TUNA::RBF

Generated by Doxygen 1.5.8

Thu Jul 2 14:07:17 2009

Contents

1 Todo List **1**

2 Namespace Index **2**

2.1 Namespace List 2

3 Class Index **2**

3.1 Class Hierarchy 2

4 Class Index **4**

4.1 Class List 4

5 File Index **6**

5.1 File List 6

6 Namespace Documentation **8**

6.1 flens Namespace Reference 8

6.1.1 Detailed Description 9

6.1.2 Function Documentation 9

6.2 KDTree Namespace Reference 12

6.2.1 Detailed Description 12

6.3 RBF Namespace Reference 12

6.3.1 Detailed Description 14

6.3.2 Function Documentation 14

6.4 Solver Namespace Reference 15

6.4.1 Detailed Description 16

6.4.2 Function Documentation 16

7 Class Documentation **16**

7.1 BoxKnots< Tprec > Class Template Reference 16

7.1.1 Detailed Description 18

7.1.2 Constructor & Destructor Documentation 18

7.1.3 Member Function Documentation 19

7.2 CartComm Class Reference 19

7.2.1 Detailed Description 19

7.3 flens::ACBFPrec< MA, TD > Class Template Reference 20

7.3.1 Detailed Description 21

7.3.2 Member Function Documentation 21

7.4 flens::DiagonalPrec< MA > Class Template Reference 21
7.4.1 Detailed Description 21
7.5 flens::gmres_< A > Struct Template Reference 22
7.5.1 Detailed Description 22
7.6 flens::gmres_< DenseVector< I > > Struct Template Reference 22
7.6.1 Detailed Description 22
7.7 flens::IdentityPrec< MA > Class Template Reference 22
7.7.1 Detailed Description 22
7.8 knot1D Struct Reference 23
7.8.1 Detailed Description 23
7.8.2 Member Function Documentation 23
7.9 knot2D Struct Reference 23
7.9.1 Detailed Description 24
7.9.2 Member Function Documentation 24
7.10 knot3D Struct Reference 24
7.10.1 Detailed Description 25
7.10.2 Member Function Documentation 25
7.11 Knots< Tdomain, Tknot > Class Template Reference 25
7.11.1 Detailed Description 27
7.11.2 Member Function Documentation 27
7.12 LineKnots< Tprec > Class Template Reference 30
7.12.1 Detailed Description 31
7.12.2 Constructor & Destructor Documentation 31
7.12.3 Member Function Documentation 31
7.13 OVRectangleKnots< Tprec > Class Template Reference 32
7.13.1 Detailed Description 33
7.13.2 Constructor & Destructor Documentation 33
7.13.3 Member Function Documentation 34
7.14 PARectangleKnots< Tprec > Class Template Reference 35
7.14.1 Detailed Description 36
7.14.2 Constructor & Destructor Documentation 36
7.14.3 Member Function Documentation 36
7.15 RBF::MQ< T, Dim > Class Template Reference 37
7.15.1 Detailed Description 37
7.16 RBF::MQ< T, 1 > Class Template Reference 38
7.16.1 Detailed Description 38

7.17 RBF::MQ< T, 2 > Class Template Reference 38
7.17.1 Detailed Description 38
7.18 RBF::MQ< T, 3 > Class Template Reference 39
7.18.1 Detailed Description 39
7.19 RBF::MQ_1DX< T, Dim > Class Template Reference 39
7.19.1 Detailed Description 39
7.20 RBF::MQ_1DX< T, 1 > Class Template Reference 39
7.20.1 Detailed Description 40
7.21 RBF::MQ_1DX< T, 2 > Class Template Reference 40
7.21.1 Detailed Description 40
7.22 RBF::MQ_1DX< T, 3 > Class Template Reference 40
7.22.1 Detailed Description 41
7.23 RBF::MQ_1DY< T, Dim > Class Template Reference 41
7.23.1 Detailed Description 41
7.24 RBF::MQ_1DY< T, 2 > Class Template Reference 41
7.24.1 Detailed Description 41
7.25 RBF::MQ_1DY< T, 3 > Class Template Reference 42
7.25.1 Detailed Description 42
7.26 RBF::MQ_1DZ< T, Dim > Class Template Reference 42
7.26.1 Detailed Description 42
7.27 RBF::MQ_1DZ< T, 3 > Class Template Reference 43
7.27.1 Detailed Description 43
7.28 RBF::MQ_2DX< T, Dim > Class Template Reference 43
7.28.1 Detailed Description 43
7.29 RBF::MQ_2DX< T, 1 > Class Template Reference 43
7.29.1 Detailed Description 44
7.30 RBF::MQ_2DX< T, 2 > Class Template Reference 44
7.30.1 Detailed Description 44
7.31 RBF::MQ_2DX< T, 3 > Class Template Reference 44
7.31.1 Detailed Description 45
7.32 RBF::MQ_2DY< T, Dim > Class Template Reference 45
7.32.1 Detailed Description 45
7.33 RBF::MQ_2DY< T, 2 > Class Template Reference 45
7.33.1 Detailed Description 45
7.34 RBF::MQ_2DY< T, 3 > Class Template Reference 46
7.34.1 Detailed Description 46

7.35 RBF::MQ_2DZ< T, Dim > Class Template Reference 46
7.35.1 Detailed Description 46
7.36 RBF::MQ_2DZ< T, 3 > Class Template Reference 47
7.36.1 Detailed Description 47
7.37 RBF::TPS< T, Dim > Class Template Reference 47
7.37.1 Detailed Description 47
7.38 RBF::TPS< T, 2 > Class Template Reference 47
7.38.1 Detailed Description 48
7.39 RBF::TPS_1DX< T, Dim > Class Template Reference 48
7.39.1 Detailed Description 48
7.40 RBF::TPS_1DX< T, 2 > Class Template Reference 48
7.40.1 Detailed Description 48
7.41 RBF::TPS_1DY< T, Dim > Class Template Reference 49
7.41.1 Detailed Description 49
7.42 RBF::TPS_1DY< T, 2 > Class Template Reference 49
7.42.1 Detailed Description 49
7.43 RBF::TPS_2DX< T, Dim > Class Template Reference 49
7.43.1 Detailed Description 49
7.44 RBF::TPS_2DX< T, 2 > Class Template Reference 50
7.44.1 Detailed Description 50
7.45 RBF::TPS_2DY< T, Dim > Class Template Reference 50
7.45.1 Detailed Description 50
7.46 RBF::TPS_2DY< T, 2 > Class Template Reference 51
7.46.1 Detailed Description 51
7.47 RectangleKnots< Tprec > Class Template Reference 51
7.47.1 Detailed Description 52
7.47.2 Constructor & Destructor Documentation 52
7.47.3 Member Function Documentation 53
7.48 RectBlocksKnots< Tprec > Class Template Reference 54
7.48.1 Detailed Description 55
7.48.2 Constructor & Destructor Documentation 55
7.48.3 Member Function Documentation 56
7.49 SemiCircleKnots< Tprec > Class Template Reference 57
7.49.1 Detailed Description 58
7.49.2 Constructor & Destructor Documentation 58
7.49.3 Member Function Documentation 59

7.50 timer Struct Reference 60
7.50.1 Detailed Description 60

8 File Documentation **60**

8.1 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/01TestKnots/box.cpp File Reference 60
8.1.1 Detailed Description 60
8.2 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/01TestKnots/line.cpp File Reference 61
8.2.1 Detailed Description 61
8.3 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/01TestKnots/rectangle.cpp File Reference 62
8.3.1 Detailed Description 62
8.4 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/01TestKnots/rectangle_-blocks.cpp File Reference 63
8.4.1 Detailed Description 63
8.5 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/01TestKnots/semicircle.cpp File Reference 64
8.5.1 Detailed Description 64
8.6 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/02Heat2D/heat2D.cpp File Reference 65
8.6.1 Detailed Description 65
8.7 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/03HeatSemiCircle/heatsemi.cpp File Reference 66
8.7.1 Detailed Description 66
8.8 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/04ConvDiff1D/convdiff01.cpp File Reference 67
8.8.1 Detailed Description 67
8.9 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/05ConvDiff2D/convdiff2D.cpp File Reference 68
8.9.1 Detailed Description 68
8.10 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/07LidDriven/lid_-driven2.cpp File Reference 69
8.10.1 Detailed Description 69
8.11 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/08BackwardFacingStep/backfacing2.cp File Reference 70
8.11.1 Detailed Description 70
8.12 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/accessor.hpp File Reference 72
8.12.1 Detailed Description 72

8.13 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/allocator.hpp File Reference . . . 72
8.13.1 Detailed Description . . . 72
8.14 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/iterator.hpp File Reference . . . 72
8.14.1 Detailed Description . . . 73
8.15 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/kdtree.hpp File Reference . . . 73
8.15.1 Detailed Description . . . 73
8.16 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/node.hpp File Reference . . . 74
8.16.1 Detailed Description . . . 74
8.17 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/region.hpp File Reference . . . 74
8.17.1 Detailed Description . . . 74
8.18 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Solvers/Krylov/gmres.hpp File Reference . . . 74
8.18.1 Detailed Description . . . 75
8.19 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Traits.hpp File Reference . . . 75
8.19.1 Detailed Description . . . 76

1 Todo List

Class CartComm Extend the functionality of CartComm class to 1D and 3D (templates)

Member flens::ACBFPrec::construct(TD domain, int max_knots, int sp=0, bool print_screen=false) Optimize the Matrix-Matrix multiplication

Member flens::ACBFPrec::construct(TD domain, int max_knots, int sp=0, bool print_screen=false) Optimize this process (L-QR-SVD), basically I need to incorporate the algorithm described in Kansa & Ling, Adv. Comp. Math. 2005, 23: 31-54, pag 39.

Class Knots< Tdomain, Tknot > To find a way to generate knot for 1D, 2D or 3D without -D_1D_, -D_2D_ and -D_3D_ compiler options.

Member Solver::Gauss(Mat &A, Vec &b) Check the matrix access, remember that I'm using ColMajor for all the matrices (see FLENS manual).

2 Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

flens (GMRES, preconditioners, based on FLENS) 8

KDTree (Contains all the stuff for KDTree algorithm) 12

RBF (Implements several RBFs) 12

Solver (Linear system solvers) 15

3 Class Index

3.1 Class Hierarchy

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

CartComm 19

flens::ACBFPrec< MA, TD > 20

flens::DiagonalPrec< MA > 21

flens::gmres_< A > 22

flens::gmres_< DenseVector< I > > 22

flens::IdentityPrec< MA > 22

knot1D 23

knot2D 23

knot3D 24

Knots< Tdomain, Tknot > 25

Knots< BoxKnots< Tprec >, knot3D > 25

BoxKnots< Tprec > 16

Knots< LineKnots< Tprec >, knot1D > 25

LineKnots< Tprec > 30

Knots< OVRectangleKnots< Tprec >, knot2D > 25

OVRectangleKnots< Tprec > 32

Knots< PARectangleKnots< Tprec >, knot2D > 25

PARectangleKnots< Tprec > 35

Knots< RectangleKnots< Tprec >, knot2D > 25

RectangleKnots< Tprec > 51

Knots< RectBlocksKnots< Tprec >, knot2D > 25

RectBlocksKnots< Tprec > 54

Knots< SemiCircleKnots< Tprec >, knot2D > 25

SemiCircleKnots< Tprec > 57

RBF::MQ< T, Dim > 37

RBF::MQ< T, 1 > 38

RBF::MQ< T, 2 > 38

RBF::MQ< T, 3 > 39

RBF::MQ_1DX< T, Dim > 39

RBF::MQ_1DX< T, 1 > 39

RBF::MQ_1DX< T, 2 > 40

RBF::MQ_1DX< T, 3 > 40

RBF::MQ_1DY< T, Dim > 41

RBF::MQ_1DY< T, 2 > 41

RBF::MQ_1DY< T, 3 > 42

RBF::MQ_1DZ< T, Dim > 42

RBF::MQ_1DZ< T, 3 > 43

RBF::MQ_2DX< T, Dim > 43

RBF::MQ_2DX< T, 1 > 43

RBF::MQ_2DX< T, 2 > 44

RBF::MQ_2DX< T, 3 > 44

RBF::MQ_2DY< T, Dim > 45

RBF::MQ_2DY< T, 2 > 45

RBF::MQ_2DY< T, 3 > 46

RBF::MQ_2DZ< T, Dim > 46

RBF::MQ_2DZ< T, 3 > 47

RBF::TPS< T, Dim > 47

RBF::TPS< T, 2 > 47

RBF::TPS_1DX< T, Dim > 48

RBF::TPS_1DX< T, 2 > 48

RBF::TPS_1DY< T, Dim > 49

RBF::TPS_1DY< T, 2 > 49

RBF::TPS_2DX< T, Dim > 49

RBF::TPS_2DX< T, 2 > 50

RBF::TPS_2DY< T, Dim > 50

RBF::TPS_2DY< T, 2 > 51

timer 60

4 Class Index

4.1 Class List

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

BoxKnots< Tprec > (Octoedro or Cuboide) 16

CartComm (2D Catersian communicator in the MPI environment) 19

flens::ACBFPrec< MA, TD > (ACBF Preconditioner) 20

flens::DiagonalPrec< MA > (Diagonal Preconditioner) 21

flens::gmres_< A > (Type definition for GMRES) 22

flens::gmres_< DenseVector< I > > (Dense Vector specialization) 22

flens::IdentityPrec< MA > (Identity Preconditioner) 22

knot1D (The leaves of the **KDTree** in 1D) 23

knot2D (The leaves of the **KDTree** in 2D) 23

knot3D (The leaves of the **KDTree** in 3D) 24

Knots< Tdomain, Tknot > (Base class for knots generation) 25

LineKnots< Tprec > (**Knots** in a line) 30

OVRectangleKnots< Tprec > (OVRectangular domain) 32

PARectangleKnots< Tprec > (PARectangular domain (NO detailed documentation)) 35

RBF::MQ< T, Dim > (Generic functor to evaluate Multiquadrics) 37

RBF::MQ< T, 1 > (Multiquadric specialization for 1D) 38

RBF::MQ< T, 2 > (Multiquadric specialization for 2D) 38

RBF::MQ< T, 3 > (Multiquadric specialization for 3D) 39

RBF::MQ_1DX< T, Dim > (Generic functor to evaluate the 1st derivative of MQ with respect to x) 39

RBF::MQ_1DX< T, 1 > (1st derivative of MQ with respect to x for 1D) 39

RBF::MQ_1DX< T, 2 > (1st derivative of MQ with respect to x for 2D) 40

RBF::MQ_1DX< T, 3 > (1st derivative of MQ with respect to x for 3D) 40

RBF::MQ_1DY< T, Dim > (Generic functor to evaluate the 1st derivative of MQ with respect to y) 41

RBF::MQ_1DY< T, 2 > (1st derivative of MQ with respect to y for 2D) 41

RBF::MQ_1DY< T, 3 > (1st derivative of MQ with respect to y for 3D) 42

RBF::MQ_1DZ< T, Dim > (Generic functor to evaluate the 1st derivative of MQ with respect to z) 42

RBF::MQ_1DZ< T, 3 > (1st derivative of MQ with respect to z for 3D) 43

RBF::MQ_2DX< T, Dim > (Generic functor to evaluate the 2nd derivative of MQ with respect to x) 43

RBF::MQ_2DX< T, 1 > (2nd derivative of MQ with respect to x for 1D) 43

RBF::MQ_2DX< T, 2 > (2nd derivative of MQ with respect to x for 2D) 44

RBF::MQ_2DX< T, 3 > (2nd derivative of MQ with respect to x for 3D) 44

RBF::MQ_2DY< T, Dim > (Generic functor to evaluate the 2nd derivative of MQ with respect to y) 45

RBF::MQ_2DY< T, 2 > (2nd derivative of MQ with respect to y for 2D) 45

RBF::MQ_2DY< T, 3 > (2nd derivative of MQ with respect to y for 3D) 46

RBF::MQ_2DZ< T, Dim > (Generic functor to evaluate the 2nd derivative of MQ with respect to z) 46

RBF::MQ_2DZ< T, 3 > (2nd derivative of MQ with respect to z for 3D) 47

RBF::TPS< T, Dim > (Generic functor to evaluate Thin Plate Spline) 47

RBF::TPS< T, 2 > (Thin Plate Spline specialization for 2D) 47

RBF::TPS_1DX< T, Dim > (Generic functor to evaluate the 1st derivative of TPS with respect to x) 48

RBF::TPS_1DX< T, 2 > (1st derivative of TPS with respect to x for 2D) 48

RBF::TPS_1DY< T, Dim > (Generic functor to evaluate the 1st derivative of TPS with respect to y) 49

RBF::TPS_1DY< T, 2 > (1st derivative of TPS with respect to y for 2D) 49

RBF::TPS_2DX< T, Dim > (Generic functor to evaluate the 2nd derivative of TPS with respect to x) 49

RBF::TPS_2DX< T, 2 > (2nd derivative of TPS with respect to x for 2D) 50

RBF::TPS_2DY< T, Dim > (Generic functor to evaluate the 2nd derivative of TPS with respect to y) 50

RBF::TPS_2DY< T, 2 > (2nd derivative of TPS with respect to y for 2D) 51

RectangleKnots< Tprec > (Rectangular domain) 51

RectBlocksKnots< Tprec > (Rectangular domain with blocks near boundaries) 54

SemiCircleKnots< Tprec > (SemiCircle domain) 57

timer (Tool for time measurements (borrowed from FLENS)) 60

5 File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/01TestKnots/box.cpp (Testing the BoxKnots class) 60

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/01TestKnots/line.cpp (Testing the LineKnots class) 61

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/01TestKnots/rectangle.cpp (Testing the RectangleKnots class) 62

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/01TestKnots/rectangle_blocks.cpp (Testing the RectBlocksKnots class) 63

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/01TestKnots/semicircle.cpp (Testing the SemiCircleKnots class) 64

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/02Heat2D/heat2D.cpp (Poisson equation in 2D) 65

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/03HeatSemiCircle/heatsemi.cpp (Laplace equation in a semicircle domain) 66

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/04ConvDiff1D/convdiff01.cpp (Non-stat adv-diff equation in 1D) 67

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/05ConvDiff2D/convdiff2D.cpp (Forced Convection in 2D) 68

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/07LidDriven/lid_-driven2.cpp (Lid Driven Cavity in 2D) 69

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/08BackwardFacingStep/backfacing2.cp (Backward-acing Step in 2D) 70

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/GNUplot.hpp ??

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Traits.hpp (Definition of the variable types used in the whole TUNA::RBF) 75

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/DD/CartComm.hpp ??

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/accessor.hpp 72

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/allocator.hpp 72

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/iterator.hpp 72

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/kdtree.hpp 73

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/node.hpp 74

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/region.hpp 74

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Knots/BoxKnots.hpp ??

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Knots/knot1D.hpp ??

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Knots/knot2D.hpp ??

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Knots/knot3D.hpp ??

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Knots/Knots.hpp ??

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Knots/LineKnots.hpp ??

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Knots/OVRectangleKnots.hpp ??

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Knots/PARectangleKnots.hpp ??

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Knots/RectangleKnots.hpp ??

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Knots/RectBlocksKnots.hpp **??**

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Knots/SemiCircleKnots.hpp **??**

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/RBF/MQ.hpp **??**

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/RBF/RBF.hpp **??**

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/RBF/TPS.hpp **??**

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Solvers/Gauss.hpp **??**

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Solvers/Krylov/acbfprec.hpp **??**

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Solvers/Krylov/diagonalprec.hpp **??**

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Solvers/Krylov/gmres.hpp **(GMRES IMPLEMENTATIONS)** **74**

/home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Solvers/Krylov/identityprec.hpp **??**

6 Namespace Documentation

6.1 flens Namespace Reference

GMRES, preconditioners, based on FLENS.

Classes

- class ACBFPrec

  ACBF Preconditioner.

- class DiagonalPrec

  Diagonal Preconditioner.

- struct gmres_

  Type definition for GMRES.

- struct gmres_< DenseVector< I > >

  Dense Vector specialization.

- class IdentityPrec

  Identity Preconditioner.

Functions

- template<class Mat , class TD >
  void mv (double alpha, const ACBFPrec< Mat, TD > &ACBF, const DenseVector< Array< double > > &b, double beta, DenseVector< Array< double > > &x)

  Multiplication definition for the ACBF preconditioner.

- template<class Mat >
  void mv (double alpha, const DiagonalPrec< Mat > &J, const DenseVector< Array< double > > &b, double beta, DenseVector< Array< double > > &x)

  Multiplication definition for the Diagonal preconditioner.

- template<class Mat , class Vec >
  int gmres (Mat const &A, Vec &x, const Vec &b, unsigned int const maxiter, double const tol)

  GMRES NON PRECONDITIONED.

- template<class Mat , class Vec , class Precond >
  int gmres (Mat const &A, Vec &x, const Vec &b, const Precond &M, unsigned int const maxiter, double const tol)

  GMRES PRECONDITIONED.

- template<class Mat >
  void mv (double alpha, const IdentityPrec< Mat > &J, const DenseVector< Array< double > > &b, double beta, DenseVector< Array< double > > &x)

  Multiplication definition for the Identity preconditioner.

6.1.1 Detailed Description

GMRES, preconditioners, based on FLENS.

6.1.2 Function Documentation

6.1.2.1 template<class Mat , class TD > void flens::mv (double *alpha*, const ACBFPrec< Mat, TD > & *ACBF*, const DenseVector< Array< double > > & *b*, double *beta*, DenseVector< Array< double > > & *x*) [inline]

Multiplication definition for the ACBF preconditioner.

Here the multiplaction `alpha * J * b + beta * x` is defined. In this case `alpha` = 1 and `beta` = 0, so we do `x = J * b` .

Parameters:

alpha scalar.

J ACBF preconditioner.

b vector.

beta scalar.

x vector where the result is stored.

Author:

Luis M. de la Cruz [Tue Dec 11 12:54:41 GMT 2007]

Definition at line 167 of file acbfprec.hpp.

6.1.2.2 template<class Mat > void flens::mv (double *alpha*, const DiagonalPrec< Mat > & *J*, const DenseVector< Array< double > > & *b*, double *beta*, DenseVector< Array< double > > & *x*) [inline]

Multiplication definition for the Diagonal preconditioner.

Here the multiplaction `alpha * J * b + beta * x` is defined. In this case `alpha` = 1 and `beta` = 0, so we do `x = J * b`, but because this operation is intended for the Diagonal preconditioner, we finally do `x = b / diag(A)`.

Parameters:

alpha scalar.

J Diagonal preconditioner.

b vector.

beta scalar.

x vector where the result is stored.

Author:

Luis M. de la Cruz [Wed Nov 28 10:06:43 GMT 2007]

Definition at line 71 of file diagonalprec.hpp.

6.1.2.3 template<class Mat , class Vec > int flens::gmres (Mat const & *A*, Vec & *x*, const Vec & *b*, unsigned int const *maxiter*, double const *tol*) [inline]

GMRES NON PRECONDITIONED.

Author:

Luis M. de la Cruz [Mon Nov 12 11:52:45 GMT 2007]

Parameters:

A coefficients matrix.

x solution vector.

b right-hand vector.

maxiter maximum number or GMRES iterations.

tol tolerance.

Returns:

number of GMRES iterations to achieve the prescribed tolerance.

Definition at line 80 of file gmres.hpp.

6.1.2.4 template<class Mat , class Vec , class Precond > int flens::gmres (Mat const & *A*, Vec & *x*, const Vec & *b*, const Precond & *M*, unsigned int const *maxiter*, double const *tol*) [inline]

GMRES PRECONDITIONED.

Author:

Luis M. de la Cruz [Wed Nov 28 10:10:09 GMT 2007]

Parameters:

A coefficients matrix.

x solution vector.

b right-hand vector.

M preconditioner.

maxiter maximum number or GMRES iterations.

tol tolerance.

Returns:

number of GMRES iterations to achieve the prescribed tolerance.

Definition at line 187 of file gmres.hpp.

6.1.2.5 template<class Mat > void flens::mv (double *alpha*, const IdentityPrec< Mat > & *J*, const DenseVector< Array< double > > & *b*, double *beta*, DenseVector< Array< double > > & *x*) [inline]

Multiplication definition for the Identity preconditioner.

Here the multiplaction `alpha * J * b + beta * x` is defined. In this case `alpha` = 1 and `beta` = 0, so we do `x = J * b`, but because this operation is intended for the Identity preconditioner, we finally do `x = b`.

Parameters:

alpha scalar.

J Identity preconditioner.

b vector.

beta scalar.

x vector where the result is stored.

Author:

Luis M. de la Cruz [Wed Nov 28 10:06:43 GMT 2007]

Definition at line 67 of file identityprec.hpp.

6.2 KDTree Namespace Reference

Contains all the stuff for KDTree algorithm.

6.2.1 Detailed Description

Contains all the stuff for KDTree algorithm.

6.3 RBF Namespace Reference

Implements several RBFs.

Classes

- class MQ
  Generic functor to evaluate Multiquadrics.
- class MQ< T, 1 >
  Multiquadric specialization for 1D.
- class MQ< T, 2 >
  Multiquadric specialization for 2D.
- class MQ< T, 3 >
  Multiquadric specialization for 3D.
- class MQ_1DX
  Generic functor to evaluate the 1st derivative of MQ with respect to x.
- class MQ_1DX< T, 1 >
  1st derivative of MQ with respect to x for 1D.
- class MQ_1DX< T, 2 >
  1st derivative of MQ with respect to x for 2D.
- class MQ_1DX< T, 3 >
  1st derivative of MQ with respect to x for 3D.
- class MQ_2DX
  Generic functor to evaluate the 2nd derivative of MQ with respect to x.
- class MQ_2DX< T, 1 >
  2nd derivative of MQ with respect to x for 1D.
- class MQ_2DX< T, 2 >
  2nd derivative of MQ with respect to x for 2D.
- class MQ_2DX< T, 3 >
  2nd derivative of MQ with respect to x for 3D.

- class MQ_1DY

  Generic functor to evaluate the 1st derivative of MQ with respect to y.

- class MQ_1DY< T, 2 >

  1st derivative of MQ with respect to y for 2D.

- class MQ_1DY< T, 3 >

  1st derivative of MQ with respect to y for 3D.

- class MQ_2DY

  Generic functor to evaluate the 2nd derivative of MQ with respect to y.

- class MQ_2DY< T, 2 >

  2nd derivative of MQ with respect to y for 2D.

- class MQ_2DY< T, 3 >

  2nd derivative of MQ with respect to y for 3D.

- class MQ_1DZ

  Generic functor to evaluate the 1st derivative of MQ with respect to z.

- class MQ_1DZ< T, 3 >

  1st derivative of MQ with respect to z for 3D.

- class MQ_2DZ

  Generic functor to evaluate the 2nd derivative of MQ with respect to z.

- class MQ_2DZ< T, 3 >

  2nd derivative of MQ with respect to z for 3D.

- class TPS

  Generic functor to evaluate Thin Plate Spline.

- class TPS< T, 2 >

  Thin Plate Spline specialization for 2D.

- class TPS_1DX

  Generic functor to evaluate the 1st derivative of TPS with respect to x.

- class TPS_1DX< T, 2 >

  1st derivative of TPS with respect to x for 2D.

- class TPS_1DY

  Generic functor to evaluate the 1st derivative of TPS with respect to y.

- class TPS_1DY< T, 2 >

  1st derivative of TPS with respect to y for 2D.

- class TPS_2DX

Generic functor to evaluate the 2nd derivative of TPS with respect to x.

- class TPS_2DX< T, 2 >

  2nd derivative of TPS with respect to x for 2D.

- class TPS_2DY

  Generic functor to evaluate the 2nd derivative of TPS with respect to y.

- class TPS_2DY< T, 2 >

  2nd derivative of TPS with respect to y for 2D.

Functions

- template<typename T_RBF , typename Tprec >
  Tprec eval (Tprec x, Tprec y, Tprec z, const Vec &xv, const Vec &yv, const Vec &zv, const Vec &lambda, T_RBF f, int NPol=0)

  Function to evaluate the solution using an RBF in 3D.

- template<typename T_RBF , typename Tprec >
  Tprec eval (Tprec x, Tprec y, const Vec &xv, const Vec &yv, const Vec &lambda, T_RBF f, int NPol=0)

  Function to evaluate the solution using an RBF in 2D.

- template<typename T_RBF , typename Tprec >
  Tprec eval (Tprec x, const Vec &xv, const Vec &lambda, T_RBF f)

  Function to evaluate the solution using an RBF in 1D.

6.3.1 Detailed Description

Implements several RBFs.

This namespace contains implementations to evaluate several Radial Basis Functions and its derivatives.

Author:

Luis M. de la Cruz [Mon Nov 12 11:52:45 GMT 2007]

6.3.2 Function Documentation

6.3.2.1 template<typename T_RBF , typename Tprec > Tprec RBF::eval (Tprec *x*, Tprec *y*, Tprec *z*, const Vec & *xv*, const Vec & *yv*, const Vec & *zv*, const Vec & *lambda*, T_RBF *f*, int *NPol* = 0) [inline]

Function to evaluate the solution using an RBF in 3D.

Parameters:

x coordinate where the solution will be evaluated.

y coordinate where the solution will be evaluated.

z coordinate where the solution will be evaluated.

xv vector of x-coords where the RBF was calculated.

yv vector of y-coords where the RBF was calculated.

zv vector of z-coords where the RBF was calculated.

lambda vector of coefficients for the RBF.

NPol 0 -> not polynomial (default), 6 -> using a degree 2 pol. (just for testing, could be eliminated...)

Definition at line 50 of file RBF.hpp.

6.3.2.2 template<typename T_RBF , typename Tprec > Tprec RBF::eval (Tprec *x*, Tprec *y*, const Vec & *xv*, const Vec & *yv*, const Vec & *lambda*, T_RBF *f*, int *NPol* = 0) [inline]

Function to evaluate the solution using an RBF in 2D.

Parameters:

x coordinate where the solution will be evaluated.

y coordinate where the solution will be evaluated.

xv vector of x-coords where the RBF was calculated.

yv vector of y-coords where the RBF was calculated.

lambda vector of coefficients for the RBF.

NPol 0 -> not polynomial (default), 6 -> using a degree 2 pol. (just for testing, could be eliminated...)

Definition at line 85 of file RBF.hpp.

6.3.2.3 template<typename T_RBF , typename Tprec > Tprec RBF::eval (Tprec *x*, const Vec & *xv*, const Vec & *lambda*, T_RBF *f*) [inline]

Function to evaluate the solution using an RBF in 1D.

Parameters:

x coordinate where the solution will be evaluated.

xv vector of x-coords where the RBF was calculated.

lambda vector of coefficients for the RBF.

Definition at line 115 of file RBF.hpp.

6.4 Solver Namespace Reference

Linear system solvers.

Functions

- template<typename T >
  Vec Gauss (Mat &A, Vec &b)

  Gauss elimination with partial pivoting algorithm for dense linear systems.

6.4.1 Detailed Description

Linear system solvers.

This namespace contains implementations for solving general linear systems.

Author:

Luis M. de la Cruz [Tue Sep 11 17:06:45 BST 2007]

6.4.2 Function Documentation

6.4.2.1 template<typename T > Vec Solver::Gauss (Mat & *A*, Vec & *b*) [inline]

Gauss elimination with partial pivoting algorithm for dense linear systems.

Author:

Daniel Cervantes & Luis M. de la Cruz [Tue Sep 11 17:06:45 BST 2007]

Parameters:

A matrix of coefficients.

b right term vector.

Returns:

the x vector solution of the linear system A ∗ x = b.

Todo

Check the matrix access, remember that I'm using ColMajor for all the matrices (see FLENS manual).

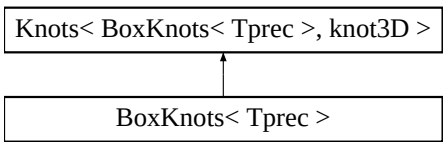
Definition at line 49 of file Gauss.hpp.

7 Class Documentation

7.1 BoxKnots< Tprec > Class Template Reference

Octoedro or Cuboide.

Inheritance diagram for BoxKnots< Tprec >::



Public Member Functions

- BoxKnots ()
  Default constructor.
- BoxKnots (prec_t l_x, int n_x, prec_t l_y, int n_y, prec_t l_z, int n_z)
  Constructor to generate knots in the centers of the "cells" of a gridded mesh.
- prec_t getDx ()
  Return the size of the mesh in x-axis.
- prec_t getDy ()
  Return the size of the mesh in y-axis.
- prec_t getDz ()
  Return the size of the mesh in z-axis.
- prec_t getLx ()
  Return the length of the box in x-axis.
- prec_t getLy ()
  Return the length of the box in y-axis.
- prec_t getLz ()
  Return the length of the box in z-axis.
- prec_t getNx ()
  Return the number of points in x-axis.
- prec_t getNy ()
  Return the number of points in y-axis.
- prec_t getNz ()
  Return the number of points in z-axis.
- bool constructKnots ()
  Construct all the knots of the domain.
- bool addSpecialPoints (iMat &, int, int)
  Add special points for the ACBF aproximation (is not defined yet).
- bool print ()
  Print info to the standard output (screen).

Private Attributes

- prec_t hx

  Horizontal length of the box.

- prec_t hy

  Vertical length of the box.

- prec_t hz

  Depth length of the box.

- int Nx

  Number of points on x-axis.

- int Ny

  Number of points on y-axis.

- int Nz

  Number of points on z-axis.

- prec_t ep

  Randomness of the points distribution (default ep = 0).

- int cel

  Used to keep uniform knots in one cell near the boundary (default cel = 0).

7.1.1 Detailed Description

template<typename Tprec> class BoxKnots< Tprec >

Octoedro or Cuboide.

This class is aimed to generate knots inside of 3D rectangular domains. The function constructKnots() must be reimplemented here to generate the knots on the rectangular domain.

Author:

Luis M. de la Cruz [Mon Nov 12 14:53:34 GMT 2007]

Definition at line 39 of file BoxKnots.hpp.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 template<typename Tprec> BoxKnots< Tprec >::BoxKnots (prec_t *l_x*, int *n_x*, prec_t *l_y*, int *n_y*, prec_t *l_z*, int *n_z*) [inline]

Constructor to generate knots in the centers of the "cells" of a gridded mesh.

It allows some randomness of the points.

Parameters:

l_x Length of the box in x-axis direction.

l_y Length of the box in y-axis direction.

l_z Length of the box in z-axis direction.

n_x Number of points in x-axis direction.

n_y Number of points in y-axis direction.

n_z Number of points in z-axis direction.

Definition at line 81 of file BoxKnots.hpp.

7.1.3 Member Function Documentation

7.1.3.1 template<typename prec_t > bool BoxKnots< prec_t >::constructKnots () [inline]

Construct all the knots of the domain.

Detailed documentation is missing, it is coming soon.

Reimplemented from Knots< BoxKnots< Tprec >, knot3D >.

Definition at line 173 of file BoxKnots.hpp.

7.2 CartComm Class Reference

2D Catersian communicator in the MPI environment.

Public Member Functions

- CartComm (int argc, char **argv, int r)
  The Cartesian topology is constructed here.

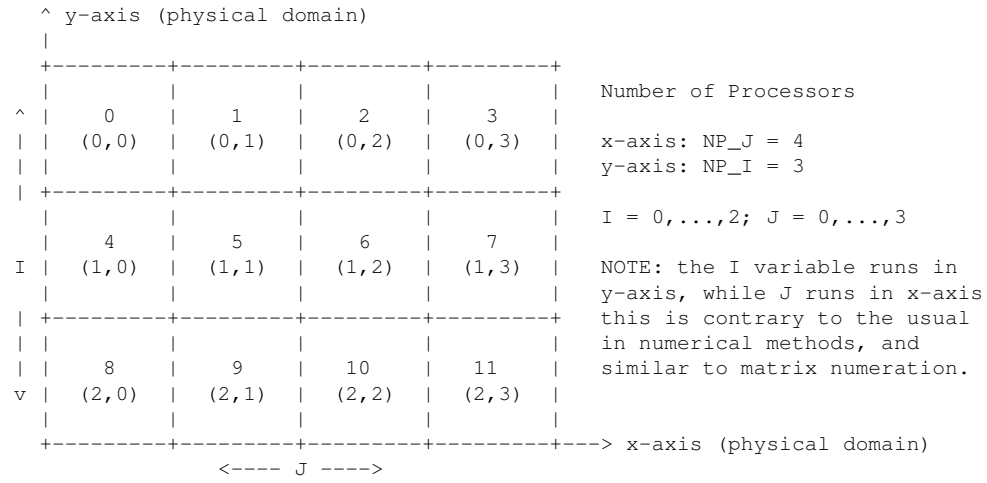
Public Attributes

- MPI::Cartcomm comm
  Communicator for a Cartesian topology.

7.2.1 Detailed Description

2D Catersian communicator in the MPI environment.

This class is used to construct a Cartesian topology to be used in simple rectangular-like domains. It is supposed all subdomains that belongs to this topology are rectangles as well. The `comm` variable is an MPI::Cartcomm communicator, and to be able to access all its functionality from outside the class in an easy way, this variable is left public. The numeration of the processors in the topology is as follows:



Todo

Extend the functionality of CartComm class to 1D and 3D (templates)

Author:

Luis M. de la Cruz [Tue Mar 18 11:10:23 GMT 2008]

Definition at line 62 of file CartComm.hpp.

7.3 flens::ACBFPrec< MA, TD > Class Template Reference

ACBF Preconditioner.

Public Member Functions

- bool construct (TD domain, int max_knots, int sp=0, bool print_screen=false)

  This function constructs the preconditioner as defined in Kansa et al.

Public Attributes

- const MA & A

  Coefficient matrix.

- iMat P

  Index matrix.

- MA W

  Preconditioner coefficients.

7.3.1 Detailed Description

template<typename MA, typename TD> class flens::ACBFPrec< MA, TD >

ACBF Preconditioner.

Author:

Luis M. de la Cruz [Tue Dec 11 12:54:41 GMT 2007]

Definition at line 47 of file acbfprec.hpp.

7.3.2 Member Function Documentation

7.3.2.1 template<typename MA, typename TD> bool flens::ACBFPrec< MA, TD >::construct (TD *domain*, int *max_knots*, int *sp* = 0, bool *print_screen* = false) [inline]

This function constructs the preconditioner as defined in Kansa et al.

[Adv. Comp. Math., 2005, 23, (1-2), 31-54].

Todo

Optimize the Matrix-Matrix multiplication

Todo

Optimize this process (L-QR-SVD), basically I need to incorporate the algorithm described in Kansa & Ling, Adv. Comp. Math. 2005, 23: 31-54, pag 39.

Definition at line 57 of file acbfprec.hpp.

7.4 flens::DiagonalPrec< MA > Class Template Reference

Diagonal Preconditioner.

Public Attributes

- const MA & A

  Coefficient matrix.

- const double EPS

  almost zero.

7.4.1 Detailed Description

template<typename MA> class flens::DiagonalPrec< MA >

Diagonal Preconditioner.

Author:

Luis M. de la Cruz [Wed Nov 28 10:06:43 GMT 2007]

Definition at line 45 of file diagonalprec.hpp.

7.5 flens::gmres_< A > Struct Template Reference

Type definition for GMRES.

7.5.1 Detailed Description

template<typename A> struct flens::gmres_< A >

Type definition for GMRES.

Author:

Luis M. de la Cruz [Wed Nov 28 10:06:43 GMT 2007]

Definition at line 49 of file gmres.hpp.

7.6 flens::gmres_< DenseVector< I > > Struct Template Reference

Dense Vector specialization.

7.6.1 Detailed Description

template<typename I> struct flens::gmres_< DenseVector< I > >

Dense Vector specialization.

Author:

Luis M. de la Cruz [Wed Nov 28 10:06:43 GMT 2007]

Definition at line 61 of file gmres.hpp.

7.7 flens::IdentityPrec< MA > Class Template Reference

Identity Preconditioner.

7.7.1 Detailed Description

template<typename MA> class flens::IdentityPrec< MA >

Identity Preconditioner.

Author:

Luis M. de la Cruz [Wed Nov 28 10:06:43 GMT 2007]

Definition at line 45 of file identityprec.hpp.

7.8 knot1D Struct Reference

The leaves of the KDTree in 1D.

Public Member Functions

- prec_t operator[ ] (size_t const N) const

  Overloaded [ ] operator required by kdtree++ to access the individual dimensional components.

Public Attributes

- prec_t coord [1]

  (x,y) knots coordinates in 1D.

- int index

  index inside the global Matrix, used for the ACBFprecond.

7.8.1 Detailed Description

The leaves of the KDTree in 1D.

Objects of this structure are used by kdtree++ as the leaves of the complete KDTree. It requires some particular behavior, explained in `http://libkdtree.alioth.debian.org/`

Author:

Luis M. de la Cruz [Mon Dec 17 10:30:21 GMT 2007]

Definition at line 38 of file knot1D.hpp.

7.8.2 Member Function Documentation

7.8.2.1 prec_t knot1D::operator[ ] (size_t const *N*) const `[inline]`

Overloaded [ ] operator required by kdtree++ to access the individual dimensional components.

Parameters:

N the dimensional component.

Definition at line 47 of file knot1D.hpp.

7.9 knot2D Struct Reference

The leaves of the KDTree in 2D.

Public Member Functions

- prec_t operator[ ] (size_t const N) const

  Overloaded [ ] operator required by kdtree++ to access the individual dimensional components.

Public Attributes

- prec_t coord [2]

  (x,y) knots coordinates in 2D.

- int index

  index inside the global Matrix, used for the ACBFprecond.

7.9.1 Detailed Description

The leaves of the KDTree in 2D.

Objects of this structure are used by kdtree++ as the leaves of the complete KDTree. It requires some particular behavior, explained in http://libkdtree.alioth.debian.org/

Author:

Luis M. de la Cruz [Mon Dec 17 10:30:21 GMT 2007]

Definition at line 38 of file knot2D.hpp.

7.9.2 Member Function Documentation

7.9.2.1 prec_t knot2D::operator[ ] (size_t const *N*) const [inline]

Overloaded [ ] operator required by kdtree++ to access the individual dimensional components.

Parameters:

N the dimensional component.

Definition at line 47 of file knot2D.hpp.

7.10 knot3D Struct Reference

The leaves of the KDTree in 3D.

Public Member Functions

- prec_t operator[ ] (size_t const N) const

  Overloaded [ ] operator required by kdtree++ to access the individual dimensional components.

Public Attributes

- prec_t coord [3]

  (x,y,z) knots coordinates in 3D.

- int index

  index inside the global Matrix, used for the ACBFprecond.

7.10.1 Detailed Description

The leaves of the KDTree in 3D.

Objects of this structure are used by kdtree++ as the leaves of the complete KDTree. It requires some particular behavior, explained in http://libkdtree.alioth.debian.org/

Author:

Luis M. de la Cruz [Mon Dec 17 10:30:21 GMT 2007]

Definition at line 38 of file knot3D.hpp.

7.10.2 Member Function Documentation

7.10.2.1 prec_t knot3D::operator[ ] (size_t const *N*) const [inline]

Overloaded [ ] operator required by kdtree++ to access the individual dimensional components.

Parameters:

N the dimensional component.

Definition at line 46 of file knot3D.hpp.

7.11 Knots< Tdomain, Tknot > Class Template Reference

Base class for knots generation.

Public Member Functions

- Tdomain & asDerived ()

  Curiosly Recursive Template Pattern (CRTP).

- bool constructKnots ()

  Delegate responsabilities to the derived classes.

- bool calcKDTree ()

  KDTree construction.

- const std::vector< Tknot > & findNeighbors (const prec_t x, const prec_t y, const prec_t range)

  Find the neighbors points of a defined knot inside a range (2D).

- const std::vector< Tknot > & findNeighbors (const prec_t x, const prec_t range)

  Find the neighbors points of a defined knot inside a range (1D).

- const std::vector< Tknot > & findNeighbors (const prec_t x, const prec_t y, const prec_t z, const prec_t range)

  Find the neighbors points of a defined knot inside a range (3D).

- bool findNeighbors (int j, iMat &P, int max_knots, int sp=0)

  Find the neighbors to all knots in the domain, this information is used by the ACBF preconditioner (acbf-precond).

- bool setN (int n)

  Set the total number of knots `N`.

- bool setNI (int ni)

  Set the number of interior knots `NI`.

- bool setNB (int nb)

  Set the number of knots on the boundary `NB`.

- Vec getKnots (axis_t n)

  Get the Knots coordinates.

- int getTotalKnots ()

  Get the total number of knots.

- int getInteriorKnots ()

  Get the number of interior knots.

- int getBoundaryKnots ()

  Get the number of knots on the boundary.

- bool readFromFile (std::string name)

  Read the knots from a file, `N` *must be defined before, actually it must be defined in the constructor of derived classes.*

- bool writeToFile (std::string name)

  Write the knots to a file, `N` *must be defined before, actually it must be defined in the constructor of derived classes.*

- bool writeToFile (std::string name, int I, int J)

  Write the knots to files defined by `I` *and* `J` *coordinates of a Cartesian topology in the MPI context,* `N` *must be defined before, actually it must be defined in the constructor of derived classes.*

- bool print ()

  Print info about the knots to the standard output (screen).

Protected Attributes

- int N

  Total number of knots.

- int NI

  Iternal knots.

- int NB

  knots on the boundary.

- int Dim

  Dimension of the problem (1,2 or 3).

- Tknot * xyz

  Coordinates array.

- prec_t max_upper_range

  Every derived class must to define this bound.

7.11.1 Detailed Description

template<typename Tdomain, typename Tknot> class Knots< Tdomain, Tknot >

Base class for knots generation.

This class contains the definition of general variables and methods to be used inside more particular classes. The template parameters are: `Tdomain` type of the domain of the derived class. `Tknot` type of the knot (knot2D or knot3D).

Author:

Luis M. de la Cruz [Mon Dec 17 11:38:56 GMT 2007]

Todo

To find a way to generate knot for 1D, 2D or 3D without -D_1D_, -D_2D_ and -D_3D_ compiler options.

Definition at line 57 of file Knots.hpp.

7.11.2 Member Function Documentation

7.11.2.1 template<typename Tdomain, typename Tknot> Tdomain& Knots< Tdomain, Tknot >::asDerived () [inline]

Curiosly Recursive Template Pattern (CRTP).

Here we use the Curiosly Recursive Template Pattern (CRTP also known as the Barton-Nackman trick), to delegate responsabilities to derived classes. The idea is to get the actual type of derived classes inside

of the base class. With this trick, the compiler is able to define the correct function that must be called when an instance of the derived class calls to the general function, defined in the base class. This is also a way to construct static polymorphism, to avoid the dynamic one and get better performance. See the C++ Templates - The Complete Guide by David Vandevoorde and Nicolai M. Josuttis, Chapter 11, pag. 174.

Definition at line 82 of file Knots.hpp.

7.11.2.2 template<typename Tdomain, typename Tknot> bool Knots< Tdomain, Tknot >::constructKnots () [inline]

Delegate responsabilities to the derived classes.

The responsability of knots construction is delegated to the subclasses. Every derived class must to define (reimplement) this function, and every such class has the freedom of defining the way the knots are constructed.

Reimplemented in BoxKnots< Tprec >, LineKnots< Tprec >, OVRectangleKnots< Tprec >, PARectangleKnots< Tprec >, RectangleKnots< Tprec >, RectBlocksKnots< Tprec >, and SemiCircleKnots< Tprec >.

Definition at line 90 of file Knots.hpp.

7.11.2.3 template<typename Tdomain, typename Tknot> bool Knots< Tdomain, Tknot >::calcKDTree () [inline]

KDTree construction.

The knots are stored in the `tree` variable defined in knot2D or knot3D, depending on the dimension of the problem.

Definition at line 96 of file Knots.hpp.

7.11.2.4 template<typename Tdomain, typename Tknot> const std::vector<Tknot>& Knots< Tdomain, Tknot >::findNeighbors (const prec_t *x*, const prec_t *y*, const prec_t *range*) [inline]

Find the neighbors points of a defined knot inside a range (2D).

Parameters:

x x-axis coordinate of the knot.

y y-axis coordinate of the knot.

range the size of the neighborhood where the neighbors live.

Returns:

`neighbors_array` will contain the list of neighbors (it is a std::vector<Tknot>).

Definition at line 116 of file Knots.hpp.

7.11.2.5 template<typename Tdomain, typename Tknot> const std::vector<Tknot>& Knots< Tdomain, Tknot >::findNeighbors (const prec_t *x*, const prec_t *range*) [inline]

Find the neighbors points of a defined knot inside a range (1D).

Parameters:

x x-axis coordinate of the knot.

range the size of the neighborhood where the neighbors live.

Returns:

`neighbors_array` will contain the list of neighbors (it is a std::vector<Tknot>).

Definition at line 138 of file Knots.hpp.

7.11.2.6 template<typename Tdomain, typename Tknot> const std::vector<Tknot>& Knots< Tdomain, Tknot >::findNeighbors (const prec_t *x*, const prec_t *y*, const prec_t *z*, const prec_t *range*) [inline]

Find the neighbors points of a defined knot inside a range (3D).

Parameters:

x x-axis coordinate of the knot.

y y-axis coordinate of the knot.

z z-axis coordinate of the knot.

range the size of the neighborhood where the neighbors live.

Returns:

`neighbors_array` will contain the list of neighbors (it is a std::vector<Tknot>).

Definition at line 159 of file Knots.hpp.

7.11.2.7 template<typename Tdomain, typename Tknot> bool Knots< Tdomain, Tknot >::findNeighbors (int *j*, iMat & *P*, int *max_knots*, int *sp* = 0) [inline]

Find the neighbors to all knots in the domain, this information is used by the ACBF preconditioner (acbf-precond).

This works for 1D, 2D and 3D.

Parameters:

j index of the working knot.

P Integer Matrix containing the indices of neighbors points for every knot.

max_knots Maximum number of knots allowed inside the range.

Returns:

Definition at line 184 of file Knots.hpp.

7.11.2.8 template<typename Tdomain, typename Tknot> Vec Knots< Tdomain, Tknot >::getKnots (axis_t *n*) [inline]

Get the Knots coordinates.

Parameters:

n is of type axis_t, this argument can be X, Y or Z.

Returns:

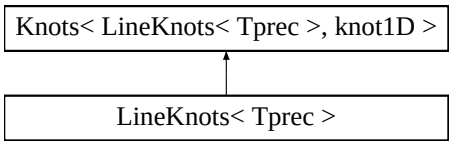
coord wll contain the n-axis coordinates of all knots.

Definition at line 282 of file Knots.hpp.

7.12 LineKnots< Tprec > Class Template Reference

Knots in a line.

Inheritance diagram for LineKnots< Tprec >::



Public Member Functions

- LineKnots ()
  Default constructor.
- LineKnots (prec_t l_x, int n_x)
  Constructor to generate the knots.
- bool constructKnots ()
  Construct all the knots of the domain.
- bool addSpecialPoints (iMat &, int, int)
  Add special points for the ACBF aproximation.
- bool print ()
  Print info to the standard output (screen).

Private Attributes

- prec_t lx

  Horizontal length of the line.

- int Nx

  Number of points on x-axis.

7.12.1 Detailed Description

template<typename Tprec> class LineKnots< Tprec >

Knots in a line.

This class is aimed to generate knots along the x-axis. The function constructKnots() must be reimplemented here to generate the knots.

Author:

Luis M. de la Cruz [Mon Nov 12 14:53:34 GMT 2007]

Definition at line 39 of file LineKnots.hpp.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 template<typename Tprec> LineKnots< Tprec >::LineKnots (prec_t *l_x*, int *n_x*) [inline]

Constructor to generate the knots.

Parameters:

l_x Length of the line.

n_x Number of points in the line.

Definition at line 70 of file LineKnots.hpp.

7.12.3 Member Function Documentation

7.12.3.1 template<typename prec_t > bool LineKnots< prec_t >::constructKnots () [inline]

Construct all the knots of the domain.

Construct regularly spaced knots on a line of length lx;.

Reimplemented from Knots< LineKnots< Tprec >, knot1D >.

Definition at line 112 of file LineKnots.hpp.

7.12.3.2 template<typename prec_t > bool LineKnots< prec_t >::addSpecialPoints (iMat & *P*, int *j*, int *nsp*) [inline]

Add special points for the ACBF aproximation.

Add special points located in the boundary of the line.

Parameters:

P indices of neighbors knots.

j actual knot.

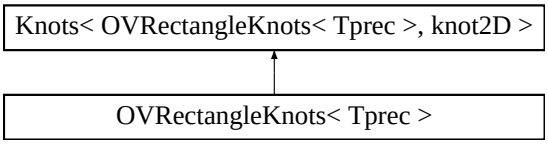
nsp number of special points.

Definition at line 137 of file LineKnots.hpp.

7.13 OVRectangleKnots< Tprec > Class Template Reference

OVRectangular domain.

Inheritance diagram for OVRectangleKnots< Tprec >::



Public Member Functions

- OVRectangleKnots ()
  Default constructor.
- OVRectangleKnots (prec_t l_x, int n_x, prec_t l_y, int n_y, prec_t g_x, prec_t g_y, CartComm &cart, random_t knd=U)
  Constructor to generate knots in the centers of the "cells" of a gridded mesh.
- bool constructKnots ()
  Construct all the knots of the domain.
- bool addSpecialPoints (iMat &, int, int)
  Add special points for the ACBF aproximation.
- void setRandomness (prec_t e)
  Random parameter.
- bool gridKnots ()
  Generate knots in a "Cartesian grid".
- bool print ()
  Print info to the standard output (screen).

Private Attributes

- prec_t lx

  Horizontal length of the rectangle.

- prec_t ly

  Vertical length of the rectangle.

- int Nx

  Number of points on x-axis.

- int Ny

  Number of points on y-axis.

- random_t knots_dist

  kind of point distribution

- prec_t epru

  Randomness of the points distribution (default epru = 0).

7.13.1 Detailed Description

template<typename Tprec> class OVRectangleKnots< Tprec >

OVRectangular domain.

This class is aimed to generate knots inside overlapping rectangular domains used by Domain Decomposition algorithms. The function constructKnots() must be reimplemented here to generate the knots on the rectangular domain.

Author:

Luis M. de la Cruz [Mon Nov 12 14:53:34 GMT 2007]

Definition at line 44 of file OVRectangleKnots.hpp.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 template<typename Tprec > OVRectangleKnots< Tprec >::OVRectangleKnots (prec_t *l_x*, int *n_x*, prec_t *l_y*, int *n_y*, prec_t *g_x*, prec_t *g_y*, CartComm & *cart*, random_t *knd* = U) [inline]

Constructor to generate knots in the centers of the "cells" of a gridded mesh.

It allows some randomness of the points.

Parameters:

l_x Global length of the rectangle in x-axis direction.

l_y Global length of the rectangle in y-axis direction.

n_x Local number of points in x-axis

n_y Local number of points in y-axis direction.

g_x overlapping in x-axis.

g_y overlapping in y-axis.

cart Communicator between processors (CartComm).

knd kind of distribution for the points.

Definition at line 91 of file OVRectangleKnots.hpp.

7.13.3 Member Function Documentation

7.13.3.1 template<typename prec_t > bool OVRectangleKnots< prec_t >::constructKnots () [inline]

Construct all the knots of the domain.

Construction of the knots.

Reimplemented from Knots< OVRectangleKnots< Tprec >, knot2D >.

Definition at line 230 of file OVRectangleKnots.hpp.

7.13.3.2 template<typename prec_t > bool OVRectangleKnots< prec_t >::addSpecialPoints (iMat & *P*, int *j*, int *nsp*) [inline]

Add special points for the ACBF aproximation.

Add special points located in the boundary of the rectangle.

Parameters:

P indices of neighbors knots.

j actual knot.

nsp number of special points.

Definition at line 411 of file OVRectangleKnots.hpp.

7.13.3.3 template<typename prec_t > bool OVRectangleKnots< prec_t >::gridKnots () [inline]

Generate knots in a "Cartesian grid".

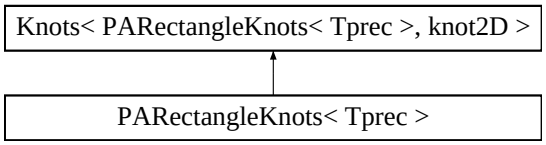
This function is aimed to generate the knots in a Cartesian uniform grid.

Definition at line 526 of file OVRectangleKnots.hpp.

7.14 PARectangleKnots< Tprec > Class Template Reference

PARectangular domain (NO detailed documentation).

Inheritance diagram for PARectangleKnots< Tprec >::



Public Member Functions

- PARectangleKnots ()
  Default constructor.
- PARectangleKnots (prec_t l_x, int n_x, prec_t l_y, int n_y, CartComm &cart, random_t knd=U)
  Constructor to generate knots in the centers of the "cells" of a gridded mesh.
- bool constructKnots ()
  Construct all the knots of the domain.
- bool addSpecialPoints (iMat &, int, int)
  Add special points for the ACBF aproximation.
- void setRandomness (prec_t e)
  Random parameter.
- bool gridKnots ()
  Generate knots in a "Cartesian grid".
- bool print ()
  Print info to the standard output (screen).

Private Attributes

- prec_t lx
  Horizontal length of the rectangle.
- prec_t ly
  Vertical length of the rectangle.
- int Nx
  Number of points on x-axis.
- int Ny
  Number of points on y-axis.

- prec_t epru

  Randomness of the points distribution (default ep = 0).

7.14.1 Detailed Description

template<typename Tprec> class PARectangleKnots< Tprec >

PARectangular domain (NO detailed documentation).

This class is aimed to generate knots inside of rectangular domains. The function constructKnots() must be reimplemented here to generate the knots on the rectangular domain.

Author:

Luis M. de la Cruz [Mon Nov 12 14:53:34 GMT 2007]

Definition at line 40 of file PARectangleKnots.hpp.

7.14.2 Constructor & Destructor Documentation

7.14.2.1 template<typename Tprec > PARectangleKnots< Tprec >::PARectangleKnots (prec_t *l_x*, int *n_x*, prec_t *l_y*, int *n_y*, CartComm & *cart*, random_t *knd* = U) [inline]

Constructor to generate knots in the centers of the "cells" of a gridded mesh.

It allows some randomness of the points.

Parameters:

l_x Length of the rectangle in x-axis direction.

l_y Length of the rectangle in y-axis direction.

n_x Number of points in x-axis direction.

n_y Number of points in y-axis direction.

e Degree of randomness of the points distribution (default ep = 0).

Definition at line 82 of file PARectangleKnots.hpp.

7.14.3 Member Function Documentation

7.14.3.1 template<typename prec_t > bool PARectangleKnots< prec_t >::constructKnots () [inline]

Construct all the knots of the domain.

The way the knots are generated is tricky: Taking the values of Nx and Ny an "invisible grid" is defined inside the rectangle, this grid will contains (Nx - 1) X (Ny - 1) "cells", for each cell a knot is generated and this knot is allowed to be in a random position inside its cell, the randomness depends on the ep parameter.

The order of the knots are as shown in the next figure:

Figure 1: Knots order

Reimplemented from Knots< PARectangleKnots< Tprec >, knot2D >.

Definition at line 211 of file PARectangleKnots.hpp.

7.14.3.2 template<typename prec_t > bool PARectangleKnots< prec_t >::addSpecialPoints (iMat & *P*, int *j*, int *nsp*) [inline]

Add special points for the ACBF aproximation.

Add special points located in the boundary of the rectangle.

Parameters:

P indices of neighbors knots.

j actual knot.

nsp number of special points.

Definition at line 343 of file PARectangleKnots.hpp.

7.14.3.3 template<typename prec_t > bool PARectangleKnots< prec_t >::gridKnots () [inline]

Generate knots in a "Cartesian grid".

This function is aimed to generate the knots in a Cartesian uniform grid.

Definition at line 458 of file PARectangleKnots.hpp.

7.15 RBF::MQ< T, Dim > Class Template Reference

Generic functor to evaluate Multiquadrics.

7.15.1 Detailed Description

template<typename T, int Dim> class RBF::MQ< T, Dim >

Generic functor to evaluate Multiquadrics.

All the functionality of these classes is in the `operator()`. This operator implements the complete MQ-RBF function. In this case:

$$\phi(r) = \sqrt{r^2 + c^2}$$

where c is the shape parameter, and the r is defined as $r = |x - x_j|$ in 1D, $r = \sqrt{(x - x_j)^2 + (y - y_j)^2}$ in 2D, and $r = \sqrt{(x - x_j)^2 + (y - y_j)^2 + (z - z_j)^2}$ in 3D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 42 of file MQ.hpp.

7.16 RBF::MQ< T, 1 > Class Template Reference

Multiquadric specialization for 1D.

Private Attributes

- T c2
  Squared shape parameter.

7.16.1 Detailed Description

template<typename T> class RBF::MQ< T, 1 >

Multiquadric specialization for 1D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 51 of file MQ.hpp.

7.17 RBF::MQ< T, 2 > Class Template Reference

Multiquadric specialization for 2D.

Private Attributes

- T c2
  Squared shape parameter.

7.17.1 Detailed Description

template<typename T> class RBF::MQ< T, 2 >

Multiquadric specialization for 2D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 69 of file MQ.hpp.

7.18 RBF::MQ< T, 3 > Class Template Reference

Multiquadric specialization for 3D.

Private Attributes

- T c2

  Squared shape parameter.

7.18.1 Detailed Description

template<typename T> class RBF::MQ< T, 3 >

Multiquadric specialization for 3D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 87 of file MQ.hpp.

7.19 RBF::MQ_1DX< T, Dim > Class Template Reference

Generic functor to evaluate the 1st derivative of MQ with respect to x.

7.19.1 Detailed Description

template<typename T, int Dim> class RBF::MQ_1DX< T, Dim >

Generic functor to evaluate the 1st derivative of MQ with respect to x.

All the functionality of these classes are in the `operator()`. This operator implements the first derivative of MQ-RBF function with respect to x. In this case:

$$\frac{\partial \phi(r)}{\partial x} = \frac{(x - x_j)}{\sqrt{r^2 + c^2}}$$

where c is the shape parameter, $r = |x - x_j|$ in 1D, $r = \sqrt{(x - x_j)^2 + (y - y_j)^2}$ in 2D, and $r = \sqrt{(x - x_j)^2 + (y - y_j)^2 + (z - z_j)^2}$ in 3D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 114 of file MQ.hpp.

7.20 RBF::MQ_1DX< T, 1 > Class Template Reference

1st derivative of MQ with respect to x for 1D.

Private Attributes

- T c2

  Squared shape parameter.

7.20.1 Detailed Description

template<typename T> class RBF::MQ_1DX< T, 1 >

1st derivative of MQ with respect to x for 1D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 123 of file MQ.hpp.

7.21 RBF::MQ_1DX< T, 2 > Class Template Reference

1st derivative of MQ with respect to x for 2D.

Private Attributes

- T c2

  Squared shape parameter.

7.21.1 Detailed Description

template<typename T> class RBF::MQ_1DX< T, 2 >

1st derivative of MQ with respect to x for 2D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 140 of file MQ.hpp.

7.22 RBF::MQ_1DX< T, 3 > Class Template Reference

1st derivative of MQ with respect to x for 3D.

Private Attributes

- T c2

  Squared shape parameter.

7.22.1 Detailed Description

template<typename T> class RBF::MQ_1DX< T, 3 >

1st derivative of MQ with respect to x for 3D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 157 of file MQ.hpp.

7.23 RBF::MQ_1DY< T, Dim > Class Template Reference

Generic functor to evaluate the 1st derivative of MQ with respect to y.

7.23.1 Detailed Description

template<typename T, int Dim> class RBF::MQ_1DY< T, Dim >

Generic functor to evaluate the 1st derivative of MQ with respect to y.

All the functionality of these classes are in the `operator()`. This operator implements the first derivative of MQ-RBF function with respect to y. In this case:

$$\frac{\partial \phi(r)}{\partial x} = \frac{(y - y_j)}{\sqrt{r^2 + c^2}}$$

where c is the shape parameter, $r = \sqrt{(x - x_j)^2 + (y - y_j)^2}$ in 2D, and $r = \sqrt{(x - x_j)^2 + (y - y_j)^2 + (z - z_j)^2}$ in 3D.

NOTE: The 1D version is not needed in this case.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 260 of file MQ.hpp.

7.24 RBF::MQ_1DY< T, 2 > Class Template Reference

1st derivative of MQ with respect to y for 2D.

7.24.1 Detailed Description

template<typename T> class RBF::MQ_1DY< T, 2 >

1st derivative of MQ with respect to y for 2D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 269 of file MQ.hpp.

7.25 RBF::MQ_1DY< T, 3 > Class Template Reference

1st derivative of MQ with respect to y for 3D.

Private Attributes

- T c2

  Squared shape parameter.

7.25.1 Detailed Description

template<typename T> class RBF::MQ_1DY< T, 3 >

1st derivative of MQ with respect to y for 3D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 287 of file MQ.hpp.

7.26 RBF::MQ_1DZ< T, Dim > Class Template Reference

Generic functor to evaluate the 1st derivative of MQ with respect to z.

7.26.1 Detailed Description

template<typename T, int Dim> class RBF::MQ_1DZ< T, Dim >

Generic functor to evaluate the 1st derivative of MQ with respect to z.

All the functionality of these classes are in the `operator()`. This operator implements the first derivative of MQ-RBF function with respect to z. In this case:

$$\frac{\partial \phi(r)}{\partial x} = \frac{(z - z_j)}{\sqrt{r^2 + c^2}}$$

where c is the shape parameter, $r = \sqrt{(x - x_j)^2 + (y - y_j)^2 + (z - z_j)^2}$ in 3D.

NOTE: The 1D and 2D versions are not needed in this case.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 371 of file MQ.hpp.

7.27 RBF::MQ_1DZ< T, 3 > Class Template Reference

1st derivative of MQ with respect to z for 3D.

Private Attributes

- T c2

  Squared shape parameter.

7.27.1 Detailed Description

template<typename T> class RBF::MQ_1DZ< T, 3 >

1st derivative of MQ with respect to z for 3D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 379 of file MQ.hpp.

7.28 RBF::MQ_2DX< T, Dim > Class Template Reference

Generic functor to evaluate the 2nd derivative of MQ with respect to x.

7.28.1 Detailed Description

template<typename T, int Dim> class RBF::MQ_2DX< T, Dim >

Generic functor to evaluate the 2nd derivative of MQ with respect to x.

All the functionality of these classes are in the `operator()`. This operator implements the second derivative of MQ-RBF function with respect to x. In this case:

$$\frac{\partial^2 \phi(r)}{\partial x^2} = \frac{r^2 + c^2 - (x - x_j)^2}{(r^2 + c^2)^{3/2}}$$

where c is the shape parameter, $r = |x - x_j|$ in 1D, $r = \sqrt{(x - x_j)^2 + (y - y_j)^2}$ in 2D, and $r = \sqrt{(x - x_j)^2 + (y - y_j)^2 + (z - z_j)^2}$ in 3D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 185 of file MQ.hpp.

7.29 RBF::MQ_2DX< T, 1 > Class Template Reference

2nd derivative of MQ with respect to x for 1D.

Private Attributes

- T c2

  Squared shape parameter.

7.29.1 Detailed Description

template<typename T> class RBF::MQ_2DX< T, 1 >

2nd derivative of MQ with respect to x for 1D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 194 of file MQ.hpp.

7.30 RBF::MQ_2DX< T, 2 > Class Template Reference

2nd derivative of MQ with respect to x for 2D.

Private Attributes

- T c2

  Squared shape parameter.

7.30.1 Detailed Description

template<typename T> class RBF::MQ_2DX< T, 2 >

2nd derivative of MQ with respect to x for 2D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 212 of file MQ.hpp.

7.31 RBF::MQ_2DX< T, 3 > Class Template Reference

2nd derivative of MQ with respect to x for 3D.

Private Attributes

- T c2

  Squared shape parameter.

7.31.1 Detailed Description

template<typename T> class RBF::MQ_2DX< T, 3 >

2nd derivative of MQ with respect to x for 3D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 231 of file MQ.hpp.

7.32 RBF::MQ_2DY< T, Dim > Class Template Reference

Generic functor to evaluate the 2nd derivative of MQ with respect to y.

7.32.1 Detailed Description

template<typename T, int Dim> class RBF::MQ_2DY< T, Dim >

Generic functor to evaluate the 2nd derivative of MQ with respect to y.

All the functionality of these classes are in the `operator()`. This operator implements the second derivative of MQ-RBF function with respect to y. In this case:

$$\frac{\partial^2 \phi(r)}{\partial x^2} = \frac{r^2 + c^2 - (y - y_j)^2}{(r^2 + c^2)^{3/2}}$$

where c is the shape parameter, $r = \sqrt{(x - x_j)^2 + (y - y_j)^2}$ in 2D, and $r = \sqrt{(x - x_j)^2 + (y - y_j)^2 + (z - z_j)^2}$ in 3D.

NOTE: The 1D version is not needed in this case.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 314 of file MQ.hpp.

7.33 RBF::MQ_2DY< T, 2 > Class Template Reference

2nd derivative of MQ with respect to y for 2D.

7.33.1 Detailed Description

template<typename T> class RBF::MQ_2DY< T, 2 >

2nd derivative of MQ with respect to y for 2D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 323 of file MQ.hpp.

7.34 RBF::MQ_2DY< T, 3 > Class Template Reference

2nd derivative of MQ with respect to y for 3D.

Private Attributes

- T c2
  Squared shape parameter.

7.34.1 Detailed Description

template<typename T> class RBF::MQ_2DY< T, 3 >

2nd derivative of MQ with respect to y for 3D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 343 of file MQ.hpp.

7.35 RBF::MQ_2DZ< T, Dim > Class Template Reference

Generic functor to evaluate the 2nd derivative of MQ with respect to z.

7.35.1 Detailed Description

template<typename T, int Dim> class RBF::MQ_2DZ< T, Dim >

Generic functor to evaluate the 2nd derivative of MQ with respect to z.

All the functionality of these classes are in the `operator()`. This operator implements the second derivative of MQ-RBF function with respect to z. In this case:

$$\frac{\partial^2 \phi(r)}{\partial x^2} = \frac{r^2 + c^2 - (z - z_j)^2}{(r^2 + c^2)^{3/2}}$$

where c is the shape parameter, $r = \sqrt{(x - x_j)^2 + (y - y_j)^2 + (z - z_j)^2}$ in 3D.

NOTE: The 1D and 2D versions are not needed in this case.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 406 of file MQ.hpp.

7.36 RBF::MQ_2DZ< T, 3 > Class Template Reference

2nd derivative of MQ with respect to z for 3D.

Private Attributes

- T c2

  Squared shape parameter.

7.36.1 Detailed Description

template<typename T> class RBF::MQ_2DZ< T, 3 >

2nd derivative of MQ with respect to z for 3D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 415 of file MQ.hpp.

7.37 RBF::TPS< T, Dim > Class Template Reference

Generic functor to evaluate Thin Plate Spline.

7.37.1 Detailed Description

template<typename T, int Dim> class RBF::TPS< T, Dim >

Generic functor to evaluate Thin Plate Spline.

All the functionality of these classes is in the `operator()`. This operator implements the complete TPS-RBF function. In this case:

$$\phi(r) = r^4 \log(r)$$

where $r = |x - x_j|$ in 1D, $r = \sqrt{(x-x_j)^2 + (y-y_j)^2}$ in 2D, and $r = \sqrt{(x-x_j)^2 + (y-y_j)^2 + (z-z_j)^2}$ in 3D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 42 of file TPS.hpp.

7.38 RBF::TPS< T, 2 > Class Template Reference

Thin Plate Spline specialization for 2D.

7.38.1 Detailed Description

template<typename T> class RBF::TPS< T, 2 >

Thin Plate Spline specialization for 2D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 51 of file TPS.hpp.

7.39 RBF::TPS_1DX< T, Dim > Class Template Reference

Generic functor to evaluate the 1st derivative of TPS with respect to x.

7.39.1 Detailed Description

template<typename T, int Dim> class RBF::TPS_1DX< T, Dim >

Generic functor to evaluate the 1st derivative of TPS with respect to x.

All the functionality of these classes are in the `operator()`. This operator implements the first derivative of TPS-RBF function with respect to x. In this case:

$$\frac{\partial \phi(r)}{\partial x} = 2r^2(x - x_j)log(r^2) + r^2(x - x_j)$$

where $r = |x - x_j|$ in 1D, $r = \sqrt{(x - x_j)^2 + (y - y_j)^2}$ in 2D, and $r = \sqrt{(x - x_j)^2 + (y - y_j)^2 + (z - z_j)^2}$ in 3D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 75 of file TPS.hpp.

7.40 RBF::TPS_1DX< T, 2 > Class Template Reference

1st derivative of TPS with respect to x for 2D.

7.40.1 Detailed Description

template<typename T> class RBF::TPS_1DX< T, 2 >

1st derivative of TPS with respect to x for 2D.

Author:

Luis M. de la Cruz [Mo]

Definition at line 83 of file TPS.hpp.

7.41 RBF::TPS_1DY< T, Dim > Class Template Reference

Generic functor to evaluate the 1st derivative of TPS with respect to y.

7.41.1 Detailed Description

template<typename T, int Dim> class RBF::TPS_1DY< T, Dim >

Generic functor to evaluate the 1st derivative of TPS with respect to y.

All the functionality of these classes are in the `operator()`. This operator implements the first derivative of TPS-RBF function with respect to y. In this case:

$$\frac{\partial \phi(r)}{\partial y} = 2r^2(y - y_j)log(r^2) + r^2(y - y_j)$$

where $r = \sqrt{(x - x_j)^2 + (y - y_j)^2}$ in 2D, and $r = \sqrt{(x - x_j)^2 + (y - y_j)^2 + (z - z_j)^2}$ in 3D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 112 of file TPS.hpp.

7.42 RBF::TPS_1DY< T, 2 > Class Template Reference

1st derivative of TPS with respect to y for 2D.

7.42.1 Detailed Description

template<typename T> class RBF::TPS_1DY< T, 2 >

1st derivative of TPS with respect to y for 2D.

Author:

Luis M. de la Cruz [Fri Apr 25 13:28:47 BST 2008]

Definition at line 120 of file TPS.hpp.

7.43 RBF::TPS_2DX< T, Dim > Class Template Reference

Generic functor to evaluate the 2nd derivative of TPS with respect to x.

7.43.1 Detailed Description

template<typename T, int Dim> class RBF::TPS_2DX< T, Dim >

Generic functor to evaluate the 2nd derivative of TPS with respect to x.

All the functionality of these classes are in the `operator()`. This operator implements the second derivative of TPS-RBF function with respect to x. In this case:

$$\frac{\partial^2 \phi(r)}{\partial x^2} = 4(x - x_i)^2 \log(r^2) + 2r^2 \log(r^2) + 6(x - x_j)^2 + r^2$$

where $r = |x - x_j|$ in 1D, $r = \sqrt{(x - x_j)^2 + (y - y_j)^2}$ in 2D, and $r = \sqrt{(x - x_j)^2 + (y - y_j)^2 + (z - z_j)^2}$ in 3D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 151 of file TPS.hpp.

7.44 RBF::TPS_2DX< T, 2 > Class Template Reference

2nd derivative of TPS with respect to x for 2D.

7.44.1 Detailed Description

template<typename T> class RBF::TPS_2DX< T, 2 >

2nd derivative of TPS with respect to x for 2D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 159 of file TPS.hpp.

7.45 RBF::TPS_2DY< T, Dim > Class Template Reference

Generic functor to evaluate the 2nd derivative of TPS with respect to y.

7.45.1 Detailed Description

template<typename T, int Dim> class RBF::TPS_2DY< T, Dim >

Generic functor to evaluate the 2nd derivative of TPS with respect to y.

All the functionality of these classes are in the `operator()`. This operator implements the second derivative of TPS-RBF function with respect to y. In this case:

$$\frac{\partial^2 \phi(r)}{\partial y^2} = 4(y - y_i)^2 \log(r^2) + 2r^2 \log(r^2) + 6(y - y_j)^2 + r^2$$

where $r = \sqrt{(x - x_j)^2 + (y - y_j)^2}$ in 2D, and $r = \sqrt{(x - x_j)^2 + (y - y_j)^2 + (z - z_j)^2}$ in 3D.

Author:

Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 188 of file TPS.hpp.

7.46 RBF::TPS_2DY< T, 2 > Class Template Reference

2nd derivative of TPS with respect to y for 2D.

7.46.1 Detailed Description

template<typename T> class RBF::TPS_2DY< T, 2 >

2nd derivative of TPS with respect to y for 2D.

Author:

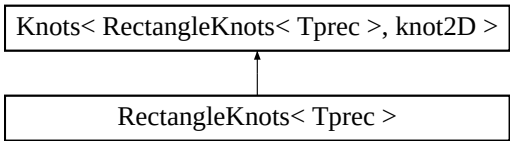
Luis M. de la Cruz [Mon Nov 12 14:02:37 GMT 2007]

Definition at line 196 of file TPS.hpp.

7.47 RectangleKnots< Tprec > Class Template Reference

Rectangular domain.

Inheritance diagram for RectangleKnots< Tprec >::



Public Member Functions

- RectangleKnots ()

  Default constructor.

- RectangleKnots (prec_t l_x, int n_x, prec_t l_y, int n_y, random_t knd=U, int layer=0)

  Constructor to generate knots in the centers of the "cells" of a gridded mesh, totally random knots or controlled random knots.

- bool constructKnots ()

  Construct all the knots of the domain.

- int getGhostKnots ()

  Get the number of Ghost knots around the domain.

- bool addSpecialPoints (iMat &, int, int)

  Add special points for the ACBF aproximation.

- void setRandomness (prec_t e)

  Random parameter.

- bool gridKnots ()

Generate knots in a "regular grid".

- bool print ()

  Print info to the standard output (screen).

Private Attributes

- prec_t lx

  Horizontal length of the rectangle.

- prec_t ly

  Vertical length of the rectangle.

- int Nx

  Number of points on x-axis.

- int Ny

  Number of points on y-axis.

- prec_t epru

  Used to control randomness in regular distr.

- int NG

  Number of ghost points around de domain.

7.47.1 Detailed Description

template<typename Tprec> class RectangleKnots< Tprec >

Rectangular domain.

This class is aimed to generate knots inside of rectangular domains. The function constructKnots() must be reimplemented here to generate the knots on the rectangular domain.

Author:

Luis M. de la Cruz [Mon Nov 12 14:53:34 GMT 2007]

Definition at line 39 of file RectangleKnots.hpp.

7.47.2 Constructor & Destructor Documentation

7.47.2.1 template<typename Tprec> RectangleKnots< Tprec >::RectangleKnots (prec_t *l_x*, int *n_x*, prec_t *l_y*, int *n_y*, random_t *knd* = `U`, int *layer* = `0`) `[inline]`

Constructor to generate knots in the centers of the "cells" of a gridded mesh, totally random knots or controlled random knots.

Parameters:

l_x Length of the rectangle in x-axis direction.

l_y Length of the rectangle in y-axis direction.

n_x Number of points in x-axis direction.

n_y Number of points in y-axis direction.

knd distribution of points (0 unif, 1 rand, 2 rand unif).

layer 0 no layers around the domain, 1 just one layer.

Definition at line 80 of file RectangleKnots.hpp.

7.47.3 Member Function Documentation

7.47.3.1 template<typename prec_t > bool RectangleKnots< prec_t >::constructKnots () [inline]

Construct all the knots of the domain.

This function generate nodes in a uniform grid (knd = 0), randomly distributed (knd = 1) and random controlled (knd = 2, 0 < epru < 0.5).

The random controlled is generated as follows: Taking the values of `Nx` and `Ny` an "invisible grid" is defined inside the rectangle, this grid will contains (`Nx` - 1) `X` (`Ny` - 1) "cells", for each cell a knot is generated and this knot is allowed to be in a random position inside its cell, the randomness depends on the `ep` parameter. The order of the knots in the `xyz` array is as shown in the next figure:

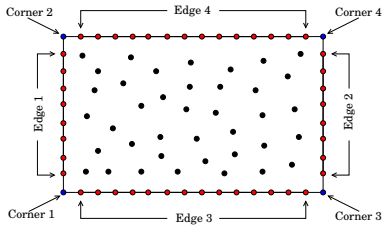


Figure 2: Knots order

Reimplemented from Knots< RectangleKnots< Tprec >, knot2D >.

Definition at line 179 of file RectangleKnots.hpp.

7.47.3.2 template<typename prec_t > bool RectangleKnots< prec_t >::addSpecialPoints (iMat & *P*, int *j*, int *nsp*) [inline]

Add special points for the ACBF aproximation.

Add special points located in the boundary of the rectangle.

Parameters:

P indices of neighbors knots.

j actual knot.

nsp number of special points.

Definition at line 292 of file RectangleKnots.hpp.

7.47.3.3 template<typename prec_t > bool RectangleKnots< prec_t >::gridKnots () [inline]

Generate knots in a "regular grid".

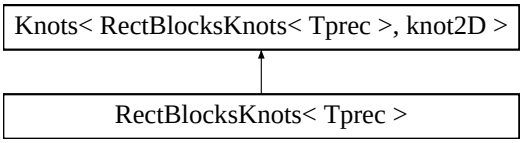
This function is aimed to generate the knots in a Cartesian uniform grid.

Definition at line 406 of file RectangleKnots.hpp.

7.48 RectBlocksKnots< Tprec > Class Template Reference

Rectangular domain with blocks near boundaries.

Inheritance diagram for RectBlocksKnots< Tprec >::



Public Member Functions

- RectBlocksKnots ()

  Default constructor.

- RectBlocksKnots (prec_t l_x, int n_x, prec_t l_y, int n_y, random_t knd=U, int layer=0, int nbl=0, int nbr=0, int nbs=0, int nbn=0, int nix=-1, int niy=-1)

  Constructor to generate knots regularly spaced near the boundaries and random our uniform knots in the internal block.

- bool constructKnots ()

  Construct all the knots of the domain.

- bool addSpecialPoints (iMat &, int, int)

  Add special points for the ACBF aproximation.

- void setRandomness (prec_t e)

  Random parameter.

- bool print ()

  Print info to the standard output (screen).

Private Attributes

- prec_t lx

  Horizontal length of the rectangle.

- prec_t ly

  Vertical length of the rectangle.

- int Nx

  Number of points on x-axis.

- int Ny

  Number of points on y-axis.

- prec_t epru

  Randomness of the points distribution (default ep = 0).

- int NG

  Number of ghost points around de domain.

7.48.1 Detailed Description

template<typename Tprec> class RectBlocksKnots< Tprec >

Rectangular domain with blocks near boundaries.

This class is aimed to generate knots inside of rectangular domains with regular distribution near the boundaries. The function constructKnots() must be reimplemented here to generate the knots on the rectangular domain.

Author:

Luis M. de la Cruz [Tue Jan 22 15:24:58 GMT 2008]

Definition at line 40 of file RectBlocksKnots.hpp.

7.48.2 Constructor & Destructor Documentation

7.48.2.1 template<typename Tprec> RectBlocksKnots< Tprec >::RectBlocksKnots (prec_t *l_x*, int *n_x*, prec_t *l_y*, int *n_y*, random_t *knd* = U, int *layer* = 0, int *nbl* = 0, int *nbr* = 0, int *nbs* = 0, int *nbn* = 0, int *nix* = -1, int *niy* = -1) [inline]

Constructor to generate knots regularly spaced near the boundaries and random our uniform knots in the internal block.

Parameters:

l_x Length of the rectangle in x-axis direction.

n_x Number of points in x-axis direction.

l_y Length of the rectangle in y-axis direction.

n_y Number of points in y-axis direction.

knd distribution of points (0 unif, 1 rand, 2 rand unif).

layer 0 no layers around the domain, 1 just one layer.

nbl x-length of the block 1

nbr x-length of the block 2

nbs y-length of the block 3

nbn y-length of the block 4

nix Number of points in x-axis for the internal block

niy Number of points in y-axis for the internal block

Definition at line 90 of file RectBlocksKnots.hpp.

7.48.3 Member Function Documentation

7.48.3.1 template<typename prec_t > bool RectBlocksKnots< prec_t >::constructKnots () [inline]

Construct all the knots of the domain.

This function generate nodes in a uniform grid (knd = 0), randomly distributed (knd = 1) and random controlled (knd = 2, 0 < epru < 0.5).

The random controlled is generated as follows: Taking the values of `Nx` and `Ny` an "invisible grid" is defined inside the rectangle, this grid will contains (`Nx` - 1) `X` (`Ny` - 1) "cells", for each cell a knot is generated and this knot is allowed to be in a random position inside its cell, the randomness depends on the `ep` parameter. Also, it is possible to generate blocks of uniform-gridded points near the boundaries setting the parameters `nbl` , `nbr` , `nbs` and `nbn` . Look at the next figure:

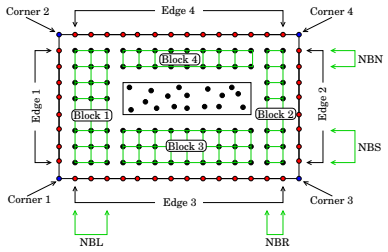


Figure 3: Knots order

Reimplemented from Knots< RectBlocksKnots< Tprec >, knot2D >.

Definition at line 207 of file RectBlocksKnots.hpp.

7.48.3.2 template<typename prec_t > bool RectBlocksKnots< prec_t >::addSpecialPoints (iMat & *P*, int *j*, int *nsp*) [inline]

Add special points for the ACBF aproximation.

Add special points located in the boundary of the rectangle and in the geometric centroid.

Parameters:

P indices of neighbors knots.

j actual knot.

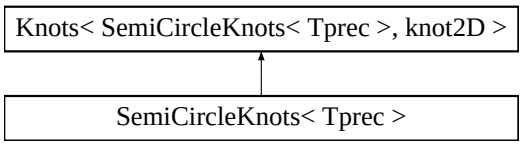
nsp number of special points.

Definition at line 428 of file RectBlocksKnots.hpp.

7.49 SemiCircleKnots< Tprec > Class Template Reference

SemiCircle domain.

Inheritance diagram for SemiCircleKnots< Tprec >::



Public Member Functions

- SemiCircleKnots ()

  Default constructor.

- SemiCircleKnots (prec_t r1, prec_t r2, int Nr, prec_t t1, prec_t t2, int Nt, random_t knd=U)

  Constructor to generate knots in the centers of the "cells" of a gridded mesh, totally random knots or controlled random knots.

- bool constructKnots ()

  Construct all the knots of the domain.

- bool addSpecialPoints (iMat &, int, int)

  Add special points for the ACBF aproximation.

- void setRandomness (prec_t e)

  Random parameter.

- bool gridKnots ()

  Generate knots in a "Cartesian grid".

- bool print ()

  Print info to the standard output (screen).

Private Attributes

- prec_t theta_1

  Initial angle.

- prec_t theta_2
  Final angle.
- prec_t radius_1
  Initial radius.
- prec_t radius_2
  Final radius.
- int Ntheta
  Number of nodes in theta.
- int Nradius
  Number of nodes in radius.
- prec_t epru
  Used to control randomness in regular distr.

7.49.1 Detailed Description

template<typename Tprec> class SemiCircleKnots< Tprec >

SemiCircle domain.

This class is aimed to generate knots inside of semicircle domains. The function constructKnots() must be reimplemented here to generate the knots in the semicircle domain.

Author:

Luis M. de la Cruz [Fri Jan 25 13:48:24 GMT 2008]

Definition at line 43 of file SemiCircleKnots.hpp.

7.49.2 Constructor & Destructor Documentation

7.49.2.1 template<typename Tprec> SemiCircleKnots< Tprec >::SemiCircleKnots (prec_t *r1*, prec_t *r2*, int *Nr*, prec_t *t1*, prec_t *t2*, int *Nt*, random_t *knd* = U) [inline]

Constructor to generate knots in the centers of the "cells" of a gridded mesh, totally random knots or controlled random knots.

Parameters:

r1 radius 1 as shown in the next figure.

r2 radius 2 as shown in the next figure.

t1 angle in degrees 1 as shown in the next figure.

t2 angle in degrees 2 as shown in the next figure.

Nr Number of points in radius.

Nt Number of points in theta.

knd distribution of points (0 unif, 1 rand, 2 rand unif).

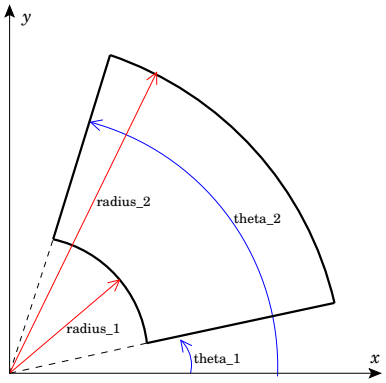


Figure 4: Semicircle definition.

Definition at line 87 of file SemiCircleKnots.hpp.

7.49.3 Member Function Documentation

7.49.3.1 template<typename prec_t > bool SemiCircleKnots< prec_t >::constructKnots () [inline]

Construct all the knots of the domain.

This function generate nodes in a uniform grid (knd = 0), randomly distributed (knd = 1) and random controlled (knd = 2, 0 < epru < 1).

The random controlled is generated as follows: Taking the values of `Nr` and `Nt` an "invisible grid" is defined inside the semicircle, this grid will contains (`Nr` - 1) `X` (`Nt` - 1) "cells", for each cell a knot is generated and this knot is allowed to be in a random position inside its cell, the randomness depends on the `ep` parameter. The order of the knots in the `xyz` array is as shown in the next figure:

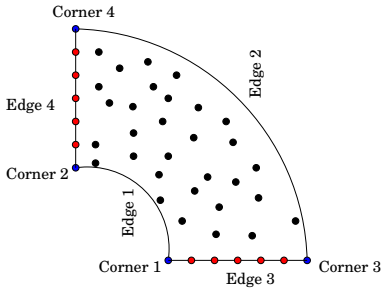


Figure 5: Knots order

Reimplemented from Knots< SemiCircleKnots< Tprec >, knot2D >.

Definition at line 174 of file SemiCircleKnots.hpp.

7.49.3.2 template<typename prec_t > bool SemiCircleKnots< prec_t >::addSpecialPoints (iMat & *P*, int *j*, int *nsp*) [inline]

Add special points for the ACBF aproximation.

Add special points located in the boundary of the semicircle and in the geometric centroid.

Parameters:

P indices of neighbors knots.

j actual knot.

nsp number of special points.

Definition at line 300 of file SemiCircleKnots.hpp.

7.50 timer Struct Reference

Tool for time measurements (borrowed from FLENS).

Public Member Functions

- void tic ()
  Start the clock.
- prec_t toc () const
  Stop the clock.

7.50.1 Detailed Description

Tool for time measurements (borrowed from FLENS).

Definition at line 47 of file Traits.hpp.

8 File Documentation

8.1 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/01TestKnots/box.c File Reference

Testing the BoxKnots class.

8.1.1 Detailed Description

Testing the BoxKnots class.

In this example some features of the class BoxKnots are tested. Particularly the function Knots::findNeighbors() is used to find the neighborhood of a target point.

Input

The `inputBox` file contains the input data required for this program: `hx` length in x-axis; `hy` length in y-axis; `hz` length in z-axis; `Nx` number of points in x-axis; `Ny` number of points in y-axis; `Nz` number of points in z-axis;

Output

`xyzBox.dat` coordinates of random points; `tarBox.dat` the target point; `neiBox.dat` list of neighbors.

Post-procesing

You can plot the results using the next command in gnuplot:

```
% gnuplot> splot "neiBox.dat" w lp, "xyzBox.dat" w p, "tarBox.dat" w p
```

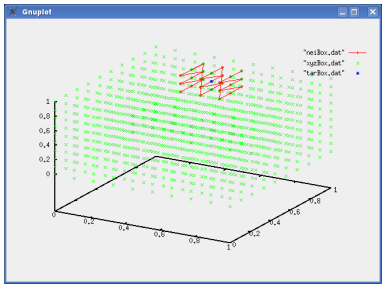


Figure 6: Knots, target and neighborhood.

Author:

Luis M. de la Cruz [Mon Apr 28 10:31:36 BST 2008]

Definition in file box.cpp.

8.2 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/01TestKnots/line.c File Reference

Testing the LineKnots class.

8.2.1 Detailed Description

Testing the LineKnots class.

In this example some features of the class LineKnots are tested. Particularly the function Knots::findNeighbors() is used to find the neighborhood of a target point.

Input

The `inputLine` contains the input data required for this program: `hx` length in x-axis; `Nx` number of points in x-axis;

Output

`xyzLine.dat` coordinates of random points; `tarLine.dat` the target point; `neiLine.dat` list of neighbors of the target.

Post-procesing

You can plot the results using the next command in gnuplot:

```
% gnuplot> p "neiLine.dat" w lp, "xyzLine.dat" w p, "tarLine.dat" w p
```

Author:

Luis M. de la Cruz [Thu Sep 6 14:35:41 BST 2007]

Definition in file line.cpp.

8.3 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/01TestKnots/recta File Reference

Testing the RectangleKnots class.

8.3.1 Detailed Description

Testing the RectangleKnots class.

In this example some features of the class RectangleKnots are tested. Particularly the function Knots::findNeighbors() is used to find the neighborhood of a target point.

Input

The `inputRectangle` file contains the input data required for this program: `hx` length in x-axis; `hy` length in y-axis; `Nx` number of points in x-axis; `Ny` number of points in y-axis; `rtype` knots distribution, (0 unif, 1 rand, 2 rand unif); `ep` degree of randomness. `layer` 0 no ghost points, 1 one layer of ghost points

Output

`xyzRect.dat` coordinates of random points; `tarRect.dat` the target point; `neiRect.dat` list of neighbors of the target.

Post-procesing

You can plot the results using the next command in gnuplot:

```
% gnuplot> p "neiRect.dat" w lp, "xyzRect.dat" w p, "tarRect.dat" w p
```

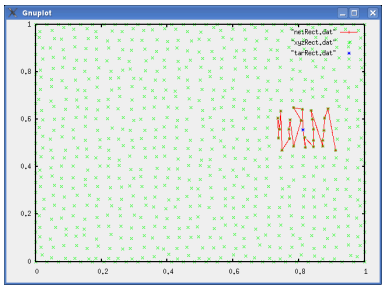


Figure 7: Knots, target and neighborhood.

Author:

Luis M. de la Cruz [Thu Sep 6 14:35:41 BST 2007]

Definition in file rectangle.cpp.

8.4 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/01TestKnots/recta blocks.cpp File Reference

Testing the RectBlocksKnots class.

8.4.1 Detailed Description

Testing the RectBlocksKnots class.

In this example some features of the class RectBlocksKnots are tested. Particularly the function Knots::findNeighbors() is used to find the neighborhood of a target point.

Input

The `inputRectangle` file contains the input data required for this program: `hx` length in x-axis; `hy` length in y-axis; `Nx` number of points in x-axis; `Ny` number of points in y-axis; `NBL` , `NBR`, `NBN` , `NBS`, the size of blocks; `NIx`, `NIy` , the size of the internal block; `rtype` knots distribution, (0 unif, 1 rand, 2 rand unif); `ep` degree of randomness. `layer` 0 no ghost points, 1 one layer of ghost points

Output

`xyzRect.dat` coordinates of random points; `tarRect.dat` the target point; `neiRect.dat` list of neighbors of the target.

Post-procesing

You can plot the results using the next command in gnuplot:

```
% gnuplot> p "neiRect.dat" w lp, "xyzRect.dat" w p, "tarRect.dat" w p
```

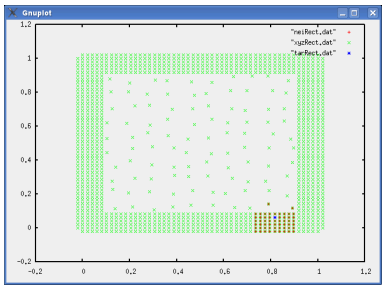


Figure 8: Knots, target and neighborhood.

Author:

Luis M. de la Cruz [Thu Sep 6 14:35:41 BST 2007]

Definition in file rectangle_blocks.cpp.

8.5 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/01TestKnots/semic File Reference

Testing the SemiCircleKnots class.

8.5.1 Detailed Description

Testing the SemiCircleKnots class.

In this example some features of the class SemiCircleKnots are tested. Particularly the function Knots::findNeighbors() is used to find the neighborhood of a target point.

Input

The `inputSemiCircle` file contains the input data required for this program: `r1` and `r2` Radii for defining the semicircle; `t1` and `t2` Angles for defining the semicircle; `Nr` number of points in R; `Nt` number of points in theta; `rtype` knots distribution (0 unif, 1 rand, 2 rand unif); `ep` degree of randomness.

Output

`xyzSemi.dat` coordinates of random points; `tarSemi.dat` the target point; `neiSemi.dat` list of neighbors of the target.

Post-procesing

You can plot the results using the next command in gnuplot:

```
% gnuplot> p "neiSemi.dat" w lp, "xyzSemi.dat" w p, "tarSemi.dat" w p
```

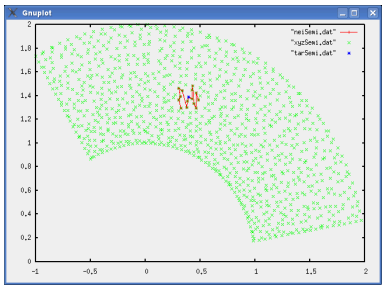


Figure 9: Knots, target and neighborhood.

Author:

Luis M. de la Cruz [Thu Sep 6 14:35:41 BST 2007]

Definition in file semicircle.cpp.

8.6 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/02Heat2D/heat2D. File Reference

Poisson equation in 2D.

8.6.1 Detailed Description

Poisson equation in 2D.

The PDE to solve is:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

and the boundary conditions are as shown in the next figure:

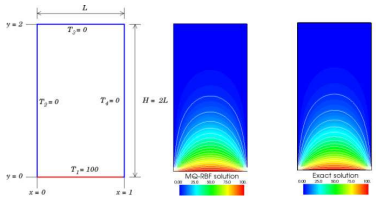


Figure 10: Geometry, boundary conditions and solution

Input

The `input` file contains the following initial data:

- `hx`, `hy`, `Nx`, `Ny` : Lenghts of the rectangle, and number of points in each axis.
- `rtype`, `ep`, `layer` : Type of knots distribution, randomness, layer around the domain.
- `c` : Shape parameter for MQ-RBF kernel ($c < 0$ implies $c = 1/\sqrt{N}$).
- `fsol` : choose a method to solve the system.
- `max_knots` : neighbors used to construct the ACBF preconditioner.

Output

- `xy_knots.dat` (x,y) coordinates of random points;
- `solution.dat` (x,y,u) evaluation of the solution in a grid;
- `errordis.dat` (x,y,e) error distribution in a grid;
- `exactsol.dat` (x,y,u) exact solution in a grid;

Author:

Luis M. de la Cruz [Thu Sep 6 14:35:41 BST 2007]

Definition in file heat2D.cpp.

8.7 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/03HeatSemiCircle File Reference

Laplace equation in a semicircle domain.

8.7.1 Detailed Description

Laplace equation in a semicircle domain.

The PDE to solve is:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

and the boundary conditions are (see the figure):

- $T = \sin(\theta)/R_{0A}$ for segment AC;
- $T = \sin(\theta)/R_{0B}$ for segment BD;
- $T = 0$ for segment AB;
- $T = 1/r$ for segment CD.

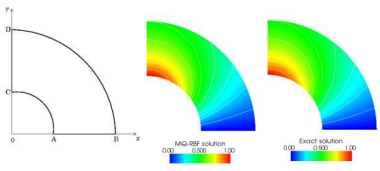


Figure 11: Geometry, boundary conditions and solution

Input

The `input` file contains the initial data to setup the problem.

- `hx`, `hy`, `Nx`, `Ny` : Lenghts of the rectangle, and number of points in each axis.
- `rtype`, `ep`, `layer` : Type of knots distribution, randomness, layer around the domain.
- `c` : Shape parameter for MQ-RBF kernel ($c < 0$ implies $c = 1/\sqrt{N}$).
- `fsol` : choose a method to solve the system.
- `max_knots` : neighbors used to construct the ACBF preconditioner.

Output

- `xy_knots.dat` coordinates of random points;
- `solrand.dat` (x,y,u) evaluation of the solution in the collocation pts.
- `errrand.dat` (x,y,|e-u|) error distribution in the collocation pts.
- `exarand.dat` (x,y,e) exact solution in the collocation pts.
- `solgrid.dat` (x,y,u) evaluation of the solution in a grid;
- `errgrid.dat` (x,y,|e - u|) error distribution in a grid;
- `exagrid.dat` (x,y,e) exact solution in a grid;

Author:

Luis M. de la Cruz [Thu Sep 6 14:35:41 BST 2007]

Definition in file heatsemi.cpp.

8.8 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/04ConvDiff1D/con File Reference

Non-stat adv-diff equation in 1D.

8.8.1 Detailed Description

Non-stat adv-diff equation in 1D.

In this example the time-dependent advection-diffusion equation in 1D is solved. The equation is written as follows:

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} - D\frac{\partial^2 f}{\partial x^2} = 0$$

where u represents a convective velocity. The initial and boundary conditions are:

$$\begin{aligned} f(x,0) &= 0 \text{ for } 0 \leq x < \infty, \\ f(0,t) &= 1 \text{ for } t \geq 0, \\ f(L,t) &= 0 \text{ for } t > 0, L \to \infty, \end{aligned}$$

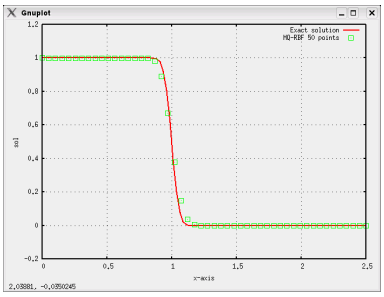


Figure 12: MQ-RBF solution for D = 0.0001 and 50 points

Input

The `input` file contains the initial data to setup the problem:

- `h` lenght of the domain
- `N` number of points
- `max_iter` max time iterations
- `dt` time step
- `D` Peclet number
- `max_knots` neighbor knots for the ACBF precond.

Output

- `xy_knots.dat` x-coordinates of the points.
- `solution.dat` evaluation of the final solution.
- `error.dat` error against the exact solution.
- `exact.dat` exact solution.

Author:

Luis M. de la Cruz [Thu Sep 6 14:35:41 BST 2007]

Definition in file convdiff01.cpp.

8.9 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/05ConvDiff2D/con File Reference

Forced Convection in 2D.

8.9.1 Detailed Description

Forced Convection in 2D.

In this example the time-dependent convection-diffusion equation is solved. The equation is written as follows:

$$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} = \Gamma\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right)$$

where (u, v) is a prescribed velocity field that fulfills the continuity equation and is given by the next formula:

$$\begin{aligned} u(x,y) &= -A\cos(\pi y)\sin(\pi\lambda x/l_x)\text{and} \\ v(x,y) &= \frac{A\lambda}{l_x}\sin(\pi y)\cos(\pi\lambda x/l_x). \end{aligned}$$

The initial condition is $T = 0$ and the boundary conditions are as shown in the next figure:

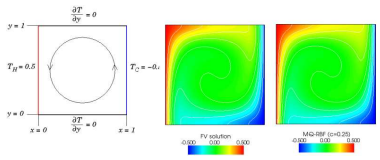


Figure 13: Geometry, boundary conditions and solution

Input

The `input` file contains the initial data to setup the problem.

- `hx`, `hy`, `Nx`, `Ny` : Lenghts of the rectangle, and number of points in each axis.
- `rtype`, `ep`, `layer` : Type of knots distribution, randomness, layer around the domain.
- `max_iter` time iterations, `dt` Step time step
- `max_knots` : neighbors used to construct the ACBF preconditioner.
- `c` : Shape parameter for MQ-RBF kernel ($c < 0$ implies $c = 1/\sqrt{N}$).
- `cboundary` : Shape parameter for MQ-RBF kernel on the boundary ($c < 0$ implies $c = 1/\sqrt{N}$).
- `A` max magnitude of the velocity.
- `Nprofile` number of points to make the velocity profiles.

Output

- `xy_knots.dat` coordinates of random points;
- `solution.dat` (x,y,u) evaluation of the solution in a grid.
- `profBF_x.dat`, `profRBF_y.dat` velocity profiles through the middle of the cavity.

Author:

Luis M. de la Cruz Wed [Dec 19 14:49:56 GMT 2007]

Definition in file convdiff2D.cpp.

8.10 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/07LidDriven/lid_driven2.cpp File Reference

Lid Driven Cavity in 2D.

8.10.1 Detailed Description

Lid Driven Cavity in 2D.

In this example the lid-driven cavity is solved in the vorticity - stream function formulation. The equations to solve are:

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = w$$

and

$$\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} = Re\left(u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y}\right)$$

where ψ is the stream function and the vorticity and the velocity are defined as follows

$$(u, v) = (\frac{\partial \psi}{\partial y}, -\frac{\partial \psi}{\partial x}) w = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$$

The boundary conditions are the typical for the lid-driven cavity, see for example [].

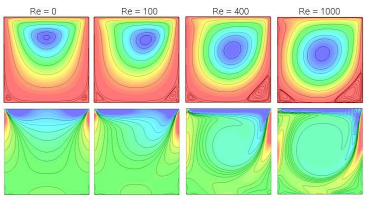


Figure 14: Streamfunction and Vorticity

Input

The `input` file contains the initial data to setup the problem:

- `hx`, hy, Nx, Ny Size of the cavity and number of points on each axis.
- `rtype`, `ep` point distribution, randomness
- `dt`, `max_iter`, `max_knots` time step, time iterations, neighbors
- `c` shape parameter for MQ-RBF kernel ($c < 0$ implies $c = 1/\sqrt{N}$).
- `Re` Reydolds number.
- `beta_s`, beta_v, beta_vb under-relaxation parameters for streamfunction, vorticity and boundary conditions for vorticity.
- `gmres_tol`, `tolerance` Tolerance for GMRES, global tolerance
- `Ngrid` points for plotting the final solution.
- `ORDER` order for the vorticity BC.

Output

- `xy_knots.dat` coordinates of random points;
- `solStream.dat` (x,y,u) Streamfunction.
- `solVort.dat` (x,y,u) Vorticity.
- `u_line.dat`, `v_line.dat` velocity profiles.

Author:

Luis M. de la Cruz Wed [Mon Apr 7 09:58:44 BST 2008]

Definition in file lid_driven2.cpp.

8.11 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/examples/08BackwardFaci File Reference

Backward-acing Step in 2D.

8.11.1 Detailed Description

Backward-acing Step in 2D.

In this example the Backward-facing step is solved in the vorticity - stream function formulation. The equations to solve are:

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = w$$

and

$$\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} = Re\left(u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y}\right)$$

where ψ is the stream function and the vorticity and the velocity are defined as follows

$$(u, v) = (\frac{\partial \psi}{\partial y}, -\frac{\partial \psi}{\partial x}) w = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$$

The boundary conditions are the typical for the backward-facing step, see for example [].

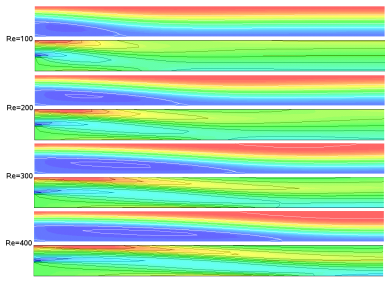


Figure 15: Streamfunction and Vorticity

Input

The `input` file contains the initial data to setup the problem.

- `hx`, hy, Nx, Ny Size of the cavity and number of points on each axis.
- `rtype`, `ep` point distribution, randomness
- `dt`, `max_iter`, `max_knots` time step, time iterations, neighbors
- `c` shape parameter for MQ-RBF kernel ($c < 0$ implies $c = 1/\sqrt{N}$).
- `Re` Reydolds number.
- `beta_s`, beta_v, beta_vb under-relaxation parameters for streamfunction, vorticity and boundary conditions for vorticity.
- `gmres_tol`, `tolerance` Tolerance for GMRES, global tolerance
- `scale_x`, scale factor for x-coordinates.

Output

- `xy_knots.dat` coordinates of random points;
- `solStream.dat` (x,y,u) Streamfunction.
- `solVort.dat` (x,y,u) Vorticity.

Author:

Luis M. de la Cruz Wed [Mon Apr 7 09:58:44 BST 2008]

Definition in file backfacing2.cpp.

8.12 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/accessor.h File Reference

Namespaces

- namespace KDTree

  Contains all the stuff for KDTree algorithm.

8.12.1 Detailed Description

Defines the interface for the Accessor class.

Author:

Martin F. Krafft <libkdtree@pobox.madduck.net>

Definition in file accessor.hpp.

8.13 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/allocator. File Reference

Namespaces

- namespace KDTree

  Contains all the stuff for KDTree algorithm.

8.13.1 Detailed Description

Defines the allocator interface as used by the KDTree class.

Author:

Martin F. Krafft <libkdtree@pobox.madduck.net>

Definition in file allocator.hpp.

8.14 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/iterator.h File Reference

Namespaces

- namespace KDTree

  Contains all the stuff for KDTree algorithm.

8.14.1 Detailed Description

Defines interfaces for iterators as used by the KDTree class.

Author:

Martin F. Krafft <libkdtree@pobox.madduck.net>

Definition in file iterator.hpp.

8.15 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/kdtree.hp File Reference

Namespaces

- namespace KDTree

  Contains all the stuff for KDTree algorithm.

8.15.1 Detailed Description

Defines the interface for the KDTree class.

Author:

Martin F. Krafft <libkdtree@pobox.madduck.net>

Paul Harris figured this stuff out (below) Notes: This is similar to a binary tree, but its not the same. There are a few important differences:

∗ Each level is sorted by a different criteria (this is fundamental to the design).

∗ It is possible to have children IDENTICAL to its parent in BOTH branches This is different to a binary tree, where identical children are always to the right So, KDTree has the relationships: ∗ The left branch is <= its parent (in binary tree, this relationship is a plain <) ∗ The right branch is <= its parent (same as binary tree)

This is done for mostly for performance. Its a LOT easier to maintain a consistent tree if we use the <= relationship. Note that this relationship only makes a difference when searching for an exact item with find() or find_exact, other search, erase and insert functions don't notice the difference.

In the case of binary trees, you can safely assume that the next identical item will be the child leaf, but in the case of KDTree, the next identical item might be a long way down a subtree, because of the various different sort criteria.

So erase()ing a node from a KDTree could require serious and complicated tree rebalancing to maintain consistency... IF we required binary-tree-like relationships.

This has no effect on insert()s, a < test is good enough to keep consistency.

It has an effect on find() searches: ∗ Instead of using compare(child,node) for a < relationship and following 1 branch, we must use !compare(node,child) for a <= relationship, and test BOTH branches, as we could potentially go down both branches.

It has no real effect on bounds-based searches (like find_nearest, find_within_range) as it compares vs a boundary and would follow both branches if required.

This has no real effect on erase()s, a < test is good enough to keep consistency.

Definition in file kdtree.hpp.

8.16 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/node.hpp File Reference

Namespaces

- namespace KDTree

  Contains all the stuff for KDTree algorithm.

8.16.1 Detailed Description

Defines interfaces for nodes as used by the KDTree class.

Author:

Martin F. Krafft <libkdtree@pobox.madduck.net>

Definition in file node.hpp.

8.17 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/kdtree++/region.hp File Reference

Namespaces

- namespace KDTree

  Contains all the stuff for KDTree algorithm.

8.17.1 Detailed Description

Defines the interface of the _Region class.

Author:

Martin F. Krafft <libkdtree@pobox.madduck.net>

Definition in file region.hpp.

8.18 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Solvers/Krylov/gm File Reference

GMRES IMPLEMENTATIONS.

Classes

- struct flens::gmres_< A >
  Type definition for GMRES.
- struct flens::gmres_< DenseVector< I > >
  Dense Vector specialization.

Namespaces

- namespace flens
  GMRES, preconditioners, based on FLENS.

Functions

- template<class Mat , class Vec >
  int flens::gmres (Mat const &A, Vec &x, const Vec &b, unsigned int const maxiter, double const tol)
  GMRES NON PRECONDITIONED.
- template<class Mat , class Vec , class Precond >
  int flens::gmres (Mat const &A, Vec &x, const Vec &b, const Precond &M, unsigned int const maxiter, double const tol)
  GMRES PRECONDITIONED.

8.18.1 Detailed Description

GMRES IMPLEMENTATIONS.

Author:

Luis M. de la Cruz

Date:

Mon Nov 12 11:52:45 GMT 2007

Definition in file gmres.hpp.

8.19 /home/luiggi/Documents/Research/Meshless_RBF/NEW/RBFSoft/include/Traits.hpp File Reference

Definition of the variable types used in the whole TUNA::RBF.

Classes

- struct timer
  Tool for time measurements (borrowed from FLENS).

Typedefs

- typedef double prec_t
  Definition of the precision for all calculations.
- typedef DenseVector< Array< prec_t > > Vec
  Dense vectors from FLENS.
- typedef GeMatrix< FullStorage< prec_t, ColMajor > > Mat
  Dense matrix from FLENS.
- typedef DenseVector< Array< int > > iVec
  Integer dense vectors from FLENS.
- typedef GeMatrix< FullStorage< int, ColMajor > > iMat
  Integer dense matrix from FLENS.

8.19.1 Detailed Description

Definition of the variable types used in the whole TUNA::RBF.

In principle, this header should be included in all files of TUNA::RBF.

Definition in file Traits.hpp.