, 148
OFELI::Vect, 605, 606	BICG_STAB_SOLVER
OFELI, 161	OFELI, 148
AddMidNodes	BMatrix
OFELI::Mesh, 448	OFELI::BMatrix, 213
Analysis	BMatrix $< T >$ , 212
OFELI, 147	BODY_FORCE
areClose	OFELI, 146
Utilities, 49	
Assembly	BOUNDARY_CONDITION
OFELI::EigenProblemSolver, 279	OFELI, 146
OFELI::Matrix, 425	BOUNDARY_FORCE
	OFELI, 147
OFELL: TimeStepping, 567	BOUNDARY_TRACTION
OFELI::Vect, 602	OFELI, 146
atGauss	BSpline
OFELI::Hexa8, 350	Utilities, 64
OFELI::Quad4, 521	BUOYANCY
atMidEdges	OFELI, 146
OFELI::Triang6S, 576	banner

Utilities, 65	build
Bar2DL2, 200	OFELI::Bar2DL2, 202
OFELI::Bar2DL2, 201	OFELI::EC2D1T3, 268
Beam3DL2, 203	OFELI::Equa_Laplace, 309
OFELI::Beam3DL2, 204, 205	OFELI::Equa_Porous, 311, 312
BendingMoment	OFELI::Equa_Therm, 317
OFELI::Beam3DL2, 206	OFELI::LaplaceDG2DP1, 389
BiCGStab	OFELI::NSP2DQ41, 481
Solver, 113, 114	buildEigen
BiCG	OFELI::Beam3DL2, 206
Solver, 111, 112	OFELI::Laplace1DL2, 379
BiotSavart, 207	OFELI::Laplace2DT3, 385
OFELI::BiotSavart, 208	OFELI::Laplace2DT6, 387
BodyRHS	Of EELEaplace25 10, 307
OFELI::DC1DL2, 222	CAPACITY
OFELI::DC2DT3, 226, 227	OFELI, 146
OFELI::DC2DT6, 231	CATCH_EXCEPTION
OFELI::DC2D10, 231 OFELI::DC3DAT3, 234	Utilities, 40
	CG_SOLVER
OFELL::DC3DT4, 238	OFELI, 148
OFELL:Elas2DQ4, 283	CGS_SOLVER
OFELI::Elas2DT3, 287	
OFELI::Elas3DH8, 290	OFELI, 148
OFELI::Elas3DT4, 292	CGS
OFELI::Equa_Laplace, 309	Solver, 117, 118
OFELI::Equa_Porous, 311	CONSISTENT_CAPACITY
OFELI::Equa_Therm, 316	OFELI, 146
OFELI::Laplace1DL2, 379	CONSISTENT_MASS
OFELI::Laplace1DL3, 382	OFELI, 146
OFELI::Laplace2DT3, 384	CONTACT_BC
OFELI::Laplace2DT6, 386	OFELI, 149
OFELI::WaterPorous2D, 612	CONTACT_DISTANCE
BoundaryNodeLoop	OFELI, 147
Finite Element Mesh, 90	CONTACT
BoundaryRHS	OFELI, 146
OFELI::DC2DT3, 227	CONVECTION
OFELI::DC2DT6, 231	OFELI, 146
OFELI::DC3DAT3, 234	CRANK_NICOLSON
OFELI::DC3DT4, 238, 239	OFELI, 147
OFELI::Elas2DQ4, 283	Capacity
OFELI::Elas2DT3, 287	OFELI::DC1DL2, 221
OFELI::Elas3DH8, 290	OFELI::DC2DT3, 225
OFELI::Elas3DT4, 292	OFELI::DC2DT6, 230
OFELI::Equa_Laplace, 309	OFELI::DC3DAT3, 233
OFELI::Equa_Porous, 311	OFELI::DC3DT4, 237
OFELI::Equa_Therm, 316	OFELI::Equa_Therm, 316
OFELI::Laplace1DL2, 380	CG
OFELI::Laplace1DL3, 382	Solver, 115, 116
OFELI::Laplace2DT3, 384	check
OFELI::Laplace2DT6, 387	OFELI::Material, 421
OFELI::WaterPorous2D, 612	OFELI::Triang3, 574
BoundarySideLoop	OFELI, 171
Finite Element Mesh, 90	
	check_error
Brick, 216	OFELI::FMM2D, 339
OFELI::Brick, 217	OFELI::FMM3D, 341

OFELI::FMMSolver, 342	OFELI, 148
checkSturm	DIFFUSION
OFELI::EigenProblemSolver, 280	OFELI, 146
Circle, 218	DILATATION
OFELI::Circle, 219	OFELI, 146
clear	DILU_PREC
Utilities, 67	OFELI, 148
Code	DIRECT_SOLVER
Finite Element Mesh, 95, 96	OFELI, 148
Conservation Law Equations, 23	DISPLACEMENT_FIELD
Contact	OFELI, 147
OFELI::Elas2DT3, 287	dLine
ContactPressure	OFELI::Figure, 336
OFELI::Elas2DT3, 287	DMatrix
	OFELI::DMatrix, 243, 244
Contains	
OFELL::Element, 298, 299	DMatrix $< T >$ , 240
OFELI::Side, 530	DOF
contains	OFELI::Edge, 272
OFELI::IPF, 374	OFELI::Node, 475
Convection	OFELI::Side, 527
OFELI::DC1DL2, 222	DP1toP1
OFELI::DC2DT3, 226	OFELI::Reconstruction, 522
OFELI::DC2DT6, 231	DSMatrix
OFELI::DC3DT4, 237, 238	OFELI::DSMatrix, 260, 261
Converged	DSMatrix $< T >$ , 258
Global Variables, 84	DSh
Coord	OFELI::Line2, 400
Finite Element Mesh, 95	OFELI::Line3, 401
createBoundarySideList	OFELI::Penta6, 504
OFELI::Mesh, 443	OFELI::Tetra4, 560
createInternalSideList	OFELI::Triang3, 574
OFELI::Mesh, 444	OFELI::Triang6S, 576
CurlEdgeSh	Deactivate
OFELI::Tetra4, 559	OFELI, 169, 170
OI LLI Ictiu 1, 507	DeformMesh
DC1DL2, 219	Finite Element Mesh, 97
OFELI::DC1DL2, 221	Delete
DC2DT3, 223	OFELI::Mesh, 446
OFELI::DC2DT3, 224, 225	DeleteElement
DC2DT6, 228	OFELI::Mesh, 445
OFELI::DC2DT6, 230	DeleteNode
DC3DAT3, 232	OFELI::Mesh, 445
OFELI::DC3DAT3, 233	DeleteSide
DC3DT4, 235	OFELI::Mesh, 445
OFELI::DC3DT4, 236	Deviator
DENSE	OFELI::Elas2DT3, 286
OFELI, 148	DG, 239
DEVIATORIC	OFELI::DG, 240
OFELI, 146	DiagBC
DGElementAssembly	OFELI::Equation, 321
OFELI::Equation, 325, 326	DiagPrescribe
DIAG_PREC	OFELI::SkMatrix, 539, 540
OFELI, 148	OFELI, 157
DIAGONAL	Diffusion

OFELI::DC1DL2, 221	Electric
OFELI::DC2DT3, 225	OFELI::EC2D1T3, 269
OFELI::DC2DT6, 230	Electromagnetics, 24
OFELI::DC3DAT3, 234	Element, 292
OFELI::DC3DT4, 237	OFELI::Element, 295, 296
·	
Dilatation	element_assembly
OFELI::Elas2DT3, 286	General Purpose Equations, 18–20
Discrepancy	ElementAssembly
Utilities, 54	OFELI::Equation, 324, 325, 327
Distance	ElementList, 302
Utilities, 49, 53	ElementLoop
Domain, 253	Finite Element Mesh, 89
OFELI::Domain, 255	ElementNodeCoordinates
Dot	OFELI::Equation, 323
Utilities, 67	ElementNodeVector
Vector and Matrix, 74, 78	OFELI::Equation, 321, 322
vector and wattry, 74, 70	ElementNodeVectorSingleDOF
E2T	ě .
	OFELI::Equation, 322
OFELI::PhaseChange, 505	ElementSideVector
EC2D1T3, 266	OFELI::Equation, 322
OFELI::EC2D1T3, 268	ElementVector
EC2D2T3, 269	OFELI::Equation, 322
OFELI::EC2D2T3, 270	ElementsAreDOF
EIGEN	OFELI::Mesh, 449
OFELI, 147	Ellipse, 303
ELECTRIC	OFELI::Ellipse, 304
OFELI, 146	EnthalpyToTemperature
ESTIM_ND_JUMP	OFELI::PhaseChange, 505
OFELI::Estimator, 330	EqDataType
ESTIM_ZZ	OFELI, 146
OFELI::Estimator, 330	Equa_Electromagnetics< T_, NEN_, NEE_, N←
EVAL	$SN_{-}$ , $NSE_{-}$ >, 305
Utilities, 40	Equa_Fluid
EXTERNAL_BOUNDARY	OFELI::Equa_Fluid, 307
OFELI::Side, 527	Equa_Fluid< T_, NEN_, NEE_, NSN_, NSE_ >,
Edge, 271	306
OFELI::Edge, 272	Equa_Laplace
EdgeList, 273	* *
•	OFELI::Equa_Laplace, 309
EdgeLoop	Equa_Laplace< T_, NEN_, NEE_, NSN_, NSE_
Finite Element Mesh, 90	>, 307
EdgeSh	Equa_Porous
OFELI::Tetra4, 559	OFELI::Equa_Porous, 311
EdgesAreDOF	Equa_Porous< T_, NEN_, NEE_, NSN_, NSE_>,
OFELI::Mesh, 449	310
EigenProblemSolver, 275	Equa_Solid
OFELI::EigenProblemSolver, 276, 277	OFELI::Equa_Solid, 314
Elas2DQ4, 280	Equa_Solid < T_, NEN_, NEE_, NSN_, NSE_ >,
OFELI::Elas2DQ4, 282	312
Elas2DT3, 284	Equa_Therm
OFELI::Elas2DT3, 285, 286	OFELI::Equa_Therm, 316
Elas3DH8, 288	Equa_Therm< T_, NEN_, NEE_, NSN_, NSE_>,
OFELI::Elas3DH8, 289	314
Elas3DT4, 290	Equal
OFELI::Elas3DT4, 291	Utilities, 68
•	•

Equation	ActiveElementLoop, 89
OFELI::Equation, 320	BoundaryNodeLoop, 90
Equation < T_, NEN_, NEE_, NSN_, NSE_ >, 317	BoundarySideLoop, 90
Estimator, 329	Code, 95, 96
OFELI::Estimator, 330	Coord, 95
EstimatorType	DeformMesh, 97
OFELI::Estimator, 330	EdgeLoop, 90
eval	ElementLoop, 89
OFELI::FMM2D, 338	GRAPH_MEMORY, 89
	•
OFELI::FMM3D, 340	getMaxElementMeasure, 101
execute	getMaxSideMeasure, 102
OFELI::FastMarching2D, 333	getMaxSize, 100
ExtendSpeed	getMeanElementMeasure, 102
OFELI::FMM2D, 338	getMeanSideMeasure, 102
OFELI::FMM3D, 340	getMinElementMeasure, 101
OFELI::FMMSolver, 342	getMinSideMeasure, 101
	getMinSize, 100
FE_2D_3N	Label, 92–94
OFELI, 148	MeshToMesh, 98, 99
FE_2D_4N	NodeInElement, 105
OFELI, 148	NodeInSide, 105
FE_2D_6N	NodeLabel, 94
OFELI, 148	NodeLoop, 90
FE_3D_4N	operator<<, 91, 92, 106
OFELI, 148	1
FE_3D_8N	operator-, 91
OFELI, 148	operator==, 96, 97
	operator&&, 91
FE_3D_AXI_3N	operator    , 91
OFELI, 148	setBoundaryNodeCodes, 103
FEShape, 333	setBoundarySideCodes, 104
OFELI::FEShape, 334	setElementCodes, 104
FEType	setNodeCodes, 103
OFELI, 147	setSideCodes, 103
FIRST_ORDER_METHOD	SideInElement, 105
OFELI::Muscl, 458	SideLoop, 90
FLUX	TheEdge, 89
OFELI, 147	TheElement, 89
FMM2D, 337	TheNode, 89
OFELI::FMM2D, 337, 338	theNodeLabel, 90
FMM3D, 339	TheSide, 89
OFELI::FMM3D, 339, 340	Fluid Dynamics, 25
FMMSolver, 341	Forward
OFELI::FMMSolver, 341	
FORWARD_EULER	OFELL ICPC2DT 250
	OFELI::ICPG2DT, 359
OFELI, 147	OFELI::LCL1D, 392
Factor	OFELI::LCL2DT, 395
OFELI::LocalMatrix, 412	OFELI::LCL3DT, 398
FactorAndSolve	Funct, 342
OFELI::LocalMatrix, 413	OFELI::Funct, 343, 344
OFELI::Matrix, 428	Function
FastMarching2D, 331	OFELI::MyNLAS, 465
OFELI::FastMarching2D, 332	•
Figure, 335	GAUSS_LEGENDRE
Finite Element Mesh, 85	OFELI, 149
•	•

	_
GMRES_SOLVER	getBamg
OFELI, 148	Utilities, 56
GMRes	getBC
Solver, 119, 120	OFELI::IPF, 371
GRADIENT	getBF
	•
OFELI::OptSolver, 494	OFELI::IPF, 371
GRAPH_MEMORY	getBoundaryNodes
Finite Element Mesh, 89	OFELI::Mesh, 444
Gauss, 344	getBoundarySides
OFELI::Gauss, 345	OFELI::Mesh, 443
genMesh	getCode
OFELI::Domain, 255	OFELI::Edge, 273
•	OFELI::Node, 476
General Purpose Equations, 17	
element_assembly, 18–20	OFELI::Side, 529
side_assembly, 20, 21	OFELI, 169
get	getColHeight
OFELI::Domain, 255	OFELI::SkMatrix, 539
OFELI::IPF, 374, 375	OFELI::SkSMatrix, 548
OFELI::Mesh, 441	getColInd
·	© .
OFELI::NLASSolver, 472	OFELI, 161
OFELI::Prescription, 519	getColumn
OFELI::SkMatrix, 541	OFELI::DMatrix, 245
OFELI::SkSMatrix, 550	OFELI::DSMatrix, 261, 262
OFELI, 151, 152, 163	getComplex
getActiveElement	OFELI::IPF, 373
OFELI::Mesh, 451	getComplexPar
	· .
getAllEdges	OFELI::IPF, 372
OFELI::Mesh, 444	getCoord
getAllSides	OFELI::Node, 476
OFELI::Mesh, 443	getCurl
getArray	OFELI::Vect, 603
OFELI::DMatrix, 253	getDOFType
getArraySize	OFELI::Vect, 594
9	·
OFELI::IPF, 374	getData
getAuxFile	OFELI::IPF, 371
OFELI::IPF, 376	getDerivative
getAverage	OFELI, 153
OFELI::Vect, 604	getDet
getB1	OFELI::FEShape, 334
OFELI::BiotSavart, 210	getDiag
getB2	OFELI::Matrix, 425
OFELI::BiotSavart, 210	getDisp
getB3	OFELI::Beam3DL2, 205
OFELI::BiotSavart, 210	getDivergence
getBC1	OFELI::Vect, 604
OFELI::BiotSavart, 211	getDouble
getBC2	OFELI::IPF, 373
	•
OFELI::BiotSavart, 211	getDoublePar
getBC3	OFELI::IPF, 372
OFELI::BiotSavart, 211	getEasymesh
getBCFile	Utilities, 56
OFELI::IPF, 376	getEdgeLabel
getBFFile	OFELI::Mesh, 451
OFELI::IPF, 376	getEigenVector
O1 DD1II 1 , 07 0	geningen rector

OFELI::EigenProblemSolver, 280	OFELI::IPF, 371
getElementLabel	getMeanElementMeasure
OFELI::Mesh, 451	Finite Element Mesh, 102
getElementNeighborElements	getMeanSideMeasure
OFELI::Mesh, 444	Finite Element Mesh, 102
getElementWiseIndex	getMeasure
OFELI::Estimator, 331	OFELI::Element, 300
getEquation	OFELI::Side, 530
OFELI::MyOpt, 467	getMesh
getFFT	OFELI::AbsEqua, 198
OFELI::Vect, 610	Utilities, 55
getFirstSideLabel	getMeshFile
OFELI::Partition, 502	OFELI::IPF, 375
getGambit	getMinElementMeasure
Utilities, 57	Finite Element Mesh, 101
getGmsh	getMinSideMeasure
Utilities, 57	Finite Element Mesh, 101
getGradient	getMinSize
OFELI::Vect, 602, 603	Finite Element Mesh, 100
getInit	getMomentum
OFELI::IPF, 371	OFELI::ICPG1D, 353
getInitFile	getName
OFELI::IPF, 376	OFELI::Material, 421
getInnerProduct	getNbAcc
OFELI::LocalMatrix, 413	OFELI::OptSolver, 498
getIntPar	getNbBoundarySides
OFELI::IPF, 372	OFELI::Mesh, 448
getInteger	getNbConnectInSubMesh
OFELI::IPF, 373	OFELI::Partition, 502
getInternalEnergy	getNbConnectOutSubMesh
OFELI::ICPG1D, 354	OFELI::Partition, 502
getInterpolate	getNbInternalSides
OFELI::Line2, 400	OFELI::Mesh, 448
getInvFFT	getNbIter
OFELI::Vect, 610	OFELI::IPF, 371
getLevel	getNbNeigEl
OFELI::Node, 477	OFELI::Node, 476
getList	getNbNeigElements
OFELI::Mesh, 450	OFELI::Element, 300
getLocal	getNbOutOfBounds
OFELI::LocalVect, 417	OFELI::OptSolver, 498
getLocalPoint	getNbSteps
OFELI::FEShape, 334	OFELI::IPF, 371
getMach	getNeigEl
OFELI::ICPG1D, 354	OFELI::Node, 476
getMatlab	getNeighborElement
Utilities, 59	OFELI::Element, 300
getMaxElementMeasure	OFELI::Side, 529
Finite Element Mesh, 101	getNetgen
getMaxSideMeasure	Utilities, 59
Finite Element Mesh, 102	getNodeLabel
getMaxSize	OFELI::Edge, 273
Finite Element Mesh, 100	OFELI::Mesh, 450
getMaxTime	getNodeLabelInMesh

OFELI::Partition, 501	getSideWiseIndex
getNodeLabelInSubMesh	OFELI::Estimator, 331
OFELI::Partition, 501	getSignedDistance
getNodeNeighborElements	OFELI::Brick, 217
OFELI::Mesh, 444	OFELI::Circle, 219
getNodeWiseIndex	OFELI::Ellipse, 304
OFELI::Estimator, 331	OFELI::Figure, 336
getNorm1	OFELI::Polygon, 513
OFELI::Vect, 594	OFELI::Rectangle, 524
getNorm2	OFELI::Sphere, 551
OFELI::Vect, 595	OFELI::Triangle, 579
getNormMax	getSolution
OFELI::Vect, 595	OFELI::MeshAdapt, 455
	OFELI::OptSolver, 499
getNormal	-
OFELI::Side, 530	getSolutionBB
getNy	OFELI::MeshAdapt, 455
OFELI, 168	getSolutionbb
getNz	OFELI::MeshAdapt, 455
OFELI, 168	getSoundSpeed
getOtherNeighborElement	OFELI::ICPG1D, 354
OFELI::Side, 529	getStresses
getOutput	OFELI::Bar2DL2, 202
OFELI::IPF, 370	getString
getPlot	OFELI::IPF, 372
OFELI::IPF, 371	getStringPar
getPlotFile	OFELI::IPF, 372
OFELI::IPF, 376	getSubMesh
getPointDoublePar	OFELI::Partition, 501, 502
OFELI::IPF, 372	getTemperature
getPrescriptionFile	OFELI::OptSolver, 498
OFELI::IPF, 376	getTetgen
getProject	Utilities, 60
OFELI::IPF, 375	getTime
getRefCoord	OFELI::Timer, 561
OFELI::Line2, 400	getTimeDerivative
getRestartFile	OFELI::ODESolver, 491
OFELI::IPF, 376	
_	getTimeStep OFELI::IPF, 371
getRow OFFI InDMatrix, 245	
OFELI::DMatrix, 245 OFELI::DSMatrix, 262	getTolerance OFELI::IPF, 371
· ·	
getSCurl	getTotalEnergy
OFELI::Vect, 603	OFELI::ICPG1D, 354
getSFFile	getTriangle
OFELI::IPF, 376	Utilities, 60
getSave	getUnitNormal
OFELI::IPF, 370	OFELI::Element, 300
getSaveFile	OFELI::Side, 530
OFELI::IPF, 376	getValue
getSecondSideLabel	OFELI, 152, 153
OFELI::Partition, 502	getWNorm2
getSF	OFELI::Vect, 595
OFELI::IPF, 371	getXYZ
getSideLabel	OFELI::Node, 476
OFELI::Mesh, 451	getP
	-

OFELI::ICPG2DT, 360	OFELI::ICPG2DT, 358
getR	ICPG3DT, 361
OFELI::ICPG2DT, 360	OFELI::ICPG3DT, 363
getV	IDENT_PREC
OFELI::ICPG2DT, 360	OFELI, 148
Global Variables, 82	IDENTITY
Converged, 84	OFELI, 148
InitPetsc, 84	ILU_PREC
MaxNbIterations, 83	OFELI, 148
	INITIAL_AUX_1
NbTimeSteps, 83	
theEdge, 83	OFELI, 146
theElement, 82	INITIAL AUX 2
theFinalTime, 84	OFELI, 146
theIteration, 83	INITIAL_AUX_3
theNode, 82	OFELI, 146
theSide, 83	INITIAL_AUX_4
theStep, 83	OFELI, 146
theTime, 84	INITIAL_FIELD
theTimeStep, 84	OFELI, 146
theTolerance, 84	INITIAL
Verbosity, 83	OFELI, 146
Grad	INTERNAL_BOUNDARY
OFELI::DC2DT3, 227, 228	OFELI::Side, 527
OFELI::DC3DAT3, 235	INTERNAL_SIDE
OFELI::DC3DT4, 239	OFELI::Side, 527
OFELI::Hexa8, 351	IOField, 366
OFELI::Quad4, 521	OFELI, 149, 150
OFELI::Triang3, 574	IPF, 367
Gradient Gradient	OFELI::IPF, 370
OFELI::MyNLAS, 465	Imag Vector and Matrix, 79
OFELI::MyOpt, 467	InitPetsc
Grid, 345	
OFELI, 165, 166	Global Variables, 84
GS	Initialize
Solver, 121	OFELI::Muscl2DT, 462
	Input/Output, 31
HEAT_FLUX	MAX_ARRAY_SIZE, 31
OFELI, 146	MAX_INPUT_STRING_LENGTH, 31
HEAT_SOURCE	MAX_NB_PAR, 31
OFELI, 146	insertBC
HEUN	OFELI::Vect, 600, 601
OFELI, 147	insertCircle
HLL_SOLVER	OFELI::Domain, 257
OFELI::Muscl, 459	insertLine
HLLC_SOLVER	OFELI::Domain, 256, 257
OFELI::Muscl, 459	insertRequiredEdge
Heat Transfer, 30	OFELI::Domain, 258
HelmholtzBT3, 348	insertRequiredVertex
OFELI::HelmholtzBT3, 349	OFELI::Domain, 257
Hexa8, 349	insertSubDomain
TICAGU, JT/	OFELI::Domain, 258
ICPC1D 351	insertVertex
ICPG1D, 351	
OFELI::ICPG1D, 353	OFELI::Domain, 256
ICPG2DT, 356	IntegND

OFELI::EC2D1T3, 269	LMass
Integration, 363	OFELI::Bar2DL2, 202
OFELI::Integration, 364	OFELI::Elas2DT3, 286
IntegrationScheme	OFELI::Equa_Solid, 314
OFELI, 149	LOAD
Interface Problems, 26	OFELI, 146
Invert	LORENTZ_FORCE
OFELI::LocalMatrix, 413	OFELI, 146
isActive	LUMPED_CAPACITY
OFELI, 170	OFELI, 146
isCloseTo	LUMPED_MASS
OFELI::Point, 509	OFELI, 146
isFactorized	Label
OFELI::Matrix, 428	Finite Element Mesh, 92–94
isGrid	Laplace equation, 27
OFELI::Vect, 594	Laplace1DL2, 378
IsIn	OFELI::Laplace1DL2, 379
OFELI::Element, 302	Laplace1DL3, 380
isOnBoundary	OFELI::Laplace1DL3, 381
OFELI::Edge, 273	Laplace1D
OFELI::Element, 300	OFELI, 156, 163
OFELI::Node, 477	Laplace2DT3, 382
OFELI::Side, 530	OFELI::Laplace2DT3, 383, 384
Iter	
	Laplace2DT6, 385
OFELI, 171	OFELI::Laplace2DT6, 386
Iter $< T >$ , 377	Laplace2D
Iteration	OFELI, 156
OFELI, 148	LaplaceDG2DP1, 387
IterationLoop	OFELI::LaplaceDG2DP1, 388
Solver, 111	Limiter
	OFELI::Muscl, 458
Jacobi	Line2, 398
Solver, 122	OFELI::Line2, 399
Joule Heating	Line3, 400
OFELI::DC2DT3, 228	LinearExchange
LCLAD 200	OFELI::DC2DT3, 226
LCL1D, 390	LinearSolver
LCL2DT, 392	OFELI::LinearSolver, 403–405
OFELI::LCL2DT, 393	LinearSolver $< T >$ , 402
LCL3DT, 395	LocalMatrix
OFELI::LCL3DT, 396	OFELI::LocalMatrix, 409, 410
LCapacity	LocalMatrix< T_, NR_, NC_ >, 407
OFELI::DC1DL2, 221	LocalNodeVector
OFELI::DC2DT3, 225	
	OFELI::Equation, 321
OFELI::DC2DT6, 230	LocalVect
OFELI::DC3DAT3, 233	OFELI::LocalVect, 416
OFELI::DC3DT4, 237	LocalVect $<$ T <sub>-</sub> , N <sub>-</sub> $>$ , 414
OFELI::Equa_Therm, 316	Localize
LEAP_FROG	OFELI::LocalMatrix, 410, 411
OFELI, 147	OFELI::LocalVect, 417
LEFT_RECTANGLE	
OFELI, 149	MAGNETIC
LF_SOLVER	OFELI, 146
OFELI::Muscl, 459	MASS
Orediniviusei, <del>1</del> 07	IVIAUU

OFELI, 146	Vector and Matrix, 79
MAX_ARRAY_SIZE	Mult
Input/Output, 31	OFELI::DMatrix, 247
MAX_ÎNPUT_STRING_LENGTH	OFELI::LocalMatrix, 412
Input/Output, 31	OFELI::SkMatrix, 538
MAX_LIMITER	OFELI::SkSMatrix, 547
OFELI::Muscl, 459	OFELI, 159
MAX_NB_EQUATIONS	MultAdd
Solver, 110	OFELI::DMatrix, 247
MAX_NB_INPUT_FIELDS	OFELI::DSMatrix, 265
Solver, 110	OFELI::LocalMatrix, 412
MAX_NB_MESHES	OFELI::SkMatrix, 537, 538
Solver, 110	OFELI::SkSMatrix, 547
MAX_NB_PAR	OFELI::Vect, 604
Input/Output, 31	OFELI, 160
MAX_SOLVER	MultAddScal
OFELI::Muscl, 459	OFELI::LocalMatrix, 412
MID_RECTANGLE	Muscl, 456
OFELI, 149	Muscl1D, 460
MINMOD_LIMITER	Muscl2DT, 461
OFELI::Muscl, 459	Muscl3DT, 463
MOBILITY	MyNLAS, 464
OFELI, 146	OFELI::MyNLAS, 465
MULTI_SLOPE_M_METHOD	MyOpt, 466
OFELI::Muscl, 458	OFELI::MyOpt, 466
MULTI_SLOPE_Q_METHOD	Of EELWiyOpt, 100
OFELI::Muscl, 458	NELDER_MEAD
Magnetic Magnetic	OFELI::OptSolver, 494
OFELI::EC2D1T3, 268	NEWMARK
Mass	OFELI, 147
OFELI::Bar2DL2, 202	NLASSolver, 467
OFELI::Bas2DT3, 286	OFELI::NLASSolver, 469
OFELI::Equa_Solid, 314	NONE
Material, 419	OFELI, 147
	NORM1
OFELI::Material, 421	OFELI, 195
Matrix	NORM2
OFELI::Matrix, 424	
Matrix $<$ T $>$ , 422	OFELI, 195
MatrixType	NORM_MAX
OFELI, 148	OFELI, 195
MaxNbIterations	NSP2DQ41, 479
Global Variables, 83	OFELI::NSP2DQ41, 480
Media	NbTimeSteps
OFELI::Elas2DT3, 286	Global Variables, 83
OFELI::Elas3DT4, 291	Node, 473
Mesh, 431	OFELI::Node, 474
OFELI::Mesh, 436–440	NodeInElement
MeshAdapt, 451	Finite Element Mesh, 105
OFELI::MeshAdapt, 454	NodeInSide
MeshToMesh	Finite Element Mesh, 105
Finite Element Mesh, 98, 99	NodeLabel
Method	Finite Element Mesh, 94
OFELI::Muscl, 458	NodeList, 477
Modulus	NodeLoop

Finite Element Mesh, 90	getBC2, 211
NodesAreDOF	getBC3, 211
OFELI::Mesh, 449	run, 211
Norm	setBoundary, 210
OFELI::Vect, 594	setCurrentDensity, 209
NormType	setMagneticInduction, 209
OFELI, 195	setPermeability, 209
Normalize	OFELI::Brick
OFELI::Point, 509	Brick, 217
Nrm2	getSignedDistance, 217
Utilities, 67	operator+=, 218
NumberEquations	setBoundingBox, 217
OFELI::Mesh, 443	OFELI::Circle
	Circle, 219
ODESolver, 481	getSignedDistance, 219
OFELI::ODESolver, 484	operator+=, 219
OFELI::AbsEqua	OFELI::DC1DL2
getMesh, 198	BodyRHS, 222
setLinearSolver, 199	Capacity, 221
setMatrixType, 199	Convection, 222
setSolver, 198	DC1DL2, 221
solveLinearSystem, 199, 200	Diffusion, 221
OFELI::BMatrix	LCapacity, 221
Axpy, 214	setInput, 222
BMatrix, 213	OFELI::DC2DT3
operator*=, 215	BodyRHS, 226, 227
operator(), 214	BoundaryRHS, 227
operator+=, 215	Capacity, 225
operator=, 215	Convection, 226
setLU, 215	DC2DT3, 224, 225
setSize, 214	Diffusion, 225
solve, 215, 216	Grad, 227, 228
OFELI::Bar2DL2	JouleHeating, 228
Bar2DL2, 201	LCapacity, 225
build, 202	LinearExchange, 226
getStresses, 202	Periodic, 227
LMass, 202	setInput, 228
Mass, 202	OFELI::DC2DT6
Stiffness, 202	BodyRHS, 231
OFELI::Beam3DL2	BoundaryRHS, 231
AxialForce, 206	Capacity, 230
Beam3DL2, 204, 205	Convection, 231
BendingMoment, 206	DC2DT6, 230
buildEigen, 206	Diffusion, 230
getDisp, 205	LCapacity, 230
set, 205	OFELI::DC3DAT3
ShearForce, 206	BodyRHS, 234
TwistingMoment, 206	BoundaryRHS, 234
OFELI::BiotSavart	Capacity, 233
BiotSavart, 208	DC3DAT3, 233
getB1, 210	Diffusion, 234
getB2, 210	Grad, 235
getB3, 210	LCapacity, 233
getBC1, 211	OFELI::DC3DT4
g <del></del> / <del></del>	- · · · · · · · · · · · · · · · · · · ·

624 OFELI's Reference Guide

	BodyRHS, 238	solve, 265, 266
	BoundaryRHS, 238, 239	TMult, 265
	Capacity, 237	OFELI::DG
	Convection, 237, 238	DG, 240
	DC3DT4, 236	OFELI::Domain
	Diffusion, 237	Domain, 255
	Grad, 239	genMesh, 255
	LCapacity, 237	get, 255
	Periodic, 239	insertCircle, 257
	LI::DMatrix	
OFE		insertLine, 256, 257
	add, 247	insertRequiredEdge, 258
	Axpy, 248	insertRequiredVertex, 257
	DMatrix, 243, 244	insertSubDomain, 258
	getArray, 253	insertVertex, 256
	getColumn, 245	operator*=, 256
	getRow, 245	setOutputFile, 258
	Mult, 247	OFELI::EC2D1T3
	MultAdd, 247	build, 268
	operator*=, 252	EC2D1T3, 268
	operator(), 249, 250	Electric, 269
	operator+=, 252	IntegND, 269
	operator-=, 252, 253	Magnetic, 268
	operator=, 252	setData, 268
	reset, 246	OFELI::EC2D2T3
	set, 246	EC2D2T3, 270
	setColumn, 246	OFELI::Edge
	setDiag, 244	DOF, 272
	setLU, 250	Edge, 272
		S .
	setQR, 248	getCode, 273
	setRow, 246	getNodeLabel, 273
	setSize, 244	isOnBoundary, 273
	setTransLU, 250	operator(), 273
	setTransQR, 248	setCode, 273
	solve, 250, 251	setDOF, 273
	solveQR, 249	OFELI::EdgeList
	solveTrans, 251, 252	selectCode, 274
	solveTransQR, 249	unselectCode, 274
	TMult, 247	OFELI::EigenProblemSolver
OFE	LI::DSMatrix	Assembly, 279
	add, 263	checkSturm, 280
	Axpy, 265	EigenProblemSolver, 276, 277
	DSMatrix, 260, 261	getEigenVector, 280
	getColumn, 261, 262	run, 278
	getRow, 262	runSubSpace, 279
	MultAdd, 265	SAssembly, 279
	operator(), 264	setMatrix, 278
	operator+=, 264	setPDE, 278
	operator, 264	setSubspaceDimension, 279
	set, 261	setTolerance, 280
	setColumn, 263	OFELI::Elas2DQ4
	setDiag, 263	BodyRHS, 283
	setLDLt, 264	BoundaryRHS, 283
	setRow, 263	Elas2DQ4, 282
	setSize, 261	PlaneStrain, 282

PlaneStress, 282	OFELI::Equa_Laplace
Strain, 283	BodyRHS, 309
Stress, 283	BoundaryRHS, 309
OFELI::Elas2DT3	build, 309
BodyRHS, 287	Equa_Laplace, 309
BoundaryRHS, 287	OFELI::Equa_Porous
Contact, 287	BodyRHS, 311
ContactPressure, 287	BoundaryRHS, 311
Deviator, 286	build, 311, 312
Dilatation, 286	Equa_Porous, 311
Elas2DT3, 285, 286	run, 312
LMass, 286	OFELI::Equa_Solid
Mass, 286	Equa_Solid, 314
Media, 286	LMass, 314
Periodic, 288	Mass, 314
Reaction, 287	OFELI::Equa_Therm
Strain, 288	BodyRHS, 316
Stress, 288	BoundaryRHS, 316
OFELI::Elas3DH8	build, 317
BodyRHS, 290	
	Capacity, 316
BoundaryRHS, 290 Elas3DH8, 289	Equa_Therm, 316
OFELI::Elas3DT4	LCapacity, 316
	setStab, 316
BodyRHS, 292	OFELI::Equation
BoundaryRHS, 292 Elas3DT4, 291	AxbAssembly, 328
Media, 291	DGElementAssembly, 325, 326
OFELI::Element	DiagBC, 321 ElementAssembly, 324, 325, 327
Add, 296–298	ElementNodeCoordinates, 323
Contains, 298, 299 Element, 295, 296	ElementNodeVector, 321, 322 ElementNodeVectorSingleDOF, 322
getMeasure, 300	ElementSideVector, 322
getNbNeigElements, 300	ElementVector, 322
getNeighborElement, 300	Equation, 320
getUnitNormal, 300	LocalNodeVector, 321
IsIn, 302	setMaterialProperty, 328
isOnBoundary, 300	SideAssembly, 326–328
operator(), 298, 300	SideNodeCoordinates, 324
Replace, 297	SideNodeVector, 321
set, 298	SideVector, 323
setChild, 302	updateBC, 320
setCode, 296, 298	OFELI::Estimator
setDOF, 298	ESTIM_ND_JUMP, 330
setLabel, 296	ESTIM_ZZ, 330
setSide, 300	Estimator, 330
OFELI::ElementList	EstimatorType, 330
selectLevel, 303	getElementWiseIndex, 331
unselectCode, 303	getNodeWiseIndex, 331
OFELI::Ellipse	getSideWiseIndex, 331
Ellipse, 304	setSolution, 330
getSignedDistance, 304	setType, 330
operator+=, 304, 305	OFELI::FEShape
OFELI::Equa_Fluid	FEShape, 334
Equa_Fluid, 307	getDet, 334

626 OFELI's Reference Guide

getLocalPoint, 334	OFELI::ICPG3DT
Sh, 334	ICPG3DT, 363
OFELI::FMM2D	setReconstruction, 363
check_error, 339	OFELI::IPF
eval, 338	contains, 374
ExtendSpeed, 338	get, 374, 375
FMM2D, 337, 338	getArraySize, 374
OFELI::FMM3D	getAuxFile, 376
check_error, 341	getBCFile, 376
eval, 340	getBFFile, 376
ExtendSpeed, 340	getBC, 371
FMM3D, 339, 340	getBF, 371
OFELI::FMMSolver	getComplex, 373
check_error, 342	getComplexPar, 372
ExtendSpeed, 342	getData, 371
FMMSolver, 341	getDouble, 373
OFELI::FastMarching2D	getDoublePar, 372
execute, 333	getInit, 371
FastMarching2D, 332	getInitFile, 376
OFELI::Figure	getIntPar, 372
dLine, 336	getInteger, 373
getSignedDistance, 336	getMaxTime, 371
OFELI::Funct	getMeshFile, 375
Funct, 343, 344	getNbIter, 371
operator=, 344	getNbSteps, 371
OFELI::Gauss	getOutput, 370
Gauss, 345	getPlot, 371
setTriangle, 345	getPlotFile, 376
OFELI::HelmholtzBT3	getPointDoublePar, 372
HelmholtzBT3, 349	getPrescriptionFile, 376
OFELI::Hexa8	getProject, 375
atGauss, 350	getRestartFile, 376
Grad, 351	getSFFile, 376
setLocal, 350	getSave, 370
OFELI::ICPG1D	getSaveFile, 376
Forward, 353	getSF, 371
getInternalEnergy, 354	getString, 372
getMach, 354	getStringPar, 372
getMomentum, 353	getTimeStep, 371
getSoundSpeed, 354	getTolerance, 371
getTotalEnergy, 354	IPF, 370
ICPG1D, 353	OFELI::Integration
setBC, 354, 355	Integration, 364
setInOutflowBC, 356	run, 366
setInitialCondition, 354	setFunction, 365
OFELI::ICPG2DT	setQuadrilateral, 365
Forward, 359	setScheme, 365
getP, 360	setTriangle, 365
getR, 360	OFELI::Iter
getV, 360	setMaxIter, 377
ICPG2DT, 358	setTolerance, 377
setBC, 359, 360	setVerbose, 378
setReconstruction, 359	OFELI::LCL1D
setSolver, 359	Forward, 392

runOneTimeStep, 391	OFELI::LinearSolver
setBC, 391, 392	LinearSolver, 403–405
setInitialCondition, 391	set, 406
setVelocity, 392	setMatrix, 405, 406
OFELI::LCL2DT	setMaxIter, 405
Forward, 395	setSolver, 406
LCL2DT, 393	solve, 406, 407
runOneTimeStep, 394	OFELI::LocalMatrix
setBC, 394	Factor, 412
setInitialCondition, 394	FactorAndSolve, 413
setVelocity, 395	getInnerProduct, 413
OFELI::LCL3DT	Invert, 413
Forward, 398	LocalMatrix, 409, 410
LCL3DT, 396	Localize, 410, 411
setBC, 397	Mult, 412
setInitialCondition, 397	MultAdd, 412
setVelocity, 397, 398	MultAddScal, 412
OFELI::Laplace1DL2	operator*, 411
BodyRHS, 379	operator*=, 412
BoundaryRHS, 380	operator(), 410
buildEigen, 379	operator+=, 411
Laplace1DL2, 379	operator-=, 411
setBoundaryCondition, 380	operator/=, 412
setTraction, 380	operator=, 411
OFELI::Laplace1DL3	solve, 413
BodyRHS, 382	Symmetrize, 412
BoundaryRHS, 382	OFELI::LocalVect
Laplace1DL3, 381	getLocal, 417
setTraction, 382	LocalVect, 416
OFELI::Laplace2DT3	Localize, 417
BodyRHS, 384	operator*=, 419
BoundaryRHS, 384	operator(), 418
buildEigen, 385	operator+=, 418
Laplace2DT3, 383, 384	operator,, 419
Post, 385	operator-=, 418
OFELI::Laplace2DT6	operator/=, 419
BodyRHS, 386	operator=, 418
BoundaryRHS, 387	operator[], 418
buildEigen, 387	OFELI::Material
Laplace2DT6, 386	check, 421
OFELI::LaplaceDG2DP1	getName, 421
build, 389	Material, 421
LaplaceDG2DP1, 388	set, 421
•	OFELI::Matrix
run, 389 set, 388, 389	
	Assembly, 425
Smooth, 389	Axpy, 425
OFELI::Line2	FactorAndSolve, 428
DSh, 400	getDiag, 425
getInterpolate, 400	isFactorized, 428
getRefCoord, 400	Matrix, 424
Line2, 399	operator*=, 430
Sh, 400	operator(), 429
OFELI::Line3	operator+=, 430
DSh, 401	operator-=, 430

operator=, 430	SidesAreDOF, 449
operator[], 430	OFELI::MeshAdapt
Prescribe, 426, 427	getSolution, 455
PrescribeSide, 427	getSolutionBB, 455
reset, 425	getSolutionbb, 455
set, 428	MeshAdapt, 454
setDiagonal, 425	run, 456
solve, 427	saveMbb, 455
OFELI::Mesh	setMaxNbVertices, 454
Add, 440, 441, 445	setNoKeep, 455
AddMidNodes, 448	setNoScaling, 455
createBoundarySideList, 443	setRatio, 454
createInternalSideList, 444	setRelaxation, 454
Delete, 446	setTheta, 455
DeleteElement, 445	OFELI::Muscl
DeleteNode, 445	FIRST_ORDER_METHOD, 458
DeleteSide, 445	HLL_SOLVER, 459
EdgesAreDOF, 449	HLLC_SOLVER, 459
ElementsAreDOF, 449	LF_SOLVER, 459
get, 441	Limiter, 458
getActiveElement, 451	MAX_LIMITER, 459
getAllEdges, 444	MAX_SOLVER, 459
getAllSides, 443	MINMOD_LIMITER, 459
getBoundaryNodes, 444	MULTI_SLOPE_M_METHOD, 458
getBoundarySides, 443	MULTI_SLOPE_Q_METHOD, 458
getEdgeLabel, 451	Method, 458
getElementLabel, 451	ROE_SOLVER, 459
getElementNeighborElements, 444	RUSANOV_SOLVER, 459
getList, 450	SUPERBEE_LIMITER, 459
getNbBoundarySides, 448	setCFL, 459
getNbInternalSides, 448	setLimiter, 460
getNodeLabel, 450	setMethod, 460
getNodeNeighborElements, 444	setReconstruction, 460
getSideLabel, 451	setReferenceLength, 459
Mesh, 436–440	setTimeStep, 459
NodesAreDOF, 449	setVerbose, 460
•	
NumberEquations, 443	SolverType, 459
operator*=, 441	VANALBADA_LIMITER, 459
put, 449	VANLEER_LIMITER, 459
Refine, 451	VFROE_SOLVER, 459
RenumberEdge, 447	OFELI::Muscl2DT
RenumberElement, 447	Initialize, 462
RenumberNode, 446	setReconstruction, 462
RenumberSide, 447	OFELI::Muscl3DT
Reorder, 445	setReconstruction, 464
Rescale, 448	OFELI::MyNLAS
save, 450	Function, 465
set, 448, 449	Gradient, 465
setDOFSupport, 442	MyNLAS, 465
setDim, 440	OFELI::MyOpt
setList, 447	getEquation, 467
setMaterial, 444	Gradient, 467
setNbDOFPerNode, 442	MyOpt, 466
setPointInDomain, 443	Objective, 467

.T 465	ATT 404
setEquation, 467	setNbEq, 484
OFELI::NLASSolver	setNewmarkParameters, 489
get, 472	setNonConstantMatrix, 490
NLASSolver, 469	setRHS, 489
setDf, 471	setRK4RHS, 486
setFunction, 470	setTolerance, 490
setGradient, 470, 471	setF, 485
setInitial, 472	OFELI::OptSolver
setMaxIter, 470	GRADIENT, 494
setPDE, 471	getNbAcc, 498
setTolerance, 470	getNbOutOfBounds, 498
setf, 471	getSolution, 499
OFELI::NSP2DQ41	getTemperature, 498
build, 481	NELDER MEAD, 494
NSP2DQ41, 480	OptMethod, 494
Periodic, 480	OptSolver, 494
runOneTimeStep, 481	run, 498
setInput, 480	SIMULATED_ANNEALING, 494
setPenalty, 480	setBC, 495
OFELI::Node	setGradient, 496
DOF, 475	setLowerBound, 497
getCode, 476	setLowerBounds, 497
getCoord, 476	setObjective, 496
getLevel, 477	setOptClass, 496
getNbNeigEl, 476	setOptMethod, 495
getNeigEl, 476	setSAOpt, 497
getXYZ, 476	setTolerance, 498
isOnBoundary, 477	setUpperBound, 496
Node, 474	setUpperBounds, 496
setCode, 475	setVerbosity, 497
setCoord, 475	TRUNCATED_NEWTON, 494
setDOF, 476	OFELI::Partition
setLevel, 477	getFirstSideLabel, 502
setOnBoundary, 476	getNbConnectInSubMesh, 502
OFELI::NodeList	getNbConnectOutSubMesh, 502
selectCode, 478	getNodeLabelInMesh, 501
selectCoordinate, 478	getNodeLabelInSubMesh, 501
unselectCode, 478	getSecondSideLabel, 502
OFELI::ODESolver	getSubMesh, 501, 502
getTimeDerivative, 491	Partition, 501
ODESolver, 484	put, 503
run, 491	OFELI::Penta6
runOneTimeStep, 491	DSh, 504
seODEVectors, 488, 489	Penta6, 504
set, 484	setLocal, 504
setCoef, 484, 485	OFELI::PhaseChange
setConstantMatrix, 490	E2T, 505
setDF, 486	EnthalpyToTemperature, 505
setInitial, 486, 487	OFELI::Point
setInitialRHS, 487, 488	isCloseTo, 509
setLinear, 485	Normalize, 509
setLinearSolver, 490	operator!=, 509
setMatrices, 488	operator*=, 508
setMaxIter, 490	operator(), 508
	1

630 OFELI's Reference Guide

operator+=, 508	getNormal, 530
operator,, 509	getOtherNeighborElement, 529
operator-=, 508	getUnitNormal, 530
operator/=, 508	INTERNAL_BOUNDARY, 527
operator=, 508	INTERNAL_SIDE, 527
operator==, 508	isOnBoundary, 530
operator[], 508	operator(), 529
Point, 507	set, 528
OFELI::Point2D	setChild, 530
operator!=, 512	setCode, 528
operator*=, 512	setDOF, 528
operator(), 511	Side, 527
operator+=, 511, 512	SideType, 527
operator-=, 511, 512	OFELI::SideList
•	selectCode, 532
operator 511 512	
operator=, 511, 512	unselectCode, 533
operator==, 512	OFELI::SkMatrix
operator[], 511	add, 538
Point2D, 511	Axpy, 537
OFELI::Polygon	DiagPrescribe, 539, 540
getSignedDistance, 513	get, 541
operator+=, 514	getColHeight, 539
Polygon, 513	Mult, 538
setVertices, 513	MultAdd, 537, 538
OFELI::Prec	operator*=, 540
Prec, 515, 516	operator(), 539
setMatrix, 516, 517	operator+=, 540
setType, 516	operator=, 540
solve, 517	set, 537
TransSolve, 517	setDOF, 536
OFELI::Prescription	setLU, <u>540</u>
get, 519	setMesh, 536
Prescription, 518	setSkyline, 536
OFELI::Quad4	SkMatrix, 535, 536
atGauss, 521	solve, 541
Grad, 521	TMult, 538
Quad4, 520	TMultAdd, 537
setLocal, 521	OFELI::SkSMatrix
OFELI::Reconstruction	add, 547
DP1toP1, 522	Axpy, 546
P0toP1, 522	get, 550
OFELI::Rectangle	getColHeight, 548
getSignedDistance, 524	Mult, 547
operator+=, 524	MultAdd, 547
Rectangle, 523	operator*=, 549
setBoundingBox, 524	operator(), 548
OFELI::Side	operator+=, 548
Add, 528	operator=, 548
Contains, 530	set, 546
DOF, 527	setLDLt, 549
EXTERNAL_BOUNDARY, 527	setMesh, 546
getCode, 529	setSkyline, 546
getMeasure, 530	SkSMatrix, 544, 545
getNeighborElement, 529	solve, 549, 550
Sett vergriboriblement, 327	50170,000

solveLDLt, 549	DSh, 576
TMult, 547	setLocal, 576
OFELI::Sphere	Sh, 575
getSignedDistance, 551	Triang6S, 575
operator+=, 551, 552	OFELI::Triangle
Sphere, 551	getSignedDistance, 579
OFELI::SteklovPoincare2DBE	operator+=, 579
run, 557	Triangle, 579
setExterior, 557	OFELI::Vect
setMesh, 557	add, 605, 606
SteklovPoincare2DBE, 556	Assembly, 602
OFELI::TINS2DT3S	Axpy, 604
setInput, 570	getAverage, 604
TINS2DT3S, 569, 570	getCurl, 603
OFELI::TINS3DT4S	0
	getDOFType, 594
setInput, 572	getDivergence, 604
TINS3DT4S, 571	getFFT, 610
OFELI::Tetra4	getGradient, 602, 603
CurlEdgeSh, 559	getInvFFT, 610
DSh, 560	getNorm1, 594
EdgeSh, 559	getNorm2, 595
Sh, 559	getNormMax, 595
OFELI::TimeStepping	getSCurl, 603
Assembly, 567	getWNorm2, 595
run, 567	insertBC, 600, 601
runOneTimeStep, 566	isGrid, 594
SAssembly, 567	MultAdd, 604
set, 563	Norm, 594
setConstantMatrix, 565	operator*=, 609
setInitial, 564	operator(), 606, 607
setInitialRHS, 565	operator+=, 608
setLinearSolver, 563, 565	operator,, 609
setNLTerm, 566	operator-=, 608, 609
setNLTerm0, 566	operator/=, 609
setNewmarkParameters, 565	operator=, 607, 608
setNonConstantMatrix, 565	push_back, 609
setPDE, 564	removeBC, 600
setRK3_TVDRHS, 564	resize, 593
setRK4RHS, 564	select, 591
setVerbose, 566	set, 591, 592, 604, 605
·	
TimeStepping, 563	setDOFType, 593
OFELI::Timer	setDG, 593
getTime, 561	setIJK, 595
Start, 561	setMesh, 592
Started, 561	setNodeBC, 595–598
Stop, 561	setSideBC, 596–599
OFELI::Triang3	setSize, 593
check, 574	setUniform, 608
DSh, 574	transferBC, 600
Grad, 574	Vect, 587–590
Sh, 573	WithMesh, 594
Triang3, 573	OFELI::WaterPorous2D
OFELI::Triang6S	BodyRHS, 612
atMidEdges, 576	BoundaryRHS, 612
-	•

setCoef, 612	CGS_SOLVER, 148
WaterPorous2D, 611	CONSISTENT_CAPACITY, 146
OFELI::triangle	CONSISTENT_MASS, 146
triangle, 578	CONTACT_BC, 149
OFELI_EPSMCH	CONTACT_DISTANCE, 147
Utilities, 40	CONTACT, 146
OFELI_GAUSS2	CONVECTION, 146
Utilities, 40	CRANK_NICOLSON, 147
OFELI_IMAG	check, 171
Utilities, 40	DENSE, 148
OFELI_ONEOVERPI	DEVIATORIC, 146
Utilities, 39	DIAG_PREC, 148
OFELI_PI	DIAGONAL, 148
Utilities, 39	DIFFUSION, 146
OFELLSIXTH	DILATATION, 146
Utilities, 39	DILU_PREC, 148
OFELI_SQRT2	DIRECT_SOLVER, 148
Utilities, 39	DISPLACEMENT_FIELD, 147
OFELL-SQRT3	Deactivate, 169, 170
Utilities, 39	DiagPrescribe, 157
OFELI_THIRD	EIGEN, 147
Utilities, 39	ELECTRIC, 146
OFELI_TOLERANCE	•
Utilities, 40	EqDataType, 146 FE_2D_3N, 148
OFELI_TWELVETH	FE_2D_3N, 148
	FE_2D_4N, 148
Utilities, 39 OFELLE	FE_3D_4N, 148
	FE_3D_8N, 148
Utilities, 39	•
OFEL 126, 172	FE_3D_AXI_3N, 148
OFELI, 126, 173 A, 149	FEType, 147 FLUX, 147
AB2, 147	
ADAMS_BASHFORTH, 147	FORWARD_EULER, 147 GAUSS_LEGENDRE, 149
•	•
AUX_INPUT_FIELD_1, 147	GMRES_SOLVER, 148
AUX_INPUT_FIELD_2, 147	get, 151, 152, 163
AUX_INPUT_FIELD_3, 147	getCode, 169
AUX_INPUT_FIELD_4, 147	getColInd, 161
add, 161	getDerivative, 153
Analysis, 147	getNy, 168
Axpy, 160, 161, 163, 164	getNz, 168
b, 149	getValue, 152, 153
BACKWARD_EULER, 147	Grid, 165, 166
BAND, 148	HEAT_FLUX, 146
BCType, 148	HEAT_SOURCE, 146
BDF2, 147	HEUN, 147
BICG_SOLVER, 148	IDENT_PREC, 148
BICG_STAB_SOLVER, 148	IDENTITY, 148
BODY_FORCE, 146	ILU_PREC, 148
BOUNDARY_CONDITION, 146	INITIAL AUX 2 146
BOUNDARY_FORCE, 147	INITIAL AUX 2, 146
BOUNDARY_TRACTION, 146	INITIAL_AUX_3, 146
BUOYANCY, 146	INITIAL_AUX_4, 146
CAPACITY, 146	INITIAL_FIELD, 146
CG_SOLVER, 148	INITIAL, 146

IOField, 149, 150	STIFFNESS, 146
IntegrationScheme, 149	SYMMETRIC, 148
isActive, 170	saveGMSH, 152
Iter, 171	set, 161
Iteration, 148	setCode, 168
LEAP_FROG, 147	setDomain, 167
LEFT_RECTANGLE, 149	setFile, 152
LOAD, 146	setGraph, 158
LORENTZ_FORCE, 146	setMesh, 157
LUMPED_CAPACITY, 146	setMeshFile, 151
LUMPED_MASS, 146	setSize, 158, 163
Laplace1D, 156, 163	
	setSolver, 162
Laplace2D, 156	setXMax, 167
MAGNETIC, 146	setXMin, 166
MASS, 146	setN, 168
MID_RECTANGLE, 149	solve, 161, 164, 165
MOBILITY, 146	SpMatrix, 154–156
MatrixType, 148	TEMPERATURE_FIELD, 147
Mult, 159	TMult, 160
MultAdd, 160	TRACTION, 147
NEWMARK, 147	TRANSIENT_ONE_STEP, 147
NONE, 147	TRANSIENT, 147
NORM1, 195	TRAPEZOIDAL, 149
NORM2, 195	TRIDIAGONAL, 148
NORM_MAX, 195	TimeScheme, 147
NormType, 195	TrMatrix, 163
OPTIMIZATION, 147	UNSYMMETRIC, 148
open, 151	VELOCITY_FIELD, 147
operator*, 159	VISCOSITY, 146
operator*=, 159, 164	WNORM1, 195
operator(), 158, 159, 164	WNORM2, 195
operator=, 161, 164	OPTIMIZATION
operator[], 159	OFELI, 147
PDE_Terms, 146	Objective
PERIODIC_A, 149	OFELI::MyOpt, 467
PERIODIC_B, 149	open
POINT_FORCE, 147	OFELI, 151
PRESSURE_FIELD, 147	operator!=
Preconditioner, 148	OFELI::Point, 509
Prev, 149	OFELI::Point2D, 512
put, 151	operator<
RIGHT_RECTANGLE, 149	Utilities, 50
RK3_TVD, 147	operator<<
RK4, 147	Finite Element Mesh, 91, 92, 106
RUNGE_KUTTA, 147	Solver, 125
SIMPSON, 149	Utilities, 41, 50, 54
SKYLINE, 148	Vector and Matrix, 75–77, 79, 80
SLIP, 149	operator>>
SOLUTION, 146	Vector and Matrix, 79
SOURCE, 147	operator*
SPARSE, 148	OFELI::LocalMatrix, 411
SSOR_PREC, 148	OFELI, 159
STATIONARY, 147	Utilities, 48, 52, 53
STEADY_STATE, 147	Vector and Matrix, 72–78, 80

operator*=	OFELI::Point, 509
OFELI::BMatrix, 215	OFELI::Vect, 609
OFELI::DMatrix, 252	operator-
OFELI::Domain, 256	Finite Element Mesh, 91
OFELI::LocalMatrix, 412	Utilities, 47, 51
OFELI::LocalVect, 419	Vector and Matrix, 74, 78
OFELI::Matrix, 430	operator-=
OFELI::Mesh, 441	OFELI::DMatrix, 252, 253
OFELI::Point, 508	OFELI::DSMatrix, 264
OFELI::Point2D, 512	OFELI::LocalMatrix, 411
OFELI::SkMatrix, 540	OFELI::LocalVect, 418
OFELI::SkSMatrix, 549	OFELI::Matrix, 430
OFELI::Vect, 609	OFELI::Point, 508
OFELI, 159, 164	OFELI::Point2D, 511, 512
operator()	OFELI::Vect, 608, 609
OFELI::BMatrix, 214	operator/
	*
OFELL:DMatrix, 249, 250	Utilities, 49, 53
OFELL::DSMatrix, 264	Vector and Matrix, 73, 74, 78
OFELI::Edge, 273	operator/=
OFELI::Element, 298, 300	OFELI::LocalMatrix, 412
OFELI::LocalMatrix, 410	OFELI::LocalVect, 419
OFELI::LocalVect, 418	OFELI::Point, 508
OFELI::Matrix, 429	OFELI::Point2D, 512
OFELI::Point, 508	OFELI::Vect, 609
OFELI::Point2D, 511	operator=
OFELI::Side, 529	OFELI::BMatrix, 215
OFELI::SkMatrix, 539	OFELI::DMatrix, 252
OFELI::SkSMatrix, 548	OFELI::Funct, 344
OFELI::Vect, 606, 607	OFELI::LocalMatrix, 411
OFELI, 158, 159, 164	OFELI::LocalVect, 418
operator+	OFELI::Matrix, 430
Utilities, 46, 47, 50, 51	OFELI::Point, 508
Vector and Matrix, 73, 74, 78	OFELI::Point2D, 511, 512
operator+=	OFELI::SkMatrix, 540
OFELI::BMatrix, 215	OFELI::SkSMatrix, 548
OFELI::Brick, 218	OFELI::Vect, 607, 608
OFELI::Circle, 219	OFELI, 161, 164
OFELI::DMatrix, 252	operator==
OFELI::DSMatrix, 264	Finite Element Mesh, 96, 97
OFELI::Ellipse, 304, 305	OFELI::Point, 508
OFELI::LocalMatrix, 411	OFELI::Point2D, 512
OFELI::LocalVect, 418	Utilities, 46, 50
OFELI::Matrix, 430	operator&&
OFELI::Point, 508	Finite Element Mesh, 91
OFELI::Point2D, 511, 512	operator[]
OFELI::Polygon, 514	OFELI::LocalVect, 418
OFELI::Rectangle, 524	OFELI::Matrix, 430
	OFELI::Point, 508
OFELL:SkMatrix, 540	
OFELL:SkSMatrix, 548	OFELL 150
OFELL::Sphere, 551, 552	OFELI, 159
OFELL: Triangle, 579	operator
OFELI::Vect, 608	Finite Element Mesh, 91
operator,	OptMethod
OFELI::LocalVect, 419	OFELI::OptSolver, 494

OptSolver, 492	Prev
OFELI::OptSolver, 494	OFELI, 149
	push_back
P0toP1	OFELI::Vect, 609
OFELI::Reconstruction, 522	put
PARSE	OFELI::Mesh, 449
Utilities, 40	OFELI::Partition, 503
PDE_Terms	OFELI, 151
OFELI, 146	_
PERIODIC_A	qksort
OFELI, 149	Utilities, 65, 66
PERIODIC_B	Quad4, 519
OFELI, 149	OFELI::Quad4, 520
POINT_FORCE	QuickSort
OFELI, 147	Utilities, 65
PRESSURE_FIELD	RIGHT_RECTANGLE
OFELI, 147	
Partition, 499	OFELI, 149 RK3_TVD
OFELI::Partition, 501	
Penta6, 503	OFELI, 147 RK4
OFELI::Penta6, 504	
Periodic	OFELI, 147 ROE_SOLVER
OFELL::DC2DT3, 227	OFELI::Muscl, 459
OFELL::DC3DT4, 239	RUNGE_KUTTA
OFELL::Elas2DT3, 288	OFELI, 147
OFELI::NSP2DQ41, 480	RUSANOV_SOLVER
Physical properties of modic 81	OFELI::Muscl, 459
Physical properties of media, 81 PlaneStrain	Reaction
	OFELI::Elas2DT3, 287
OFELI::Elas2DQ4, 282 PlaneStress	Real
	Vector and Matrix, 79
OFELI::Elas2DQ4, 282 Point	Reconstruction, 521
OFELI::Point, 507	Rectangle, 523
Point< T <sub>-</sub> >, 506	OFELI::Rectangle, 523
Point2D< T_>,500	Refine
Point2D	OFELI::Mesh, 451
OFELI::Point2D, 511	removeBC
Polygon, 512	OFELI::Vect, 600
OFELI::Polygon, 513	RenumberEdge
Porous Media problems, 28	OFELI::Mesh, 447
Post	RenumberElement
OFELI::Laplace2DT3, 385	OFELI::Mesh, 447
Prec	RenumberNode
OFELI::Prec, 515, 516	OFELI::Mesh, 446
Prec < T > 514	RenumberSide
Preconditioner	OFELI::Mesh, 447
OFELI, 148	Reorder
Prescribe	OFELI::Mesh, 445
OFELI::Matrix, 426, 427	Replace
PrescribeSide	OFELI::Element, 297
OFELI::Matrix, 427	Rescale
Prescription, 518	OFELI::Mesh, 448
OFELI::Prescription, 518	reset
1 ,	

OFELI::DMatrix, 246	OFELI::Muscl, 459
OFELI::Matrix, 425	SYMMETRIC
resize	OFELI, 148
OFELI::Vect, 593	save
Richardson	OFELI::Mesh, 450
Solver, 122	saveBamg
run	Utilities, 64
OFELI::BiotSavart, 211	saveField
OFELI::EigenProblemSolver, 278	Utilities, 41, 42
OFELI::Equa_Porous, 312	saveGMSH
OFELI::Integration, 366	OFELI, 152
OFELI::LaplaceDG2DP1, 389	saveGmsh
OFELI::MeshAdapt, 456	Utilities, 45, 46, 61
OFELI::ODESolver, 491	saveGnuplot
OFELI::OptSolver, 498	Utilities, 43, 62
OFELI::SteklovPoincare2DBE, 557	saveMatlab
OFELI::TimeStepping, 567	Utilities, 62
runOneTimeStep	saveMbb
OFELI::LCL1D, 391	OFELI::MeshAdapt, 455
OFELI::LCL2DT, 394	saveMesh
OFELI::NSP2DQ41, 481	Utilities, 61
OFELI::ODESolver, 491	saveTecplot
OFELI::TimeStepping, 566	Utilities, 44, 63
runSubSpace	saveVTK
OFELI::EigenProblemSolver, 279	Utilities, 44, 45, 63
Ŭ	Scale
SAssembly	Utilities, 66
OFELI::EigenProblemSolver, 279	Schur
OFELI::TimeStepping, 567	Solver, 123
SIMPSON	seODEVectors
OFELI, 149	OFELI::ODESolver, 488, 489
SIMULATED_ANNEALING	select
OFELI::OptSolver, 494	OFELI::Vect, 591
SKYLINE	selectCode
OFELI, 148	OFELI::EdgeList, 274
SLIP	OFELI::NodeList, 478
OFELI, 149	OFELI::SideList, 532
SOLUTION	selectCoordinate
OFELI, 146	OFELI::NodeList, 478
SOURCE	selectLevel
OFELI, 147	OFELI::ElementList, 303
SPARSE	set
OFELI, 148	OFELI::Beam3DL2, 205
SSOR_PREC	OFELI::DMatrix, 246
OFELI, 148	OFELI::DSMatrix, 261
SSOR	OFELI::Element, 298
Solver, 124	OFELI::LaplaceDG2DP1, 388, 389
STATIONARY	OFELI::LinearSolver, 406
OFELI, 147	OFELI::Material, 421
STEADY STATE	OFELI::Matrix, 428
OFELI, 147	OFELI::Mesh, 448, 449
STIFFNESS	
	OFFI In ODFSalvar 181
OFFI I 146	OFELI::ODESolver, 484
OFELI, 146 SUPERBEE_LIMITER	OFELI::ODESolver, 484 OFELI::Side, 528 OFELI::SkMatrix, 537

OFELI:SkSMatrix, 546 OFELI:SkKMatrix, 536 OFELI:SkEZDIT3, 268 OFELI:SCPCIDT), 359, 360 OFELI:CPCIDT), 359, 360 OFELI:CCIDT, 394 OFELI:CLCIDT, 397 OFELI:CLCIDT, 397 OFELI:BiotSavart, 210 setBoundary Offeli:BiotSavart, 210 setBoundaryNodeCodes Finite Element Mesh, 103 setBoundarySideCodes Finite Element Mesh, 104 setBoundingBox OFELI:Brick, 217 OFELI:Brick, 217 OFELI:Brement, 302 OFELI:Element, 302 OFELI:Element, 302 OFELI:Element, 302 OFELI:Element, 305 OFELI:Element, 306 OFELI:Element, 306 OFELI:Blement, 306 OFELI:Blement, 306 OFELI:Blement, 306 OFELI:Blement, 306 OFELI:DSMatrix, 246 OFELI:DSMatrix, 246 OFELI:DDESolver, 484, 485 OFELI:DDESolver, 480 OFELI:DDESolver, 480 OFELI:DDESolver, 490 OF		
OFELL:Vect, 591, 592, 604, 605 OFELL:COPELD, 359, 350 OFELL:CPC2DT, 359, 360 OFELL:CL2DT, 394 OFELL:CL2DT, 394 OFELL:CL2DT, 397 OFELL:BiotSavart, 210 SetBoundaryCondition OFELL:BibicSavart, 210 SetBoundaryNodeCodes Finite Element Mesh, 103 SetBoundaryNodeCodes Finite Element Mesh, 103 SetBoundarySideCodes Finite Element Mesh, 104 SetBoundingBox OFELL:Brick, 217 OFELL:Brick, 217 OFELL:Brick, 217 OFELL:Brick, 217 OFELL:Bick, 217 OFELL:Brick, 217 OFELL:Drick, 259 SetFile OFELL:Drick, 259 SetFile OFELL:Drick, 259 SetFore OFELL:Brick, 217 OFELL:Crick, 593 SetDors OFELL:Brick, 217 OFELL:Drick, 222	OFELI::SkSMatrix, 546	OFELI::Side, 528
OFELL:Vect, 591, 592, 604, 605 OFELL:CPCDTD, 359, 355 OFELL:ICPCZDTD, 359, 360 OFELL:CLC1DT, 394 OFELL:CLC1DT, 397 OFELL:CLC1DT, 397 OFELL:BiotSavart, 210 setBoundaryCondition OFELL: Laplacel DL2, 380 setBoundaryNodeCodes Finite Element Mesh, 103 setBoundaryNodeCodes Finite Element Mesh, 104 setBoundingBox OFELL:Brick, 217 OFELL:Drick, 217 OFEL	OFELI::TimeStepping, 563	OFELI::SkMatrix, 536
OFELL;161 setBC OFELL:ICPG1D, 354, 355 OFELL:ICPG2DT, 359, 360 OFELL:ICLDD, 391, 392 OFELL:ICLDDT, 394 OFELL:CDPSolver, 495 setBoundary OFELL:BictSovart, 210 setBoundaryCondition OFELL:BictSovart, 210 setBoundarySideCodes Finite Element Mesh, 103 setBoundarySideCodes Finite Element Mesh, 104 setBoundingBox OFELL:Brick, 217 OFELL:Brick, 217 OFELL:Brick, 217 OFELL:Bittside, 258 OFELL:Side, 530 setCode OFELL:Blement, 296, 298 OFELL:Side, 528 OFELL:SuderPorous2D, 612 setCoulmn OFELL:DDBSolver, 484, 485 OFELL:Bobatrix, 263 setCode OFELL:DDBSolver, 484, 485 OFELL:DDBSolver, 490 OFELL:Bittside, 268 OFELL:Bobatrix, 263 setCorsantMatrix OFELL:Bittside, 475 setCurrentDensity OFELL:Bittsek, 442 setDoFType OFELL:Bittsek, 273 OFELL:Bittsek, 289 OFELL:Bittside, 289 OFELL:Bittside, 289 OFELL:Bittside, 289 OFELL:Bittside, 289 OFELL:CDDESolver, 480, 485 OFELL:Bittside, 258 OFELL:Bobatrix, 263 setConstantMatrix OFELL:Bobatrix, 263 setCorsantMatrix OFELL:Bittside, 475 setCurrentDensity OFELL:Bittsidesvart, 209 setDOFSupport OFELL:Bittsek, 442 setDOFType OFELL:Bittsek, 273 OFELL:CDESolver, 487, 488 OFELL:Timestepping, 565 setDof OFELL:CDESolver, 487, 488 OFELL:Timestepping, 565 setInut:DOESolver, 487, 488 OFELL:DDESolver, 487, 488 OFELL:Timestepping, 565		setData
setBC OFELL::ICPG1D, 354, 355 OFELL::ICPG2DT, 359, 360 OFELL::LCL1DT, 391, 392 OFELL::LCL2DT, 394 OFELL::CL1DT, 397 OFELL::Cl2DT, 397 OFELL::DISOlver, 495 setBoundary OFELL::BiotSavart, 210 setBoundaryCondition OFELL::ApplaceIDL2, 380 setBoundaryNodeCodes Finite Element Mesh, 103 setBoundarySideCodes Finite Element Mesh, 104 setBoundingBox OFELL::Brick, 217 OFELL::Brick, 217 OFELL::Retangle, 524 setCrl OFELL::Muscl, 459 setChild OFELL::Ielement, 302 OFELL::Blement, 302 OFELL::Blement, 296, 298 OFELL::Blement, 296, 298 OFELL::Node, 475 OFELL::DESolver, 484, 485 OFELL::DDESolver, 484, 485 OFELL::DDESolver, 484, 485 OFELL::DDESolver, 490 OFELL::DDESolver, 490 OFELL::DDESolver, 490 OFELL::DTiobSupport OFELL::Mode, 475 setCurrentDensity OFELL::Mode, 475 setDorFype OFELL::Mode, 475 setDorFype OFELL::Mode, 475 OFELL::BiotSavart, 209 setDOFSupport OFELL::Mode, 475 OFELL::BiotSavart, 209 setDOFType OFELL::Mesh, 442 SetDOFType OFELL::Clage, 273 OFELL::Immestepping, 565 setLoor OFELL::Mesh, 442 SetDOFType OFELL::Desolver, 487, 488 OFELL::Immestepping, 565 setLoop OFELL::Desolver, 487, 488 OFELL::DDESolver, 487, 488 OFELL::Immestepping, 565 setLoop OFELL::DESolver, 487, 488 OFELL::DESolver, 487, 488 OFELL::DESolver, 487, 488 OFELL::DDESolver, 487, 488 OFELL::DESOlver, 487, 488 OFELL:		OFELI::EC2D1T3, 268
OFELI::ICPG1D, 354, 355 OFELI::ICPG2DT, 359, 360 OFELI::LCLID, 391, 392 OFELI::LCLID, 391, 392 OFELI::LCLIDT, 394 OFELI::LCJDT, 394 OFELI::CptSolver, 495 setBoundary OFELI::BiotSavart, 210 setBoundaryCondition OFELI::Baplace1DL2, 380 setBoundarySideCodes Finite Element Mesh, 103 setBoundarySideCodes Finite Element Mesh, 104 setBoundingBox OFELI::Brick, 217 OFELI::Brick, 217 OFELI::Brenet, 302 OFELI::Blement, 303 SetCode OFELI::DoESolver, 496 OFELI::DDESolver, 484, 485 OFELI::DDESolver, 484, 485 OFELI::DDESolver, 490 OFELI::DDESolver, 490 OFELI::DDESolver, 490 OFELI::TimeStepping, 565 setCord OFELI::Node, 475 setCurrentDensity OFELI::Mode, 475 setCurrentDensity OFELI::Mode, 475 setCurrentDensity OFELI::Mode, 475 setCurrentDensity OFELI::Mode, 475 setDOFFype OFELI::Mode, 33 setDOF OFELI::Mode, 33 setDOF OFELI::Mode, 347 OFELI::DESolver, 490 OFELI::DESolver, 490 OFELI::ImeStepping, 564 setInitialCondition OFELI::CLIDT, 394 OFELI::CLIDT, 394 OFELI::CLIDT, 394 OFELI::CLIDT, 394 OFELI::DESolver, 487, 488 OFELI::DES		•
OFELL::LCL2DT, 359, 360 OFELL::LCL3DT, 394 OFELL::LCL3DT, 397 OFELL::DqtSolver, 495 setBoundary OFELL::Biot5avart, 210 setBoundaryNodeCodes Finite Element Mesh, 103 setBoundaryNodeCodes Finite Element Mesh, 104 setBoundarySideCodes Finite Element Mesh, 104 setBoundarySideCodes Finite Element Mesh, 104 setBoundarySideCodes Finite Element Mesh, 104 setBoundirySideCodes Finite Element Mesh, 104 setEquation OFELL::Mesh, 440 setEquation OFELL::Mesh, 440 setEquation OFELL::MyOpt, 467 setExerior OFELL::SteklovPoincare2DBE, 557 setExerior OFELL::MyOpt, 467 setExerior OFELL::Node, 475 OFELL::Side, 528 OFELL::Node, 475 OFELL::DDESolver, 484, 485 OFELL::DDESolver, 484, 485 OFELL::DDESolver, 480 OFELL::DDESolver, 490 OFELL::DDESolver, 490 OFELL::DDESolver, 490 OFELL::DDESolver, 490 OFELL::DDESolver, 490 OFELL::DDESolver, 490 OFELL::DOESolver, 490 OFELL::CL2DT, 394 OFELL::CL2DT, 394 OFELL::CL2DT, 394 OFELL::DCESOlver, 487, 488 OFELL::Dement, 298 OFELL::Descolver, 487, 488 OFELL::Doesolver, 487, 488 OFELL::Dement, 298 OFELL::Descolver, 487, 488 OFELL::Dement, 298 OFELL::Dement, 298 OFELL::Descolver, 487, 488 OFELL::Doesolver, 487, 488 OFELL::Doesolver, 487, 488 OFELL::Dement, 298 OFELL::Descolver, 487, 488 OFELL::Dement, 298 OFELL::DelciDescolver, 487, 488 OFELL::Dement, 298 OFELL::DelciDescolver, 487, 488 OFELL::Descolver		
OFELL::LCL1D, 391, 392 OFELL::LCL2DT, 394 OFELL::CL2DT, 397 OFELL::Cl3DT, 397 OFELL::OptSolver, 495 setBoundary OFELL::BiotSavart, 210 setBoundaryCondition OFELL::Baplace1DL2, 380 setBoundaryNodeCodes Finite Element Mesh, 103 setBoundarySideCodes Finite Element Mesh, 104 setBoundingBox OFELL::Brick, 217 OFELL::Blement, 302 OFELL::Blement, 302 OFELL::Blement, 302 OFELL::Blement, 296, 298 OFELL::Clament, 296, 298 OFELL::Clament, 296, 298 OFELL::DSMatrix, 246 OFELL::WaterPorous2D, 612 setCode OFELL::DSMatrix, 246 OFELL::DSMatrix, 246 OFELL::DSMatrix, 246 OFELL::DSMatrix, 263 setCode OFELL::DSMatrix, 246 OFELL::DEsolver, 490 OFELL::DEsolv		•
OFELI::LCL2DT, 394 OFELI::CQtSolver, 495 setBoundary OFELI::Disortsolver, 495 setBoundary OFELI::BiotSavart, 210 setBoundaryCondition OFELI::Laplace1DL2, 380 setBoundaryNodeCodes Finite Element Mesh, 103 setBoundarySideCodes Finite Element Mesh, 104 setBoundingBox OFELI::Muscl, 217 OFELI::Rectangle, 524 setCrl OFELI::Rectangle, 524 setChild OFELI::Biotsavart, 209 SetCode OFELI::Side, 530 OFELI::Side, 530 SetCode OFELI::Side, 530 OFELI::Side, 530 SetCode OFELI::Side, 530 OFELI::Side, 530 OFELI::DosMatrix, 246 OFELI::DosMatrix, 245 OFELI::DosMatrix, 245 OFELI::DosMatrix, 245 SetDosmain OFELI::Math, 440 SetDosmain OFELI::Mote, 440 SetElementCodes Finite Element Mesh, 104 SetElementCodes Finite Ilement Mesh, 104 SetElementCodes Fini		
OFELL::Ct3DT, 397 OFELL::OptSolver, 495 setBoundary OFELL::BiotSavart, 210 setBoundaryCondition OFELL::Laplace1DL2, 380 setBoundaryNodeCodes Finite Element Mesh, 103 setBoundarySideCodes Finite Element Mesh, 104 setBoundingBox OFELL::Brick, 217 OFELL::Brick, 217 OFELL::Brick, 217 OFELL::Brick, 217 OFELL::Brick, 217 OFELL::Busel, 459 setChild OFELL::Element, 302 OFELL::Element, 302 OFELL::Blement, 302 OFELL::Blement, 296, 298 OFELL::Blement, 296, 298 OFELL::Node, 475 OFELL::WaterPorous2D, 612 setColumn OFELL::DSSolver, 484, 485 OFELL::DSSolver, 490 OFELL::DSSolver, 490 OFELL::Diesolver, 490 OFELL::BiotSavart, 209 setDOFSupport OFELL::BiotSavart, 209 setDOFSupport OFELL::Bicdee, 273 OFELL::Bicdsevart, 209 setDOFSupport OFELL::Bicdee, 273 OFELL::Bicdee, 273 OFELL::Cdge, 273 OFELL::Cdge, 273 OFELL::DopeSolver, 480 OFELL::DopeSolver, 490 OFELL::Diesolver, 490 OFELL::Diesolver, 490 OFELL::BiotSavart, 209 setDOFSupport OFELL::BiotSavart, 209 setDOFSupport OFELL::Bicdee, 273 OFELL::Cdge, 273 OFELL::ClaDT, 394 OFELL::ClaDT, 397 setInjut OFELL::DCIDL2, 222		·
oFELL::OptSolver, 495 setBoundary OFELL::BiotSavart, 210 setBoundaryCondition OFELL::LaplaceIDL2, 380 setBoundaryNodeCodes Finite Element Mesh, 103 setBoundarySideCodes Finite Element Mesh, 104 setBoundingBox OFELL::Brick, 217 OFELL::Brick, 217 OFELL::Rectangle, 524 setCrL OFELL::Blement, 302 OFELL::Side, 530 setCode OFELL::Blement, 296, 298 OFELL::Side, 530 SetCode OFELL::Side, 530 SetCode OFELL::Blement, 296, 298 OFELL::Node, 475 OFELL::Side, 528 OFELL::Node, 475 OFELL::DDESolver, 484, 485 OFELL::DDSAlartix, 246 OFELL::DSMatrix, 246 OFELL::DSMatrix, 246 OFELL::DSMatrix, 263 setConstantMatrix OFELL::DMatrix, 246 OFELL::DSMatrix, 263 setCord OFELL::DMatrix, 246 OFELL::DSMatrix, 263 setCord OFELL::DMatrix, 246 OFELL::DSSolver, 490 OFELL::DSSolver, 490 OFELL::DSSolver, 490 OFELL::TimeStepping, 565 setCord OFELL::Node, 475 setCurrentDensity OFELL::Moscolver, 490 OFELL::Node, 475 setCurrentDensity OFELL::Moscolver, 490 OFELL::DSSolver, 490 OFEL::DSSolver, 490 OFELL::DSSolver, 490 OFELL::DSSolver, 490 OFEL::DSSolver, 490 OFEL::DSSolver, 490 OFEL::DSSolver, 490 OFEL::DS		
setBoundary OFELL::BiotSavart, 210 setBoundaryCondition OFELL::Laplace1DL2, 380 setBoundaryNodeCodes Finite Element Mesh, 103 setBoundarySideCodes Finite Element Mesh, 104 setBoundingBox OFELL::Retangle, 524 setCFL OFELL::Retangle, 524 setClid OFELL::Biement, 302 OFELL::Biement, 302 OFELL::Bement, 302 OFELL::Bement, 302 OFELL::Bement, 296, 298 OFELL::Blement, 296, 298 OFELL::Side, 530 OFELL::Side, 530 SetCode OFELL::Side, 528 OFELL::MaterPorous 2D, 612 setCoum OFELL::DMatrix, 246 OFELL::DSMatrix, 263 setComm OFELL::DSMatrix, 263 setCond OFELL::DSMatrix, 264 OFELL::DSMatrix, 263 setCond OFELL::DSMatrix, 264 OFELL::DSMatrix, 263 setCond OFELL::DSMatrix, 264 OFELL::DSMatrix, 265 setCond OFELL::DSMatrix, 266 OFELL::DSMatrix, 267 OFELL::DSMatrix, 268 oFELL::DCSDSolver, 486, 487 OFELL::DIESIOTSAVART, 209 setDOFSupport OFELL::RibiotSavart, 209 setDOFSupport OFELL::Club, 391 OFELL::Club, 391 OFELL::Club, 391 OFELL::Club, 391 OFELL::Club, 391 OFELL::Club, 391 OFELL::DCSDSolver, 487, 488 OFELL::DCSDSolver, 487, 488 OFELL::DCSDSolver, 487, 488 OFELL::DCIDL2, 222		·
oFELL::BiotSavart, 210 setBoundaryCondition OFELL::Laplace1DL2, 380 setBoundaryNodeCodes Finite Element Mesh, 103 setBoundarySideCodes Finite Element Mesh, 104 setBoundarySideCodes Finite Element Mesh, 104 setBoundarySideCodes OFELL::Brick, 217 OFELL::Rectangle, 524 setCFL OFELL::Rectangle, 524 setCfL OFELL::Muscl, 459 setCided OFELL::Belment, 302 OFELL::Side, 530 setCode OFELL::Belment, 296, 298 OFELL::Node, 475 OFELL::Side, 528 OFELL::OPESolver, 484, 485 OFELL::DMatrix, 246 OFELL::DMatrix, 246 OFELL::DSMatrix, 263 setCostantMatrix OFELL::DSMatrix, 263 setCord OFELL::DSMatrix, 263 setCostantMatrix OFELL::DSMatrix, 263 setCord OFELL::DSMatrix, 263 setCord OFELL::DSMatrix, 263 setCostantMatrix OFELL::DSMatrix, 263 setCord OFELL::DSMatrix, 265 setCord OFELL::DSMatrix, 267 OFELL::DSMatrix, 268 SetCord OFELL::DSMatrix, 269 OFELL::DDSOlver, 490 OFELL::DDSOlver, 4		
setBoundaryCondition OFELL::Laplace1DL2, 380 SetBoundaryNodeCodes Finite Element Mesh, 103 SetBoundarySideCodes Finite Element Mesh, 104 SetBoundingBox OFELL::Rectangle, 524 SetClL::Rectangle, 524 SetChild OFELL::Muscl, 459 SetCode OFELL::Blement, 302 OFELL::Blement, 302 OFELL::Blement, 296, 298 OFELL::Blement, 296, 298 OFELL::Node, 475 OFELL::MaterPorous2D, 612 SetColum OFELL::DMatrix, 246 OFELL::DMatrix, 246 OFELL::DSMatrix, 263 SetCoord OFELL::DMatrix, 263 SetCoord OFELL::Node, 475 SetCli:DMatrix, 263 SetCoord OFELL::DSMatrix, 263 SetCoord OFELL::Node, 475 OFELL::DSSolver, 490 OFELL::DSSolver, 490 OFELL::DSSolver, 490 OFELL::Node, 475 SetCoord OFELL::Node, 475 SetCli::DDSSolver, 490 OFELL::DSSolver, 490 OFELL::DSSolver, 490 OFELL::DSSolver, 490 OFELL::Node, 475 SetCoord OFELL::Node, 475 SetCli::Node, 475 SetCli::Node, 475 SetCli::DSSolver, 490 OFELL::DSSolver, 490 OFELL::DSSolver, 490 OFELL::DSSolver, 490 OFELL::DSSolver, 490 OFELL::Node, 475 SetCloord OFELL::Node, 475 SetCli::Node, 475 SetCli::Node, 475 SetCli::Node, 475 SetCli::Node, 475 SetCli::Node, 475 SetCli::DSSolver, 490 OFELL::DSSolver, 490 OFELL::DSSolver, 490 OFELL::DSSolver, 490 OFELL::Node, 475 SetCli::Node, 475 Set	setBoundary	OFELI::DMatrix, 244
oFELI::Laplace1DL2, 380 setBoundaryNodeCodes Finite Element Mesh, 103 setBoundarySideCodes Finite Element Mesh, 104 setBoundarySideCodes Finite Element Mesh, 104 setBoundarySideCodes Finite Element Mesh, 104 setBoundingBox OFELI::Brick, 217 OFELI::Brick, 217 OFELI::Rectangle, 524 setCFL OFELI::Muscl, 459 setChild OFELI::Element, 302 OFELI::Belement, 302 OFELI::Belement, 302 OFELI::Belement, 302 OFELI::Belement, 302 OFELI::Belement, 296, 298 OFELI::Node, 475 OFELI::Node, 475 OFELI::Node, 475 OFELI::DOESolver, 484, 485 OFELI::DDESolver, 484, 485 OFELI::DMatrix, 246 OFELI::DSMatrix, 263 setConstantMatrix OFELI::DSSatrix, 263 setCord OFELI::DSSatrix, 263 setCord OFELI::Node, 475 setCord OFELI::DSSolver, 490 OFELI::DSSolver, 490 OFELI::DSSolver, 490 OFELI::DDESolver, 490 OFELI::DDESolver, 490 OFELI::DDESolver, 490 OFELI::DSSolver, 490 OFELI::DDESolver, 490 OFELI::DOESolver, 490 OFELI::CDESolver, 490 OFELI::DOESolver, 490 OFELI::DOESolver, 490 OFELI::DOESolver, 490 OFELI::DOESolver, 490 OFELI::CLID, 391 OFELI::DOESolver, 490 OFELI::CLID, 391 OFELI::CLID, 391 OFELI::CLID, 391 OFELI::CLID, 391 OFELI::CDESolver, 487, 488 OFELI::DOESolver, 487, 488 OFELI::DOE	OFELI::BiotSavart, 210	OFELI::DSMatrix, 263
setBoundaryNodeCodes Finite Element Mesh, 103 setBoundarySideCodes Finite Element Mesh, 104 setBoundingBox OFELI::Brick, 217 OFELI::Brick, 217 OFELI::Rectangle, 524 setCFL OFELI::Muscl, 459 setChild OFELI::Side, 530 setCode OFELI::Blement, 302 OFELI::Blement, 296, 298 OFELI::Node, 475 OFELI::OPESolver, 484, 485 OFELI::ODESolver, 484, 485 OFELI::DMatrix, 246 OFELI::DMatrix, 246 OFELI::DMatrix, 246 OFELI::DSMatrix, 263 setCourd OFELI::DMatrix, 246 OFELI::DSMatrix, 263 setCord OFELI::Node, 475 setCourn OFELI::DSSolver, 490 OFELI::Node, 475 setCourn OFELI::Node, 475 setCourn OFELI::Node, 475 setCourn OFELI::DESolver, 490 OFELI::DDESolver, 490 OFELI::Node, 475 setCourn OFELI::Node, 475 setC	setBoundaryCondition	setDiagonal
setBoundaryNodeCodes Finite Element Mesh, 103 setBoundarySideCodes Finite Element Mesh, 104 setBoundingBox OFELI::Brick, 217 OFELI::Brick, 217 OFELI::Rectangle, 524 setCFL OFELI::Muscl, 459 setChild OFELI::Side, 530 setCode OFELI::Blement, 302 OFELI::Blement, 296, 298 OFELI::Node, 475 OFELI::OPESolver, 484, 485 OFELI::ODESolver, 484, 485 OFELI::DMatrix, 246 OFELI::DMatrix, 246 OFELI::DMatrix, 246 OFELI::DSMatrix, 263 setCourd OFELI::DMatrix, 246 OFELI::DSMatrix, 263 setCord OFELI::Node, 475 setCourn OFELI::DSSolver, 490 OFELI::Node, 475 setCourn OFELI::Node, 475 setCourn OFELI::Node, 475 setCourn OFELI::DESolver, 490 OFELI::DDESolver, 490 OFELI::Node, 475 setCourn OFELI::Node, 475 setC	OFELI::Laplace1DL2, 380	OFELI::Matrix, 425
Finite Élement Mesh, 103 setBoundarySideCodes Finite Element Mesh, 104 setBoundingBox OFELI::Brick, 217 OFELI::Brick, 217 OFELI::Rectangle, 524 setCFL OFELI::Muscl, 459 setChild OFELI::Blement, 302 OFELI::Bement, 302 OFELI::Bement, 302 OFELI::Bement, 296, 298 OFELI::Node, 475 OFELI::Node, 475 SetCode OFELI::ODESolver, 484, 485 OFELI::DSMatrix, 246 OFELI::DSMatrix, 246 OFELI::DSSMatrix, 263 setCoord OFELI::DSSMatrix, 263 setCoord OFELI::Node, 475 setInitialCondition OFELI::DSDisolver, 486, 487 OFELI::DIsolver, 486, 487 OFELI::DIsolver, 486, 487 OFELI::DIsolver, 486, 487 OFELI::LCLID, 391 oFELI::LCLID, 391 setDOFTupport OFELI::LCLID, 391 OFELI::LCLID, 391 OFELI::LCLID, 391 OFELI::LCLID, 397 setDOFTupe OFELI::DESolver, 487, 488 OFELI::DIsolver, 487, 488 OFELI::DIso		
setBoundarySideCodes Finite Element Mesh, 104 setBoundingBox OFELL:Brick, 217 OFELI::Brick, 217 OFELI::Rectangle, 524 setCFL OFELI::Muscl, 459 setChild OFELI::Belment, 302 OFELI::Belment, 302 OFELI::Belment, 302 OFELI::Belment, 296, 298 OFELI::Belment, 296, 298 OFELI::Side, 528 OFELI::NasSolver, 470 OFELI::NasSolver, 470 OFELI::DESolver, 484, 485 OFELI::DESolver, 484, 485 OFELI::DESolver, 484, 485 OFELI::DESolver, 484, 485 OFELI::DSMatrix, 246 OFELI::DSMatrix, 246 OFELI::DESolver, 490 OFELI::Node, 475 setCoord OFELI::Node, 475 setCoord OFELI::Node, 475 setCoord OFELI::DESolver, 490 OFELI::DESolver, 490 OFELI::Node, 475 setCurrentDensity OFELI::Node, 475 setCurrentDensity OFELI::BiotSavart, 209 setDOFSupport OFELI::BiotSavart, 209 setDOFSupport OFELI::Mesh, 442 setDOFType OFELI::Edge, 273 OFELI::Edge, 273 OFELI::Edge, 273 OFELI::Edge, 273 OFELI::Element, 298 OFELI::DC1DL2, 222		OFELI::Mesh. 440
Finite Élement Mesh, 104 setBoundingBox OFELL:Brick, 217 OFELL:Brick, 217 OFELL:Rectangle, 524 setCFL OFELL:Muscl, 459 setChild OFELL:Element, 302 OFELL:Side, 530 SetCode OFELL:Brick, 273 OFELL:Brick, 273 OFELL:Brick, 275 OFELL:Side, 528 OFELL:Side, 528 OFELL:ODESolver, 484, 485 OFELL:DDESolver, 484, 485 OFELL:DDESolver, 490 OFELL:DDESolver, 490 OFELL:TimeStepping, 565 setCourrentDensity OFELL:Mode, 475 SetCourrentDensity OFELL:Mode, 475 SetCourrentDensity OFELL:Mesh, 442 SetDOFType OFELL:Mesh, 442 SetDOFT OFELL:Element, 209 OFELL:DDESolver, 490 OFELL:DFSolver, 490 OFELL:DFSolver, 490 OFELL:DESORY OFELL:Mesh, 442 SetDOFType OFELL:Mesh, 442 SetDOFType OFELL:Element, 298 OFELL:DOESOlver, 487, 488 OFELL:Brick, 425 OFELL:Brick, 426 OFELL:Brick, 426 OFELL:Brick, 426 OFELL:Brick, 426 OFELL:CDESOlver, 490 OFELL:CDESOlver, 490 OFELL:CLIDT, 356 SetConstantMatrix OFELL:DESOLVER, 490 OFELL:CLIDT, 391 OFELL:CLIDT, 391 SetDOFType OFELL:Brick, 593 SetDOFType OFELL:Element, 298 OFELL:DCIDL2, 222	·	
setBoundingBox OFELI:Brick, 217 OFELI:Brick, 217 OFELI:Rectangle, 524 setCFL OFELI:Muscl, 459 setChild OFELI:SteklovPoincare2DBE, 557 setChild OFELI:Side, 530 SetCode OFELI:Bilement, 296, 298 OFELI:Side, 528 OFELI:Side, 528 OFELI:Side, 58 OFELI:WaterPorous2D, 612 setColumn OFELI:DMatrix, 246 OFELI:DSMatrix, 263 setConstantMatrix OFELI:DESolver, 490 OFELI:TimeStepping, 565 setCord OFELI:Bilesidsavart, 209 setDoFSupport OFELI:Bilesh, 442 setDOFType OFELI:West, 593 setDOF OFELI:West, 593 SetDOF OFELI:Edge, 273 OFELI:IElement, 298 SetElement Codes Finite Element Mesh, 104 setEquation OFELI:MyOpt, 467 setELI:SteklovPoincare2DBE, 557 setELI:SteklovPoincare2DBE, 557 setELI:Itimegration, 365 OFELI, 152 setFile OFELI:Integration, 365 SetGraph OFELI:Integration, 365 SetGraph OFELI:Integration, 365 SetGraph OFELI:Integration, 365 SetGraph OFELI:Integration, 365 SetIntial OFELI:Integration, 365 OFELI:Integration, 365 SetIntegration OFELI:Integration, 365 SetIntegration OFELI:Integration, 365 OFELI:Integratio		
OFELL:Brick, 217 OFELI:Rectangle, 524 setCFL OFELI:Muscl, 459 setChild OFELI:Side, 530 setCode OFELI:Blement, 296, 298 OFELI:Side, 528 OFELI:Side, 528 OFELI:Side, 528 OFELI:OPESolver, 484, 485 OFELI:DMatrix, 246 OFELI:DMatrix, 246 OFELI:DSMatrix, 263 setConstantMatrix OFELI:DEsolver, 490 OFELI:TimeStepping, 565 setCourrentDensity OFELI:BioSsavart, 209 setDOFSupport OFELI:BioSsavart, 209 setDOFSupport OFELI:Mesh, 442 SetDOFType OFELI:Wets, 593 SetDOFELI:Element, 298 OFELI:Element, 298 OFELI:IElement, 298 OFELI:DMatrix, 246 OFELI:DSSolver, 490 OFELI:DEsolver, 486, 487 OFELI:DEsolver, 486, 488 OFELI:LICCIDESOlver, 487, 488 OFELI:LICCIDESOlver, 487, 488 OFELI:LICCIDESOlver, 487, 488 OFELI:DEsolver, 487, 48	•	· · · · · · · · · · · · · · · · · · ·
ofeli::Rectangle, 524 setCFL ofeli::Muscl, 459 setChild offeli::Blement, 302 offeli::Blement, 302 offeli::Side, 530 setCode offeli::Blement, 296, 298 offeli::Nassolver, 470 offeli::Nassolver, 470 offeli::Side, 528 offeli::Nassolver, 470 offeli::OptSolver, 484, 485 offeli::WaterPorous2D, 612 setColumn offeli::DMatrix, 246 offeli::DMatrix, 263 setConstantMatrix offeli::OptSolver, 490 offeli::Dismote, 475 setCord offeli::TimeStepping, 565 setCoord offeli::Nassolver, 470 offeli::Dismote, 470 offeli::Dismote, 470 offeli::DSsolver, 484, 485 offeli::Nassolver, 470 offeli::DSsolver, 484, 485 offeli::Nassolver, 470 offeli::DSsolver, 484, 485 offeli::Nassolver, 470 offeli::DSsolver, 480 offeli::Dismote, 475 setConstantMatrix offeli::Nassolver, 472 offeli::TimeStepping, 565 setCoord offeli::Dismote, 475 setCurrentDensity offeli::BiotSavart, 209 setDoffsupport offeli::Massolver, 486, 487 offeli::LCladde, 475 setCurrentDensity offeli::LCladde, 475 setCurrentDensity offeli::LCladde, 475 setDoffupport offeli::LCladde, 475 offeli::LCladde, 475 setDoffupport offeli::LCladde, 475 offeli::LCladde, 475 setDoffupport offeli::LCladde, 475 o		
setCFL OFELI::Muscl, 459 setExterior  SetExterior OFELI::Element, 302 OFELI::Side, 530 SetCode OFELI::Bdee, 273 OFELI::Bdee, 273 OFELI::Blement, 296, 298 OFELI::Nassolver, 470 OFELI::Side, 528 OFELI::Nassolver, 470 OFELI::OptSolver, 484, 485 OFELI::WaterPorous2D, 612 SetColumn OFELI::DMatrix, 246 OFELI::DMatrix, 263 SetConstantMatrix OFELI::DSMatrix, 263 SetCord OFELI::DSMatrix, 263 SetConstantMatrix OFELI::DSMatrix, 263 SetCord OFELI::TimeStepping, 565 SetCord OFELI::TimeStepping, 565 SetCord OFELI::BiotSavart, 209 SetDofSupport OFELI::Mesh, 442 SetDofType OFELI::Edge, 273 OFELI::Element, 298 OFELI::DCIDL2, 222  OFELI::TimeStepping, 565 SetIngular OFELI::DDESolver, 484, 485 OFELI::LCDDESolver, 486, 487 OFELI::DOFELI::DOESolver, 486, 487 OFELI::DOESolver, 486, 487 OFELI::DOESolver, 486, 487 OFELI::DOESolver, 486, 487 OFELI::DOESolver, 486, 487 OFELI::LCL2DT, 391 OFELI::LCL2DT, 394 OFELI::LCL2DT, 394 OFELI::LCL2DT, 397 SetInitialRHS OFELI::DOESolver, 487, 488 OFELI::DOESolver, 487, 488 OFELI::DOESolver, 487, 488 OFELI::DCIDL2, 222	•	
setChild OFELI::SteklovPoincare2DBE, 557  setChild OFELI::SteklovPoincare2DBE, 557  offeli::SteklovPoincare2DBE, 557  setFile Offeli::SteklovPoincare2DBE, 557  setFile Offeli::SteklovPoincare2DBE, 557  setFile Offeli::SteklovPoincare2DBE, 557  setCode offeli::SteklovPoincare2DBE, 557  setFile Offeli::Listege, 273 Offeli::Integration, 365 Offeli::Integration, 365 Offeli::Integration, 365 Offeli::Integration, 365 Offeli::Integration, 365 Offeli::NLASSolver, 470 Offeli::NLASSolver, 470 Offeli::OptSolver, 484 Offeli::OptSolver, 496 setGraph Offeli::OptSolver, 496 offeli::DMatrix, 246 Offeli::DMatrix, 246 Offeli::DMatrix, 263 setConstantMatrix Offeli::OptSolver, 490 Offeli::TimeStepping, 565 setCord Offeli::NdASSolver, 472 Offeli::DDESolver, 490 Offeli::Integration, 365 Offeli::OptSolver, 470 Offeli::DDESolver, 484 Offeli::OptSolver, 490 Offeli::CDESolver, 490 Offeli::CDESolver, 490 Offeli::DDESolver, 486, 487 Offeli::DDESolver, 486, 487 Offeli::CDESolver, 486, 487 Offeli::LCLDID, 391 setDOFSupport Offeli::LCLDID, 394 Offeli::LCLDID, 394 Offeli::LCLDID, 397 setDOFType offeli::OptSolver, 487, 488 Offeli::DDESolver, 487 Offeli::DDESolver,		
setChild OFELI::Element, 302 OFELI::Side, 530 SetCode OFELI::Edge, 273 OFELI::Integration, 365 OFELI::Node, 475 OFELI::Side, 528 OFELI::OptSolver, 470 OFELI::ODESolver, 484, 485 OFELI::DMatrix, 246 OFELI::DBSAdtrix, 263 SetConstantMatrix OFELI::DESolver, 490 OFELI::TimeStepping, 565 SetCoord OFELI::Node, 475 SetCurrentDensity OFELI::BiotSavart, 209 SetDOFSupport OFELI::Besh, 442 SetDOF OFELI::Mesh, 442 SetDOF OFELI::Mesh, 442 SetDOF OFELI::Mesh, 442 SetDOF OFELI::DESolver, 487, 488 OFELI::DESOlver, 487, 488 OFELI::DDESolver, 487, 488 OFELI::DCIDL2, 222		, <u>,</u>
OFELI::Element, 302 OFELI::Side, 530 SetCode OFELI::Edge, 273 OFELI::Integration, 365 OFELI::NLASSolver, 470 OFELI::Node, 475 OFELI::Side, 528 OFELI::OptSolver, 496 SetCoef OFELI::OptSolver, 484, 485 OFELI::WaterPorous2D, 612 SetGolumn OFELI::DMatrix, 246 OFELI::DSMatrix, 263 SetConstantMatrix OFELI::ODESolver, 490 OFELI::ODESolver, 490 OFELI::DDESolver, 490 OFELI::Node, 475 SetCoord OFELI::Node, 475 SetCord OFELI::Node, 475 SetCord OFELI::Node, 475 SetCord OFELI::Node, 475 SetCord OFELI::DDESolver, 490 OFELI::DDESolver, 486, 487 OFELI::DDESolver, 490 OFELI::DCIDE, 391 OFELI::LCL2DT, 394 OFELI::LCL2DT, 394 OFELI::LCL3DT, 397 SetDOFType OFELI::CDESolver, 487, 488 OFELI::DOESolver, 487, 488 OFELI::DOESolv		
OFELI:Side, 530  setCode  OFELI:Edge, 273  OFELI:Element, 296, 298  OFELI::Node, 475  OFELI::Node, 475  OFELI::Node, 475  OFELI::Node, 475  OFELI::NLASSolver, 470  OFELI::NLASSolver, 470  OFELI::NLASSolver, 470, 471  OFELI::OptSolver, 480  OFELI::OptSolver, 484, 485  OFELI::OptSolver, 484, 485  OFELI::WaterPorous2D, 612  setColumn  OFELI::DMatrix, 246  OFELI::DSMatrix, 263  setConstantMatrix  OFELI::OptSolver, 490  OFELI::OptSolver, 490  OFELI::TimeStepping, 565  setCoord  OFELI::TimeStepping, 565  setCurrentDensity  OFELI::Node, 475  setCurrentDensity  OFELI::BiotSavart, 209  setDOFSupport  OFELI::Mesh, 442  setDOFType  OFELI::Mesh, 442  setDOFType  OFELI::CodeSolver, 487, 488  OFELI::TimeStepping, 565  OFELI::CodeSolver, 487, 488  OFELI::TimeStepping, 565  SetInput  OFELI::TimeStepping, 565  SetInput  OFELI::DCIDL2, 222		
setCode OFELI::Edge, 273 OFELI::Integration, 365 OFELI::Node, 475 OFELI::Node, 475 OFELI::Node, 475 OFELI::Node, 528 OFELI::NLASSolver, 470, 471 OFELI, 168 SetCoef OFELI::OptSolver, 484, 485 OFELI::WaterPorous2D, 612 SetColumn OFELI::DMatrix, 246 OFELI::DSMatrix, 263 SetConstantMatrix OFELI::DESolver, 490 OFELI::DESolver, 490 OFELI::DESolver, 490 OFELI::TimeStepping, 565 SetCoord OFELI::Node, 475 SetCurrentDensity OFELI::Node, 475 SetDOFSupport OFELI::BiotSavart, 209 SetDOFSupport OFELI::Mesh, 442 SetDOFType OFELI::Mesh, 442 SetDOF OFELI::Mesh, 442 SetDOF OFELI::Edge, 273 OFELI::Element, 298 OFELI::DCIDL2, 222		setFile
OFELI::Edge, 273 OFELI::Integration, 365 OFELI::Element, 296, 298 OFELI::NLASSolver, 470 OFELI::Node, 475 OFELI::Node, 475 OFELI::Side, 528 OFELI::NLASSolver, 470, 471 OFELI, 168 SetCoef OFELI::ODESolver, 484, 485 OFELI::WaterPorous2D, 612 SetIli:WaterPorous2D, 612 SetColumn OFELI::DMatrix, 246 OFELI::DMatrix, 246 OFELI::DSMatrix, 263 SetConstantMatrix OFELI::ODESolver, 490 OFELI::TimeStepping, 565 SetCoord OFELI::TimeStepping, 565 SetCoord OFELI::NLASSolver, 472 OFELI::Node, 475 SetCurrentDensity OFELI::BiotSavart, 209 SetDOFSupport OFELI::BiotSavart, 209 SetDOFSupport OFELI::Mesh, 442 SetInitialRHS OFELI::CL2DT, 394 OFELI::LC1DT, 397 SetDOFType OFELI::Wect, 593 SetInitialRHS OFELI::DDESolver, 487, 488 SetDOF OFELI::Edge, 273 OFELI::TimeStepping, 565 SetInput OFELI::TimeStepping, 565	OFELI::Side, 530	OFELI, 152
OFELI::Blement, 296, 298 OFELI::Node, 475 OFELI::Node, 475 OFELI::Side, 528 OFELI::Side, 528 OFELI::OptSolver, 470, 471 OFELI, 168 SetCoef OFELI::ODESolver, 484, 485 OFELI::WaterPorous2D, 612 SetClumn OFELI::DMatrix, 246 OFELI::DMatrix, 246 OFELI::DSMatrix, 263 SetConstantMatrix OFELI::ODESolver, 490 OFELI::TimeStepping, 565 SetCoord OFELI::TimeStepping, 565 SetCourrentDensity OFELI::BiotSavart, 209 SetDOFSupport OFELI::Mesh, 442 SetDOFType OFELI::Wect, 593 SetDOFS SetDOFS OFELI::CDESolver, 488 SetDOF OFELI::Wect, 593 SetInitial OFELI::LCL2DT, 397 SetInitialRHS OFELI::TimeStepping, 565 OFELI::CDESolver, 487, 488 OFELI::LEdge, 273 OFELI::LEdge, 273 OFELI::LEdge, 273 OFELI::LEdgen, 298 OFELI::DC1DL2, 222	setCode	setFunction
OFELI::Blement, 296, 298 OFELI::Node, 475 OFELI::Node, 475 OFELI::Side, 528 OFELI::Side, 528 OFELI::OptSolver, 470, 471 OFELI, 168 SetCoef OFELI::ODESolver, 484, 485 OFELI::WaterPorous2D, 612 SetClumn OFELI::DMatrix, 246 OFELI::DMatrix, 246 OFELI::DSMatrix, 263 SetConstantMatrix OFELI::ODESolver, 490 OFELI::TimeStepping, 565 SetCoord OFELI::TimeStepping, 565 SetCourrentDensity OFELI::BiotSavart, 209 SetDOFSupport OFELI::Mesh, 442 SetDOFType OFELI::Wect, 593 SetDOFS SetDOFS OFELI::CDESolver, 488 SetDOF OFELI::Wect, 593 SetInitial OFELI::LCL2DT, 397 SetInitialRHS OFELI::TimeStepping, 565 OFELI::CDESolver, 487, 488 OFELI::LEdge, 273 OFELI::LEdge, 273 OFELI::LEdge, 273 OFELI::LEdgen, 298 OFELI::DC1DL2, 222	OFELI::Edge, 273	OFELI::Integration, 365
OFELI::Node, 475 OFELI::Side, 528 OFELI::Side, 528 OFELI::OptSolver, 470, 471 OFELI, 168 SetCoef SetGraph OFELI::ODESolver, 484, 485 OFELI::WaterPorous2D, 612 SetColumn OFELI::DMatrix, 246 OFELI::DMatrix, 246 OFELI::DSMatrix, 263 SetConstantMatrix OFELI::ODESolver, 490 OFELI::TimeStepping, 565 SetCord OFELI::TimeStepping, 565 SetCord OFELI::Node, 475 SetCurrentDensity OFELI::BiotSavart, 209 SetDOFSupport OFELI::Mesh, 442 OFELI::Mesh, 442 OFELI::LCL2DT, 394 OFELI::Vect, 593 SetDOF OFELI::Edge, 273 OFELI::Element, 298 OFELI::TimeStepping, 565 SetInput OFELI::TimeStepping, 565 OFELI::DDESolver, 487, 488 OFELI::Element, 298 OFELI::DC1DL2, 222		· ·
OFELI::Side, 528 OFELI::OptSolver, 470, 471 OFELI, 168 OFELI::OptSolver, 496  setCoef OFELI::ODESolver, 484, 485 OFELI::WaterPorous2D, 612 setColumn OFELI::DMatrix, 246 OFELI::DSMatrix, 263 setConstantMatrix OFELI::ODESolver, 490 OFELI::TimeStepping, 565 setCoord OFELI::TimeStepping, 565 setCurrentDensity OFELI::Node, 475 setCurrentDensity OFELI::BiotSavart, 209 setDOFSupport OFELI::Mesh, 442 SetDOFType OFELI::Vect, 593 setDOF OFELI::Edge, 273 OFELI::Element, 298 OFELI::DC1DL2, 222		
OFELI, 168 setCoef setGraph OFELI::ODESolver, 484, 485 OFELI::WaterPorous2D, 612 setClumn OFELI::DMatrix, 246 OFELI::DSMatrix, 263 setConstantMatrix OFELI::ODESolver, 490 OFELI::TimeStepping, 565 setCoord OFELI::Nde, 475 setCurrentDensity OFELI::BiotSavart, 209 setDOFSupport OFELI::Mesh, 442 SetDOFType OFELI::Vect, 593 setInitial OFELI::CDESolver, 487, 488 OFELI::Vect, 593 SetInitialRHS OFELI::DESolver, 487, 488 OFELI::Vect, 593 SetInitialRHS OFELI::DESolver, 487, 488 OFELI::Edge, 273 OFELI::Element, 298 OFELI::DC1DL2, 222		
setCoef OFELI::ODESolver, 484, 485 OFELI::WaterPorous2D, 612 setIli:WaterPorous2D, 612 setIli:WaterPorous2D, 612 setIli:WaterPorous2D, 612 setIli:WaterPorous2D, 612 setIli:Vect, 595 oFELI::Vect, 595 setInOutflowBC OFELI::DSMatrix, 263 setConstantMatrix OFELI::ODESolver, 490 OFELI::ODESolver, 490 OFELI::NLASSolver, 472 OFELI::TimeStepping, 565 setCoord OFELI::Node, 475 setCoord OFELI::Node, 475 setCurrentDensity OFELI::BiotSavart, 209 setDOFSupport OFELI::BiotSavart, 209 setDOFSupport OFELI::Mesh, 442 OFELI::LCL2DT, 394 OFELI::LCL3DT, 397 setDOFType OFELI::Vect, 593 setDOF OFELI::Edge, 273 OFELI::Edge, 273 OFELI::Element, 298 OFELI::DC1DL2, 222		
OFELI::ODESolver, 484, 485 OFELI::WaterPorous2D, 612 setColumn OFELI::DMatrix, 246 OFELI::DSMatrix, 263 setConstantMatrix OFELI::ODESolver, 490 OFELI::TimeStepping, 565 setCoord OFELI::Node, 475 setCurrentDensity OFELI::BiotSavart, 209 setDOFSupport OFELI::Mesh, 442 SetDOFType OFELI::Vect, 593 SetDOF OFELI::Edge, 273 OFELI::Element, 298 OFELI::DCESOlver, 484, 485 SetInitial OFELI::DESolver, 486, 487 OFELI::LCL2DT, 394 OFELI::LCL2DT, 397 SetInitialRHS OFELI::CDESolver, 487, 488 OFELI::DDESolver, 487, 488 OFELI::DDESolver, 487, 488 OFELI::Edge, 273 OFELI::Element, 298 OFELI::DC1DL2, 222		
OFELI::WaterPorous2D, 612  setColumn OFELI::DMatrix, 246 OFELI::DSMatrix, 263 setInOutflowBC OFELI::ICPG1D, 356 setConstantMatrix OFELI::ODESolver, 490 OFELI::TimeStepping, 565 setCoord OFELI::NLASSolver, 472 OFELI::TimeStepping, 564 OFELI::Node, 475 setCourrentDensity OFELI::BiotSavart, 209 setDOFSupport OFELI::Mesh, 442 SetDOFType OFELI::Wect, 593 SetDOF OFELI::CDESolver, 486, 487 OFELI::LCL2DT, 394 OFELI::LCL2DT, 394 OFELI::LCL3DT, 397 setDOFSupport OFELI::CDESolver, 487, 488 setDOF OFELI::LEdge, 273 OFELI::Edge, 273 OFELI::Element, 298 OFELI::DC1DL2, 222		*
setColumn OFELI::DMatrix, 246 OFELI::DSMatrix, 263 setConstantMatrix OFELI::ODESolver, 490 OFELI::TimeStepping, 565 setCoord OFELI::Node, 475 setCurrentDensity OFELI::BiotSavart, 209 setDOFSupport OFELI::Mesh, 442 setDOFType OFELI::Wect, 593 setDOF OFELI::Vect, 593 OFELI::Edge, 273 OFELI::Element, 298  OFELI::DC1DL2, 222  SetInOutflowBC OFELI::ICPG1D, 356 setInitialConditial OFELI::NLASSolver, 472 OFELI::NLASSolver, 472 OFELI::NLESolver, 486, 487 OFELI::TimeStepping, 564 setInitialCondition OFELI::ICPG1D, 354 OFELI::LCL1D, 391 OFELI::LCL2DT, 394 OFELI::LCL3DT, 397 setInitialRHS OFELI::ODESolver, 487, 488 OFELI::DESolver, 487, 488 OFELI::DESolver, 487, 488 OFELI::DESOLVER, 273 OFELI::DC1DL2, 222		
OFELI::DMatrix, 246 OFELI::DSMatrix, 263 SetConstantMatrix OFELI::ODESolver, 490 OFELI::TimeStepping, 565 SetCoord OFELI::Node, 475 SetCurrentDensity OFELI::BiotSavart, 209 SetDOFSupport OFELI::Mesh, 442 SetDOFType OFELI::Wect, 593 SetDOF OFELI::Vect, 593 OFELI::Edge, 273 OFELI::Element, 298 SetInoutflowBC OFELI::ICPG1D, 356 SetInitial OFELI::NLASSolver, 472 OFELI::ODESolver, 486, 487 OFELI::TimeStepping, 564 SetInitialCondition SetInitialCondition OFELI::ICPG1D, 354 OFELI::LCL1D, 391 OFELI::LCL2DT, 394 OFELI::LCL3DT, 397 SetInitialRHS OFELI::ODESolver, 487, 488 OFELI::TimeStepping, 565 SetInput OFELI::Element, 298 OFELI::DC1DL2, 222		· ·
OFELI::DSMatrix, 263 setConstantMatrix OFELI::ODESolver, 490 OFELI::TimeStepping, 565 setCoord OFELI::Node, 475 setCurrentDensity OFELI::BiotSavart, 209 setDOFSupport OFELI::Mesh, 442 SetDOFType OFELI::Wect, 593 setDOF OFELI::Vect, 593 SetDOF OFELI::Edge, 273 OFELI::Element, 298 OFELI::DC1DL2, 222 OFELI::DC1DL2, 222		·
setConstantMatrix OFELI::ODESolver, 490 OFELI::TimeStepping, 565 SetCoord OFELI::Node, 475 SetCurrentDensity OFELI::BiotSavart, 209 SetDOFSupport OFELI::Mesh, 442 SetDOFType OFELI::Vect, 593 SetDOF OFELI::Vect, 593 SetDOF OFELI::Edge, 273 OFELI::Element, 298 OFELI::DC1DL2, 222  OFELI::DC1DL2, 222		
OFELI::ODESolver, 490 OFELI::TimeStepping, 565 SetCoord OFELI::Node, 475 SetCurrentDensity OFELI::BiotSavart, 209 SetDOFSupport OFELI::Mesh, 442 OFELI::Mesh, 442 SetDOFType OFELI::Vect, 593 SetDOF OFELI::Vect, 593 SetDOF OFELI::Edge, 273 OFELI::Element, 298 OFELI::DC1DL2, 222	· ·	
OFELI::TimeStepping, 565 setCoord OFELI::Node, 475 setCurrentDensity OFELI::BiotSavart, 209 setDOFSupport OFELI::Mesh, 442 OFELI::Mesh, 442 OFELI::West, 593 setDOF OFELI::Vect, 593 SetDOF OFELI::Edge, 273 OFELI::Element, 298 OFELI::DOESolver, 486, 487 OFELI::ODESolver, 354 OFELI::CL1D, 391 OFELI::LCL2DT, 394 OFELI::LCL3DT, 397 SetInitialRHS OFELI::ODESolver, 487, 488 OFELI::TimeStepping, 565 SetInput OFELI::Element, 298 OFELI::DC1DL2, 222		
setCoord OFELI::TimeStepping, 564 OFELI::Node, 475 setInitialCondition setCurrentDensity OFELI::ICPG1D, 354 OFELI::BiotSavart, 209 OFELI::LCL1D, 391 setDOFSupport OFELI::LCL2DT, 394 OFELI::Mesh, 442 OFELI::LCL3DT, 397 setDOFType setInitialRHS OFELI::Vect, 593 SetInitialRHS oFELI::ODESolver, 487, 488 setDOF OFELI::Edge, 273 OFELI::Element, 298 OFELI::DC1DL2, 222		OFELI::NLASSolver, 472
OFELI::Node, 475 setCurrentDensity OFELI::BiotSavart, 209 oFELI::BiotSavart, 209 oFELI::LCL1D, 391 setDOFSupport OFELI::Mesh, 442 OFELI::LCL2DT, 394 OFELI::LCL3DT, 397 setDOFType setInitialRHS OFELI::Vect, 593 setDOF OFELI::Vect, 593 setDOF OFELI::Edge, 273 OFELI::Element, 298 oFELI::DC1DL2, 222	OFELI::TimeStepping, 565	OFELI::ODESolver, 486, 487
setCurrentDensity OFELI::ICPG1D, 354 OFELI::BiotSavart, 209 OFELI::LCL1D, 391 SetDOFSupport OFELI::Mesh, 442 OFELI::LCL2DT, 394 OFELI::LCL3DT, 397 SetDOFType SetInitialRHS OFELI::Vect, 593 SetDOF OFELI::Vect, 593 SetDOF OFELI::Edge, 273 OFELI::Element, 298 OFELI::DC1DL2, 222	setCoord	OFELI::TimeStepping, 564
OFELI::BiotSavart, 209  setDOFSupport OFELI::LCL2DT, 394 OFELI::Mesh, 442 OFELI::LCL3DT, 397  setDOFType setInitialRHS OFELI::Vect, 593 SetDOF OFELI::Edge, 273 OFELI::Element, 298 OFELI::DC1DL2, 222	OFELI::Node, 475	setInitialCondition
OFELI::BiotSavart, 209  setDOFSupport OFELI::LCL2DT, 394 OFELI::Mesh, 442 OFELI::LCL3DT, 397  setDOFType setInitialRHS OFELI::Vect, 593 SetDOF OFELI::Edge, 273 OFELI::Element, 298 OFELI::DC1DL2, 222	setCurrentDensity	OFELI::ICPG1D, 354
setDOFSupport OFELI::Mesh, 442 OFELI::LCL2DT, 394 OFELI::Mesh, 442 OFELI::LCL3DT, 397 setDOFType setInitialRHS OFELI::ODESolver, 487, 488 oFELI::TimeStepping, 565 OFELI::Edge, 273 OFELI::Element, 298 OFELI::DC1DL2, 222	•	OFELI::LCL1D, 391
OFELI::Mesh, 442 SetDOFType SetInitialRHS OFELI::Vect, 593 SetDOF OFELI::Edge, 273 OFELI::Element, 298 OFELI::DC1DL2, 222	•	•
setDOFType setInitialRHS OFELI::Vect, 593 OFELI::ODESolver, 487, 488 setDOF OFELI::Edge, 273 SetInput OFELI::Element, 298 OFELI::DC1DL2, 222		
OFELI::Vect, 593  setDOF OFELI::Edge, 273 OFELI::Element, 298  OFELI::DC1DL2, 222		•
setDOF OFELI::TimeStepping, 565 OFELI::Edge, 273 setInput OFELI::Element, 298 OFELI::DC1DL2, 222		
OFELI::Edge, 273 setInput OFELI::Element, 298 OFELI::DC1DL2, 222		
OFELI::Element, 298 OFELI::DC1DL2, 222		11 0
		*
Ofeli::Node, 4/0 Ofeli::DC2D13, 228		
	Offilinoue, 4/0	OFELI::DC2D13, 228

OFELI::NSP2DQ41, 480	setMesh
OFELI::TINS2DT3S, 570	OFELI::SkMatrix, 536
OFELI::TINS3DT4S, 572	OFELI::SkSMatrix, 546
setLDLt	OFELI::SteklovPoincare2DBE, 557
OFELI::DSMatrix, 264	OFELI::Vect, 592
OFELI::SkSMatrix, 549	OFELI, 157
setLabel	setMeshFile
OFELI::Element, 296	OFELI, 151
setLevel	setMethod
OFELI::Node, 477	OFELI::Muscl, 460
setLimiter	setNLTerm
OFELI::Muscl, 460	OFELI::TimeStepping, 566
setLinear	setNLTerm0
OFELI::ODESolver, 485	OFELI::TimeStepping, 566
setLinearSolver	setNbDOFPerNode
OFELI::AbsEqua, 199	OFELI::Mesh, 442
OFELI::ODESolver, 490	setNbEq
OFELI::TimeStepping, 563, 565	OFELI::ODESolver, 484
setList	setNewmarkParameters
	OFELI::ODESolver, 489
OFELI::Mesh, 447 setLocal	the contract of the contract o
00000	OFELI::TimeStepping, 565
OFELL Parts ( 504	setNoKeep
OFELL:Penta6, 504	OFELI::MeshAdapt, 455
OFELI::Quad4, 521	setNoScaling
OFELI::Triang6S, 576	OFELI::MeshAdapt, 455
setLowerBound	setNodeBC
OFELI::OptSolver, 497	OFELI::Vect, 595–598
setLowerBounds	setNodeCodes
OFELI::OptSolver, 497	Finite Element Mesh, 103
setLU	setNonConstantMatrix
OFELI::BMatrix, 215	OFELI::ODESolver, 490
OFELI::DMatrix, 250	OFELI::TimeStepping, 565
OFELI::SkMatrix, 540	setObjective
setMagneticInduction	OFELI::OptSolver, 496
OFELI::BiotSavart, 209	setOnBoundary
setMaterial	OFELI::Node, 476
OFELI::Mesh, 444	setOptClass
setMaterialProperty	OFELI::OptSolver, 496
OFELI::Equation, 328	setOptMethod
setMatrices	OFELI::OptSolver, 495
OFELI::ODESolver, 488	setOutputFile
setMatrix	OFELI::Domain, 258
OFELI::EigenProblemSolver, 278	setPDE
OFELI::LinearSolver, 405, 406	OFELI::EigenProblemSolver, 278
OFELI::Prec, 516, 517	OFELI::NLASSolver, 471
setMatrixType	OFELI::TimeStepping, 564
OFELI::AbsEqua, 199	setPenalty
setMaxIter	OFELI::NSP2DQ41, 480
OFELI::Iter, 377	setPermeability
OFELI::LinearSolver, 405	OFELI::BiotSavart, 209
OFELI::NLASSolver, 470	setPointInDomain
OFELI::ODESolver, 490	OFELI::Mesh, 443
setMaxNbVertices	setQR
OFELI::MeshAdapt, 454	OFELI::DMatrix, 248
•	

setQuadrilateral	setTheta
OFELI::Integration, 365	OFELI::MeshAdapt, 455
setRHS	setTimeStep
OFELI::ODESolver, 489	OFELI::Muscl, 459
setRK3_TVDRHS	setTolerance
OFELI::TimeStepping, 564	OFELI::EigenProblemSolver, 280
setRK4RHS	OFELI::Iter, 377
OFELI::ODESolver, 486	OFELI::NLASSolver, 470
OFELI::TimeStepping, 564	OFELI::ODESolver, 490
setRatio	OFELI::OptSolver, 498
OFELI::MeshAdapt, 454	setTraction
setReconstruction	OFELI::Laplace1DL2, 380
OFELI::ICPG2DT, 359	OFELI::Laplace1DL3, 382
OFELI::ICPG3DT, 363	setTransLU
OFELI::Muscl, 460	OFELI::DMatrix, 250
OFELI::Muscl2DT, 462	setTransQR
OFELI::Muscl3DT, 464	OFELI::DMatrix, 248
setReferenceLength	setTriangle
OFELI::Muscl, 459	OFELI::Gauss, 345
setRelaxation	OFELI::Integration, 365
OFELI::MeshAdapt, 454	setType
setRow	OFELI::Estimator, 330
OFELI::DMatrix, 246	OFELI::Prec, 516
OFELI::DSMatrix, 263	setUniform
setSAOpt	OFELI::Vect, 608
OFELI::OptSolver, 497	setUpperBound
setScheme	OFELI::OptSolver, 496
OFELI::Integration, 365	setUpperBounds
setSide	OFELI::OptSolver, 496
	•
OFELI::Element, 300 setSideBC	setVelocity OFELI::LCL1D, 392
	OFELI::LCL2DT, 395
OFELI::Vect, 596–599 setSideCodes	
	OFELI::LCL3DT, 397, 398
Finite Element Mesh, 103	setVerbose
setSize	OFELL:Iter, 378
OFELI::BMatrix, 214	OFELI::Muscl, 460
OFELI::DMatrix, 244	OFELI::TimeStepping, 566
OFELI::DSMatrix, 261	setVerbosity
OFELI::Vect, 593	OFELI::OptSolver, 497
OFELI, 158, 163	setVertices
setSkyline	OFELI::Polygon, 513
OFELI::SkMatrix, 536	setXMax
OFELI::SkSMatrix, 546	OFELI, 167
setSolution	setXMin
OFELI::Estimator, 330	OFELI, 166
setSolver	setF
OFELI::AbsEqua, 198	OFELI::ODESolver, 485
OFELI::ICPG2DT, 359	setf
OFELI::LinearSolver, 406	OFELI::NLASSolver, 471
OFELI, 162	setN
setStab	OFELI, 168
OFELI::Equa_Therm, 316	Sh
setSubspaceDimension	OFELI::FEShape, 334
OFELI::EigenProblemSolver, 279	OFELI::Line2, 400

OFELI::Tetra4, 559	OFELI::DMatrix, 251, 252
OFELI::Triang3, 573	solveTransQR
OFELI::Triang6S, 575	OFELI::DMatrix, 249
Shape Function, 107	Solver, 108
ShearForce	BiCGStab, 113, 114
OFELI::Beam3DL2, 206	BiCG, 111, 112
Side, 524	CGS, 117, 118
OFELI::Side, 527	CG, 115, 116
side_assembly	GMRes, 119, 120
General Purpose Equations, 20, 21	GS, 121
SideAssembly	IterationLoop, 111
OFELI::Equation, 326–328	Jacobi, 122
SideInElement	MAX_NB_EQUATIONS, 110
Finite Element Mesh, 105	MAX_NB_INPUT_FIELDS, 110
SideList, 531	MAX_NB_MESHES, 110
SideLoop Finite Flament Mach, 90	operator<<, 125
Finite Element Mesh, 90	Richardson, 122
SideNodeCoordinates	SSOR, 124
OFELI::Equation, 324	Schur, 123
SideNodeVector	TIME_LOOP, 110
OFELI::Equation, 321	TimeLoop, 110
SideType	SolverType
OFELI::Side, 527	OFELI::Muscl, 459
SideVector	SpMatrix
OFELI::Equation, 323	OFELI, 154–156
SidesAreDOF	SpMatrix $<$ T $>$ , 552
OFELI::Mesh, 449	Sphere, 550
SkMatrix	OFELI::Sphere, 551
OFELI::SkMatrix, 535, 536	SqrDistance
SkMatrix $<$ T $>$ , 533	Utilities, 49, 53
SkSMatrix	Start
OFELI::SkSMatrix, 544, 545	OFELI::Timer, 561
SkSMatrix $<$ T $>$ , 542	Started
Smooth	OFELI::Timer, 561
OFELI::LaplaceDG2DP1, 389	SteklovPoincare2DBE, 555
Solid Mechanics, 29	OFELI::SteklovPoincare2DBE, 556
solve	Stiffness
OFELI::BMatrix, 215, 216	OFELI::Bar2DL2, 202
OFELI::DMatrix, 250, 251	Stop
OFELI::DSMatrix, 265, 266	OFELI::Timer, 561
OFELI::LinearSolver, 406, 407	Strain
OFELI::LocalMatrix, 413	OFELI::Elas2DQ4, 283
OFELI::Matrix, 427	OFELI::Elas2DT3, 288
OFELI::Prec, 517	Stress
OFELI::SkMatrix, 541	OFELI::Elas2DQ4, 283
OFELI::SkSMatrix, 549, 550	OFELI::Elas2DT3, 288
OFELI, 161, 164, 165	Symmetrize
solveLDLt	OFELI::LocalMatrix, 412
OFELI::SkSMatrix, 549	Of EELEocaliviatily, 412
solveLinearSystem	TEMPERATURE_FIELD
OFELI::AbsEqua, 199, 200	OFELI, 147
solveQR	TIME_LOOP
OFELI::DMatrix, 249	Solver, 110
solveTrans	TINS2DT3S, 568
bolve litulo	1111020100,000

OFELI::TINS2DT3S, 569, 570	Global Variables, 84
TINS3DT4S, 570	TimeLoop
OFELI::TINS3DT4S, 571	Solver, 110
TMult	TimeScheme
OFELI::DMatrix, 247	OFELI, 147
OFELI::DSMatrix, 265	TimeStepping, 561
OFELI::SkMatrix, 538	OFELI::TimeStepping, 563
OFELI::SkSMatrix, 547	Timer, 560
OFELI, 160	TrMatrix
TMultAdd	OFELI, 163
OFELI::SkMatrix, 537	TrMatrix $<$ T <sub>-</sub> $>$ , 579
TRACTION	TransSolve
OFELI, 147	OFELI::Prec, 517
TRANSIENT_ONE_STEP	transferBC
OFELI, 147	OFELI::Vect, 600
TRANSIENT	Triang3, 572
OFELI, 147	OFELI::Triang3, 573
TRAPEZOIDAL	O
	Triang6S, 574
OFELI, 149	OFELI::Triang6S, 575
TRIDIAGONAL	Triangle, 578
OFELI, 148	OFELI::Triangle, 579
TRUNCATED_NEWTON	triangle, 576
OFELI::OptSolver, 494	OFELI::triangle, 578
Tabulation, 557	TwistingMoment
Tetra4, 558	OFELI::Beam3DL2, 206
TheEdge	LINICVMMETDIC
Finite Element Mesh, 89	UNSYMMETRIC
theEdge	OFELI, 148
Global Variables, 83	unselectCode
The Element	OFELI::EdgeList, 274
Finite Element Mesh, 89	OFELI::ElementList, 303
theElement	OFELL Side List, 478
Global Variables, 82	OFELI::SideList, 533
theFinalTime	updateBC
Global Variables, 84	OFELI::Equation, 320
theIteration	Utilities, 32
Global Variables, 83	areClose, 49
TheNode	Axpy, 67
Finite Element Mesh, 89	BSpline, 64
theNode	banner, 65
Global Variables, 82	CATCH_EXCEPTION, 40
theNodeLabel	clear, 67
Finite Element Mesh, 90	Discrepancy, 54
TheSide	Distance, 49, 53
Finite Element Mesh, 89	Dot, 67
theSide	EVAL, 40
Global Variables, 83	Equal, 68
theStep	getBamg, 56
Global Variables, 83	getEasymesh, 56
theTime	getGambit, 57
Global Variables, 84	getGmsh, 57
theTimeStep	getMatlab, 59
Global Variables, 84	getMesh, 55
theTolerance	getNetgen, 59

getTetgen, 60 getTriangle, 60 Nrm2, 67 OFELI_EPSMCH, 40 OFELI_GAUSS2, 40 OFELI_IMAG, 40 OFELI_ONEOVERPI, 39 OFELI_PI, 39 OFELI_SIXTH, 39 OFELI_SQRT2, 39 OFELI_SQRT3, 39 OFELI_THIRD, 39 OFELI_TOLERANCE, 40 OFELI_TWELVETH, 39 OFELI_E, 39 operator<, 50 operator*, 48, 52, 53 operator*, 46, 47, 50, 51 operator-, 47, 51 operator-, 47, 51 operator-, 49, 53 operator==, 46, 50 PARSE, 40 qksort, 65, 66 QuickSort, 65 saveBamg, 64 saveField, 41, 42 saveGmsh, 45, 46, 61 saveGnuplot, 43, 62 saveMatlab, 62 saveMesh, 61 saveTecplot, 44, 63 saveVTK, 44, 45, 63 Scale, 66 SqrDistance, 49, 53 VLG, 40	Dot, 74, 78 Imag, 79 Modulus, 79 operator <<, 75–77, 79, 80 operator >>, 79 operator +, 72–78, 80 operator +, 73, 74, 78 operator /, 73, 74, 78 operator /, 73, 74, 78 Real, 79 Verbosity Global Variables, 83  WNORM1 OFELI, 195 WNORM2 OFELI, 195 WaterPorous2D, 610 OFELI::WaterPorous2D, 611 WithMesh OFELI::Vect, 594  Xpy Utilities, 67
Xpy, 67  VANALBADA_LIMITER     OFELI::Muscl, 459  VANLEER_LIMITER     OFELI::Muscl, 459  VELOCITY_FIELD     OFELI, 147  VFROE_SOLVER     OFELI::Muscl, 459  VISCOSITY     OFELI, 146  VLG     Utilities, 40  Vect     OFELI::Vect, 587–590  Vect< T_ >, 581  Vector and Matrix, 69	