MTK: Mimetic Methods Toolkit

Generated by Doxygen 1.8.8

Tue Nov 24 2015 11:56:13

Contents

Chapter 1

Introduction

We define numerical methods that are based on discretizations preserving the properties of their continuum counterparts to be **mimetic**.

The **Mimetic Methods Toolkit (MTK)** is a C++ library for mimetic numerical methods. It is arranged as a set of classes for **mimetic quadratures**, mimetic interpolation$**$, and **mimetic finite differences** methods for the numerical solution of ordinary and partial differential equations.

An older version of this library is available outside of GitHub... just email me about it, and you can have it... it is ugly, yet functional and more complete.

1.1 MTK Concerns

Since collaborative development efforts are definitely important in achieving the level of generality we intend the library to possess, we have divided the library's source code according to the designated purpose the classes possess within the library. These divisions (or concerns) are grouped by layers, and are hierarchically related by the dependence they have among them.

One concern is said to depend on another one, if the classes the first concern includes, rely on the classes the second concern includes.

In order of dependence these are:

1. Roots.
2. Enumerations.
3. Tools.
4. Data Structures.
5. Numerical Methods.
6. Grids.
7. Mimetic Operators.

1.2 MTK Flavors

The MTK collection of wrappers is:

1. MMTK: MATLAB wrappers collection for MTK; intended for sequential computations.

Others are being designed and developed.

1.3 Contact, Support and Credits

The MTK is developed by researchers and adjuncts to the Computational Science Research Center (CSRC) at San Diego State University (SDSU).

Developers are members of:

1. Mimetic Numerical Methods Research and Development Group.
2. Computational Geoscience Research and Development Group.
3. Ocean Modeling Research and Development Group.

Currently the developers are:

1. **Eduardo J. Sanchez, Ph.D. - esanchez at mail dot sdsu dot edu** - ejspeiro
2. Jose E. Castillo, Ph.D. - jcastillo at mail dot sdsu dot edu
3. Guillermo F. Miranda, Ph.D. - unigrav at hotmail dot com
4. Christopher P. Paolini, Ph.D. - paolini at engineering dot sdsu dot edu
5. Angel Boada.
6. Johnny Corbino.
7. Raul Vargas–Navarro.

1.4 Acknowledgements and Contributions

The authors would like to acknowledge valuable advising, contributions and feedback, from research personnel at the Computational Science Research Center at San Diego State University, which were vital to the fruition of this work. Specifically, our thanks go to (alphabetical order):

1. Mohammad Abouali, Ph.D.
2. Dany De Cecchis, Ph.D.
3. Julia Rossi.

Chapter 2

Programming Tools

The development of MTK has been made possible through the use of the following applications:

1. Editor: Kate - KDE Advanced Text Editor. Version 3.13.3. Using KDE Development Platform 4.13.3 (C) 2000-2005 The Kate Authors.
2. Compiler: gcc version 4.4.5 (Ubuntu/Linaro 4.4.4-14ubuntu5). Copyright (C) 2013 Free Software Foundation, Inc.
3. Debugger: GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1. Copyright (C) 2014 Free Software Foundation, Inc.
4. Memory Profiler: valgrind-3.10.0.SVN.

Chapter 3

Licensing and Modifications

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu and a copy of the modified files should be reported once modifications are completed, unless these modifications are made through the project's GitHub page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications should be developed and included in any deliverable.
2. Redistributions of source code must be done through direct downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
3. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
4. Usage of the binary form on proprietary applications shall require explicit prior written permission from the the copyright holders, and due credit should be given to the copyright holders.
5. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

The copyright holders provide no reassurances that the source code provided does not infringe any patent, copyright, or any other intellectual property rights of third parties. The copyright holders disclaim any liability to any recipient for claims brought against recipient by any third party for infringement of that parties intellectual property rights.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Chapter 4

Read Me File and Installation Instructions

README File for the Mimetic Methods Toolkit (MTK)

By: **Eduardo J. Sanchez, Ph.D. – esanchez at mail dot sdsu dot edu**

1. Description

We define numerical methods that are based on discretizations preserving the properties of their continuum counterparts to be **mimetic**.

The **Mimetic Methods Toolkit (MTK)** is a C++ library for mimetic numerical methods. It is arranged as a set of classes for **mimetic quadratures**, **mimetic interpolation**, and **mimetic finite differences** methods for the numerical solution of ordinary and partial differential equations.

An older version of this library is available outside of GitHub... just email me about it, and you can have it... it is ugly, yet functional and more complete.

2. Dependencies

This README assumes all of these dependencies are installed in the following folder:

```
$(HOME)/Libraries/
```

In this version, the MTK optionally uses ATLAS-optimized BLAS and LAPACK routines for the internal computation on some of the layers. However, ATLAS requires both BLAS and LAPACK in order to create their optimized distributions. Therefore, the following dependencies tree arises:

For Linux:

1. LAPACK - Available from: http://www.netlib.org/lapack/
   1. BLAS - Available from: http://www.netlib.org/blas/

2. GLPK - Available from: https://www.gnu.org/software/glpk/

3. (Optional) ATLAS - Available from: http://math-atlas.sourceforge.net/
   1. LAPACK - Available from: http://www.netlib.org/lapack/
      1. BLAS - Available from: http://www.netlib.org/blas

4. (Optional) Valgrind - Available from: http://valgrind.org/

5. (Optional) Doxygen - Available from http://www.stack.nl/~dimitri/doxygen/

For OS X:

1. GLPK - Available from: https://www.gnu.org/software/glpk/

3. Installation

PART 1. CONFIGURATION OF THE MAKEFILE.

The following steps are required the build and test the MTK. Please use the accompanying Makefile.inc file, which should provide a solid template to start with. The following command provides help on the options for make:

```
$ make help
-----
Makefile for the MTK.

Options are:
- all: builds the library, the tests, and examples.
- mtklib: builds the library.
- test: builds the test files.
- example: builds the examples.

- testall: runs all the tests.

- gendoc: generates the documentation for the library.

- clean: cleans all the generated files.
- cleanlib: cleans the generated archive and object files.
- cleantest: cleans the generated tests executables.
- cleanexample: cleans the generated examples executables.
-----
```

PART 2. BUILD THE LIBRARY.

```
$ make
```

If successful you'll read (before building the examples):

```
----- Library created! Check in /home/ejspeiro/Dropbox/MTK/lib
```

Examples and tests will also be built.

4. Frequently Asked Questions

Q: Why haven't you guys implemented GBS to build the library?
A: I'm on it as we speak! ;)

Q: Is there any main reference when it comes to the theory on Mimetic Methods?
A: Yes! Check: http://www.csrc.sdsu.edu/mimetic-book

Q: Do I need to generate the documentation myself?
A: You can if you want to... but if you DO NOT want to, just go to our website.

5. Contact, Support, and Credits

The MTK is developed by researchers and adjuncts to the
Computational Science Research Center (CSRC)
at San Diego State University (SDSU).

Developers are members of:

1. Mimetic Numerical Methods Research and Development Group.
2. Computational Geoscience Research and Development Group.
3. Ocean Modeling Research and Development Group.

Currently the developers are:

1. **Eduardo J. Sanchez, Ph.D. - esanchez at mail dot sdsu dot edu** - ejspeiro
2. Jose E. Castillo, Ph.D. - jcastillo at mail dot sdsu dot edu
3. Guillermo F. Miranda, Ph.D. - unigrav at hotmail dot com
4. Christopher P. Paolini, Ph.D. - paolini at engineering dot sdsu dot edu
5. Angel Boada.
6. Johnny Corbino.
7. Raul Vargas-Navarro.

Finally, please feel free to contact me with suggestions or corrections:

Eduardo J. Sanchez, Ph.D. - esanchez at mail dot sdsu dot edu - ejspeiro

Thanks and happy coding!

Chapter 5

Tests and Test Architectures

Tests are given in the `files list` section. They are provided in the /tests/ folder within the distributed software.

In this page we intend to make a summary of all of the architectures in where the MTK has been tested. The MTK is intended to be as portable as possible throughout architectures. The following architectures have provided flawless installations of the API and correct execution of the examples:

1. Linux 3.2.0-23-generic-pae #36-Ubuntu SMP i386 GNU/Linux.
   Intel(R) Pentium(R) M CPU 1.73 GHz 2048 KB of cache and stepping of 8.
   gcc version 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5).

2. Linux 3.13.0-67-generic #110-Ubuntu SMP x86_64 GNU/Linux.
   Intel(R) Core(TM) i7-4700MQ CPU 2.40 GHz 6144 KB of cache and stepping of 3.
   gcc version 4.8.4 (Ubuntu 4.4.4-2ubuntu1~14.04).

3. Linux 3.16.7-29-desktop #1 SMP PREEMPT (6be6a97) x86_64 GNU/Linux
   Intel(R) Core(TM) i7-4600U CPU 2.10 GHz 4096 KB of cache and a stepping of 1.
   gcc (SUSE Linux) 4.8.3 20140627 [gcc-4_8-branch revision 212064].

Further architectures will be tested!

Chapter 6

Examples

Examples are given in the `files list` section. They are provided in the /examples/ folder within the distributed software.

Chapter 7

User Manual, References and Theory

The main source of references for this work can be found in:

http://www.csrc.sdsu.edu/mimetic-book/

However, a .PDF copy of this manual can be found here.

Chapter 8

Todo List

Member mtk::BCDescriptor2D::ImposeOnGrid (UniStgGrid2D &grid) const

Implement imposition for vector-valued grids.

Member mtk::BCDescriptor2D::ImposeOnSouthBoundaryNoSpace (const mtk::UniStgGrid2D &grid, mtk::DenseMatrix &matrix, const int &order_accuracy) const

Impose the Neumann conditions on every pole, for every scenario.

Member mtk::BCDescriptor2D::ImposeOnSouthBoundaryWithSpace (const mtk::UniStgGrid2D &grid, mtk::DenseMatrix &matrix, const int &order_accuracy) const

Impose Harmonic mean on the corners for the case when the generated space is available, for all poles.

Member mtk::DenseMatrix::Kron (const DenseMatrix &aa, const DenseMatrix &bb)

Implement Kronecker product using the BLAS.

Member mtk::DenseMatrix::OrderColMajor ()

Improve this so that no new arrays have to be created.

Member mtk::DenseMatrix::OrderRowMajor ()

Improve this so that no new arrays have to be created.

Member mtk::DenseMatrix::Transpose ()

Improve this so that no extra arrays have to be created.

Class mtk::GLPKAdapter

Rescind from the GLPK as the numerical core for CLO problems.

Member mtk::Matrix::IncreaseNumNull () noexcept

Review the definition of sparse matrices properties.

Member mtk::Matrix::IncreaseNumZero () noexcept

Review the definition of sparse matrices properties.

Member mtk::Tools::Prevent (const bool complement, const char *const fname, int lineno, const char *const fxname) noexcept

Check if this is the best way of stalling execution.

Member mtk::Tools::test_number_

Check usage of static methods and private members.

Member mtk::UniStgGrid1D::discrete_domain_x () const

Review const-correctness of the pointer we return.

Member mtk::UniStgGrid1D::discrete_field_u ()

Review const-correctness of the pointer we return. Look at the STL!

Member mtk::UniStgGrid2D::discrete_domain_x () const

Review const-correctness of the pointer we return.

Member mtk::UniStgGrid2D::discrete_domain_y () const

Review const-correctness of the pointer we return.

File mtk_div_1d.cc

Overload ostream operator as in mtk::Lap1D.

Implement creation of ■ w. mtk::BLASAdapter.

File mtk_glpk_adapter_test.cc

Test the mtk::GLPKAdapter class.

File mtk_grad_1d.cc

Overload ostream operator as in mtk::Lap1D.

Implement creation of ■ w. mtk::BLASAdapter.

File mtk_lapack_adapter.cc

Write documentation using LaTeX.

File mtk_lapack_adapter_test.cc

Test the mtk::LAPACKAdapter class.

File mtk_quad_1d.h

Implement this class.

File mtk_roots.h

Documentation should (better?) capture effects from selective compilation.

Test selective precision mechanisms.

File mtk_uni_stg_grid_1d.h

Create overloaded binding routines that read data from files.

File mtk_uni_stg_grid_2d.h

Create overloaded binding routines that read data from files.

Chapter 9

Bug List

Member mtk::Matrix::set_num_null (const int &in) noexcept

-nan assigned on construction time due to num_values_ being 0.

Member mtk::Matrix::set_num_zero (const int &in) noexcept

-nan assigned on construction time due to num_values_ being 0.

Chapter 10

Module Index

10.1 Modules

Here is a list of all modules:

Chapter 11

Namespace Index

11.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 12

Class Index

12.1 Class List

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

Chapter 13

File Index

13.1 File List

Here is a list of all files with brief descriptions:

Chapter 14

Module Documentation

14.1 Roots.

Fundamental execution parameters and defined types.

Typedefs

- typedef float mtk::Real

  Users can simply change this to build a double- or single-precision MTK.

Variables

- const float mtk::kZero {0.0f}

  MTK's zero defined according to selective compilation.
- const float mtk::kOne {1.0f}

  MTK's one defined according to selective compilation.
- const float mtk::kTwo {2.0f}

  MTK's two defined according to selective compilation.
- const float mtk::kDefaultTolerance {1e-7f}

  Considered tolerance for comparisons in numerical methods.
- const int mtk::kDefaultOrderAccuracy {2}

  Default order of accuracy for mimetic operators.
- const float mtk::kDefaultMimeticThreshold {1e-6f}

  Default tolerance for higher-order mimetic operators.
- const int mtk::kCriticalOrderAccuracyDiv {8}

  At this order (and higher) we must use the CBSA to construct.
- const int mtk::kCriticalOrderAccuracyGrad {10}

  At this order (and higher) we must use the CBSA to construct.

14.1.1 Detailed Description

Fundamental execution parameters and defined types.

14.1.2 Typedef Documentation

14.1.2.1 mtk::Real

Definition at line 83 of file mtk_roots.h.

14.1.3 Variable Documentation

14.1.3.1 mtk::kCriticalOrderAccuracyDiv {8}

Definition at line 167 of file mtk_roots.h.

14.1.3.2 mtk::kCriticalOrderAccuracyGrad {10}

Definition at line 176 of file mtk_roots.h.

14.1.3.3 mtk::kDefaultMimeticThreshold {1e-6f}

Warning

Declared as double if MTK_PRECISION_DOUBLE is defined.

Definition at line 157 of file mtk_roots.h.

14.1.3.4 mtk::kDefaultOrderAccuracy {2}

Warning

Declared as double if MTK_PRECISION_DOUBLE is defined.

Definition at line 143 of file mtk_roots.h.

14.1.3.5 mtk::kDefaultTolerance {1e-7f}

Definition at line 131 of file mtk_roots.h.

14.1.3.6 mtk::kOne {1.0f}

Warning

Declared as double if MTK_PRECISION_DOUBLE is defined.

Definition at line 117 of file mtk_roots.h.

14.1.3.7 mtk::kTwo {2.0f}

Warning

Declared as double if MTK_PRECISION_DOUBLE is defined.

Definition at line 118 of file mtk_roots.h.

14.1.3.8 mtk::kZero {0.0f}

Warning

Declared as double if MTK_PRECISION_DOUBLE is defined.

Definition at line 116 of file mtk_roots.h.

14.2 Enumerations.

Enumerations.

Enumerations

- enum mtk::MatrixStorage { mtk::DENSE, mtk::BANDED, mtk::CRS }

  Considered matrix storage schemes to implement sparse matrices.
- enum mtk::MatrixOrdering { mtk::ROW_MAJOR, mtk::COL_MAJOR }

  Considered matrix ordering (for Fortran purposes).
- enum mtk::FieldNature { mtk::SCALAR, mtk::VECTOR }

  Nature of the field discretized in a given grid.
- enum mtk::DirInterp { mtk::SCALAR_TO_VECTOR, mtk::VECTOR_TO_SCALAR }

  Interpolation operator.

14.2.1 Detailed Description

Enumerations.

14.2.2 Enumeration Type Documentation

14.2.2.1 enum mtk::DirInterp

Used to tag different directions of interpolation supported.

Enumerator

SCALAR_TO_VECTOR Interpolations places scalar on vectors' location.

VECTOR_TO_SCALAR Interpolations places vectors on scalars' location.

Definition at line 127 of file mtk_enums.h.

14.2.2.2 enum mtk::FieldNature

Fields can be **scalar** or **vector** in nature.

See also

https://en.wikipedia.org/wiki/Scalar_field
https://en.wikipedia.org/wiki/Vector_field

Enumerator

SCALAR Scalar-valued field.

VECTOR Vector-valued field.

Definition at line 113 of file mtk_enums.h.

14.2.2.3 enum mtk::MatrixOrdering

Row-major ordering is used for most application in C/C++. For Fortran purposes, the matrices must be listed in a column-major ordering.

See also

https://en.wikipedia.org/wiki/Row-major_order

Enumerator

ROW_MAJOR Row-major ordering (C/C++).

COL_MAJOR Column-major ordering (Fortran).

Definition at line 95 of file mtk_enums.h.

14.2.2.4 enum mtk::MatrixStorage

The considered sparse storage schemes are selected so that these are compatible with some of the most used mathematical APIs, as follows: DENSE and BANDED for BLAS, LAPACK, and ScaLAPACK. Finally, CRS for SuperLU.

Enumerator

DENSE Dense matrices, implemented as a 1D array: DenseMatrix.

BANDED Banded matrices ala LAPACK and ScaLAPACK: Must be implemented.

CRS Compressed-Rows Storage: Must be implemented.

Definition at line 77 of file mtk_enums.h.

14.3 Execution tools.

Tools to ensure execution correctness.

Classes

- class mtk::Tools

  Tool manager class.

14.3.1 Detailed Description

Tools to ensure execution correctness.

14.4 Data structures.

Fundamental data structures.

Classes

- class mtk::DenseMatrix

  Defines a common dense matrix, using a 1D array.
- class mtk::Matrix

  Definition of the representation of a matrix in the MTK.

14.4.1 Detailed Description

Fundamental data structures.

14.5 Numerical methods.

Adapter classes and auxiliary numerical methods.

Classes

- class mtk::BLASAdapter

  Adapter class for the BLAS API.
- class mtk::GLPKAdapter

  Adapter class for the GLPK API.
- class mtk::LAPACKAdapter

  Adapter class for the LAPACK API.

14.5.1 Detailed Description

Adapter classes and auxiliary numerical methods.

14.6 Grids.

Uniform rectangular staggered grids.

Classes

- class mtk::UniStgGrid1D

  Uniform 1D Staggered Grid.
- class mtk::UniStgGrid2D

  Uniform 2D Staggered Grid.

14.6.1 Detailed Description

Uniform rectangular staggered grids.

14.7 Mimetic operators.

Mimetic operators.

Classes

- class mtk::BCDescriptor2D

  Enforces boundary conditions in either the operator or the grid.
- class mtk::Div1D

  Implements a 1D mimetic divergence operator.
- class mtk::Div2D

  Implements a 2D mimetic divergence operator.
- class mtk::Grad1D

  Implements a 1D mimetic gradient operator.
- class mtk::Grad2D

  Implements a 2D mimetic gradient operator.
- class mtk::Interp1D

  Implements a 1D interpolation operator.
- class mtk::Interp2D

  Implements a 2D interpolation operator.
- class mtk::Lap1D

  Implements a 1D mimetic Laplacian operator.
- class mtk::Lap2D

  Implements a 2D mimetic Laplacian operator.
- class mtk::Quad1D

  Implements a 1D mimetic quadrature.
- class mtk::BCDescriptor1D

  Imposes boundary conditions on the operators or on the grids.

Typedefs

- typedef Real(∗ mtk::CoefficientFunction2D)(const Real &, const Real &)

  A function of a BC coefficient evaluated on a 2D domain.

14.7.1 Detailed Description

Mimetic operators.

14.7.2 Typedef Documentation

14.7.2.1 mtk::CoefficientFunction2D

Definition at line 98 of file mtk_bc_descriptor_2d.h.

Chapter 15

Namespace Documentation

15.1 mtk Namespace Reference

Mimetic Methods Toolkit namespace.

Classes

- class BCDescriptor1D

  Imposes boundary conditions on the operators or on the grids.
- class BCDescriptor2D

  Enforces boundary conditions in either the operator or the grid.
- class BLASAdapter

  Adapter class for the BLAS API.
- class DenseMatrix

  Defines a common dense matrix, using a 1D array.
- class Div1D

  Implements a 1D mimetic divergence operator.
- class Div2D

  Implements a 2D mimetic divergence operator.
- class GLPKAdapter

  Adapter class for the GLPK API.
- class Grad1D

  Implements a 1D mimetic gradient operator.
- class Grad2D

  Implements a 2D mimetic gradient operator.
- class Interp1D

  Implements a 1D interpolation operator.
- class Interp2D

  Implements a 2D interpolation operator.
- class Lap1D

  Implements a 1D mimetic Laplacian operator.
- class Lap2D

  Implements a 2D mimetic Laplacian operator.

- class LAPACKAdapter

  Adapter class for the LAPACK API.
- class Matrix

  Definition of the representation of a matrix in the MTK.
- class Quad1D

  Implements a 1D mimetic quadrature.
- class Tools

  Tool manager class.
- class UniStgGrid1D

  Uniform 1D Staggered Grid.
- class UniStgGrid2D

  Uniform 2D Staggered Grid.

Typedefs

- typedef Real(∗ CoefficientFunction2D)(const Real &, const Real &)

  A function of a BC coefficient evaluated on a 2D domain.
- typedef float Real

  Users can simply change this to build a double- or single-precision MTK.

Enumerations

- enum MatrixStorage { DENSE, BANDED, CRS }

  Considered matrix storage schemes to implement sparse matrices.
- enum MatrixOrdering { ROW_MAJOR, COL_MAJOR }

  Considered matrix ordering (for Fortran purposes).
- enum FieldNature { SCALAR, VECTOR }

  Nature of the field discretized in a given grid.
- enum DirInterp { SCALAR_TO_VECTOR, VECTOR_TO_SCALAR }

  Interpolation operator.

Functions

- float snrm2_ (int ∗n, float ∗x, int ∗incx)
- void saxpy_ (int ∗n, float ∗sa, float ∗sx, int ∗incx, float ∗sy, int ∗incy)
- void sgemv_ (char ∗trans, int ∗m, int ∗n, float ∗alpha, float ∗a, int ∗lda, float ∗x, int ∗incx, float ∗beta, float ∗y, int ∗incy)
- void sgemm_ (char ∗transa, char ∗transb, int ∗m, int ∗n, int ∗k, double ∗alpha, double ∗a, int ∗lda, double ∗b, aamm int ∗ldb, double ∗beta, double ∗c, int ∗ldc)
- std::ostream & operator<< (std::ostream &stream, mtk::DenseMatrix &in)
- std::ostream & operator<< (std::ostream &stream, mtk::Div1D &in)
- std::ostream & operator<< (std::ostream &stream, mtk::Grad1D &in)
- std::ostream & operator<< (std::ostream &stream, mtk::Interp1D &in)
- std::ostream & operator<< (std::ostream &stream, mtk::Lap1D &in)
- void sgesv_ (int ∗n, int ∗nrhs, Real ∗a, int ∗lda, int ∗ipiv, Real ∗b, int ∗ldb, int ∗info)
- void sgels_ (char ∗trans, int ∗m, int ∗n, int ∗nrhs, Real ∗a, int ∗lda, Real ∗b, int ∗ldb, Real ∗work, int ∗lwork, int ∗info)

Single-precision GEneral matrix Least Squares solver.

- void sgeqrf_ (int *m, int *n, Real *a, int *lda, Real *tau, Real *work, int *lwork, int *info)

  Single-precision GEneral matrix QR Factorization.

- void sormqr_ (char *side, char *trans, int *m, int *n, int *k, Real *a, int *lda, Real *tau, Real *c, int *ldc, Real *work, int *lwork, int *info)

  Single-precision Orthogonal Matrix from QR factorization.

- std::ostream & operator<< (std::ostream &stream, mtk::UniStgGrid1D &in)
- std::ostream & operator<< (std::ostream &stream, mtk::UniStgGrid2D &in)

Variables

- const float kZero {0.0f}

  MTK's zero defined according to selective compilation.

- const float kOne {1.0f}

  MTK's one defined according to selective compilation.

- const float kTwo {2.0f}

  MTK's two defined according to selective compilation.

- const float kDefaultTolerance {1e-7f}

  Considered tolerance for comparisons in numerical methods.

- const int kDefaultOrderAccuracy {2}

  Default order of accuracy for mimetic operators.

- const float kDefaultMimeticThreshold {1e-6f}

  Default tolerance for higher-order mimetic operators.

- const int kCriticalOrderAccuracyDiv {8}

  At this order (and higher) we must use the CBSA to construct.

- const int kCriticalOrderAccuracyGrad {10}

  At this order (and higher) we must use the CBSA to construct.

15.1.1 Function Documentation

15.1.1.1 std::ostream& mtk::operator<< (std::ostream & *stream,* mtk::Interp1D & *in*)

1. Print approximating coefficients for the interior.

Definition at line 66 of file mtk_interp_1d.cc.

15.1.1.2 std::ostream& mtk::operator<< (std::ostream & *stream,* mtk::UniStgGrid2D & *in*)

1. Print spatial coordinates.
2. Print scalar field.

Definition at line 67 of file mtk_uni_stg_grid_2d.cc.

15.1.1.3 std::ostream& mtk::operator<< (std::ostream & *stream,* mtk::UniStgGrid1D & *in*)

1. Print spatial coordinates.
2. Print scalar field.

Definition at line 68 of file mtk_uni_stg_grid_1d.cc.

15.1.1.4 std::ostream& mtk::operator<< (std::ostream & *stream,* mtk::Lap1D & *in*)

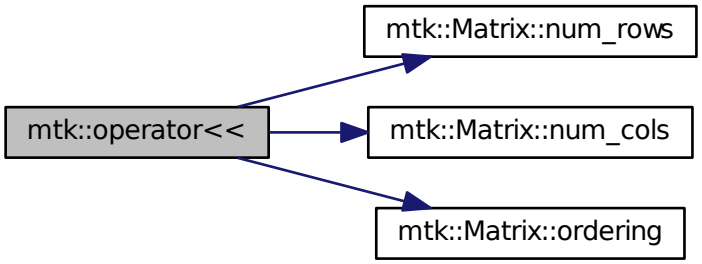
1. Print order of accuracy.
2. Print approximating coefficients for the interior.
3. No weights, thus print the mimetic boundary coefficients.

Definition at line 73 of file mtk_lap_1d.cc.

15.1.1.5 std::ostream& mtk::operator<< (std::ostream & *stream,* mtk::DenseMatrix & *in*)

Definition at line 77 of file mtk_dense_matrix.cc.

Here is the call graph for this function:



15.1.1.6 std::ostream& mtk::operator<< (std::ostream & *stream,* mtk::Grad1D & *in*)

1. Print order of accuracy.
2. Print approximating coefficients for the interior.
3. Print mimetic weights.
4. Print mimetic approximations at the boundary.

Definition at line 79 of file mtk_grad_1d.cc.

15.1.1.7 std::ostream& mtk::operator<< (std::ostream & *stream,* mtk::Div1D & *in*)

1. Print order of accuracy.
2. Print approximating coefficients for the interior.
3. Print mimetic weights.
4. Print mimetic approximations at the boundary.

Definition at line 79 of file mtk_div_1d.cc.

15.1.1.8 void mtk::saxpy_ (int * *n,* float * *sa,* float * *sx,* int * *incx,* float * *sy,* int * *incy*)

Here is the caller graph for this function:

15.1.1.9 void mtk::sgels_ (char * *trans,* int * *m,* int * *n,* int * *nrhs,* Real * *a,* int * *lda,* Real * *b,* int * *ldb,* Real * *work,* int * *lwork,* int * *info*)

SGELS solves overdetermined or underdetermined real linear systems involving an M-by-N matrix A, or its transpose, using a QR or LQ factorization of A. It is assumed that A has full rank.

The following options are provided:

1. If TRANS = 'N' and m >= n: find the least squares solution of an overdetermined system, i.e., solve the least squares problem

```
minimize || B - A*X ||.
```

2. If TRANS = 'N' and m < n: find the minimum norm solution of an underdetermined system A * X = B.
3. If TRANS = 'T' and m >= n: find the minimum norm solution of an undetermined system A**T * X = B.
4. If TRANS = 'T' and m < n: find the least squares solution of an overdetermined system, i.e., solve the least squares problem

```
minimize || B - A**T * X ||.
```

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

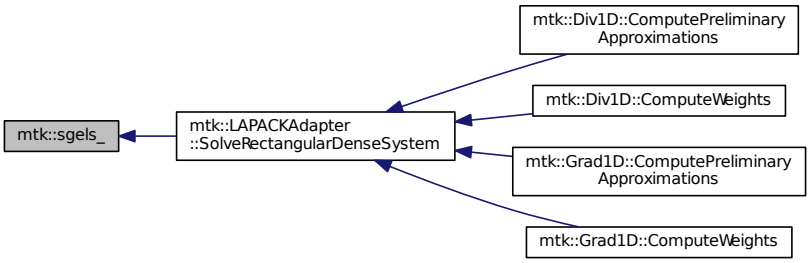
See also

http://www.math.utah.edu/software/lapack/lapack-s/sgels.html

Parameters

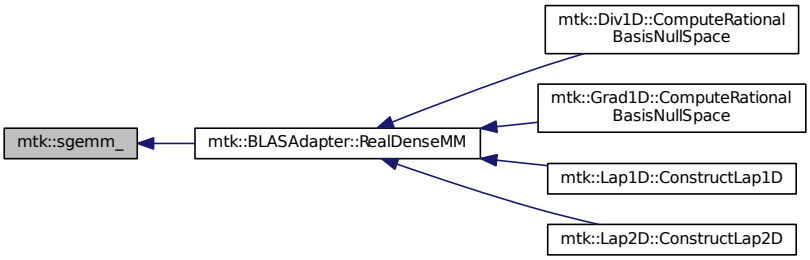
in	*trans*	Am I giving the transpose of the matrix?
in	*m*	The number of rows of the matrix a. m >= 0.
in	*n*	The number of columns of the matrix a. n >= 0.
in	*nrhs*	The number of right-hand sides.
in,out	*a*	On entry, the m-by-n matrix a.
in	*lda*	The leading dimension of a. lda >= max(1,m).
in,out	*b*	On entry, matrix b of right-hand side vectors.
in	*ldb*	The leading dimension of b. ldb >= max(1,m,n).
in,out	*work*	On exit, if info = 0, work(1) is optimal lwork.
in,out	*lwork*	The dimension of the array work.
in,out	*info*	If info = 0, then successful exit.

Here is the caller graph for this function:



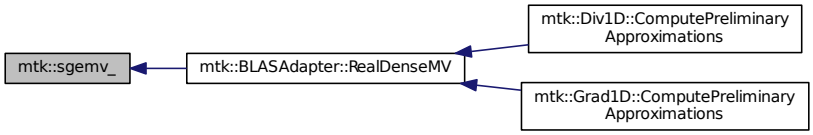
15.1.1.10 void mtk::sgemm_ (char * *transa*, char * *transb*, int * *m*, int * *n*, int * *k*, double * *alpha*, double * *a*, int * *lda*, double * *b*, aamm int * *ldb*, double * *beta*, double * *c*, int * *ldc*)

Here is the caller graph for this function:



15.1.1.11 void mtk::sgemv_ (char * *trans*, int * *m*, int * *n*, float * *alpha*, float * *a*, int * *lda*, float * *x*, int * *incx*, float * *beta*, float * *y*, int * *incy*)

Here is the caller graph for this function:



15.1.1.12 void mtk::sgeqrf_ (int * *m*, int * *n*, Real * *a*, int * *lda*, Real * *tau*, Real * *work*, int * *lwork*, int * *info*)

Single-Precision Orthogonal Make Q from QR: dormqr_ overwrites the general real M-by-N matrix C with (Table 1):

```
SIDE = 'L'      SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q**T * C C * Q**T
where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by SGEQRF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

See also

http://www.netlib.org/lapack/explore-html/df/d97/sgeqrf_8f.html

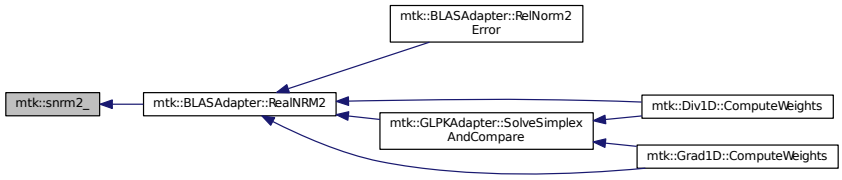
Parameters

in	*m*	The number of columns of the matrix a. n >= 0.
in	*n*	The number of columns of the matrix a. n >= 0.
in,out	*a*	On entry, the n-by-n matrix a.
in	*lda*	Leading dimension matrix. LDA >= max(1,M).
in,out	*tau*	Scalars from elementary reflectors. min(M,N).
in,out	*work*	Workspace. info = 0, work(1) is optimal lwork.
in	*lwork*	The dimension of work. lwork >= max(1,n).
in	*info*	info = 0: successful exit.

15.1.1.13 void mtk::sgesv_ (int * *n*, int * *nrhs*, Real * *a*, int * *lda*, int * *ipiv*, Real * *b*, int * *ldb*, int * *info*)

15.1.1.14 float mtk::snrm2_ (int * *n*, float * *x*, int * *incx*)

Here is the caller graph for this function:



15.1.1.15 void mtk::sormqr_ (char * *side*, char * *trans*, int * *m*, int * *n*, int * *k*, Real * *a*, int * *lda*, Real * *tau*, Real * *c*, int * *ldc*, Real * *work*, int * *lwork*, int * *info*)

Single-Precision Orthogonal Make Q from QR: sormqr_ overwrites the general real M-by-N matrix C with (Table 1):

```
SIDE = 'L'      SIDE = 'R'
```

TRANS = 'N': Q * C C * Q TRANS = 'T': Q**T * C C * Q**T
where Q is a real orthogonal matrix defined as the product of k elementary reflectors

```
Q = H(1) H(2) . . . H(k)
```

as returned by SGEQRF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

See also

http://www.netlib.org/lapack/explore-html/d0/d98/sormqr_8f_source.html

Parameters

in	*side*	See Table 1 above.
in	*trans*	See Table 1 above.
in	*m*	Number of rows of the C matrix.
in	*n*	Number of columns of the C matrix.
in	*k*	Number of reflectors.
in,out	*a*	The matrix containing the reflectors.
in	*lda*	The dimension of work. lwork >= max(1,n).
in	*tau*	Scalar factors of the elementary reflectors.
in	*c*	Output matrix.
in	*ldc*	Leading dimension of the output matrix.
in,out	*work*	Workspace. info = 0, work(1) optimal lwork.
in	*lwork*	The dimension of work.

in,out	*info*	info = 0: successful exit.

Chapter 16

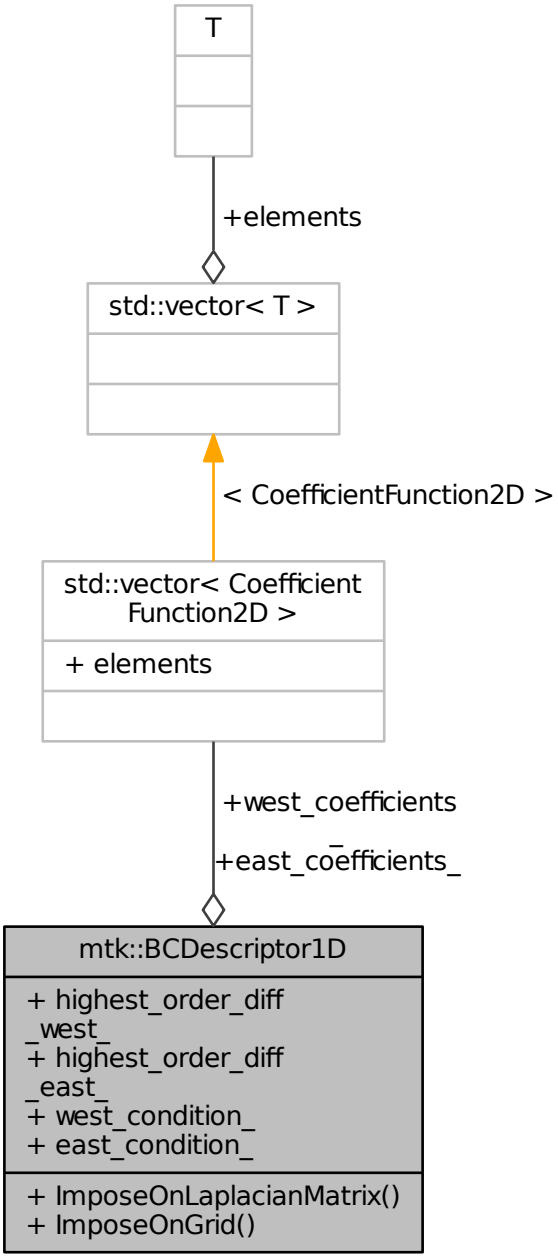
Class Documentation

16.1 mtk::BCDescriptor1D Class Reference

Imposes boundary conditions on the operators or on the grids.

`#include <mtk_robin_bc_descriptor_1d.h>`

Collaboration diagram for mtk::BCDescriptor1D:



Static Public Member Functions

- static void ImposeOnLaplacianMatrix (DenseMatrix &matrix, const std::vector< Real > &west, const std::vector< Real > &east)

Enforces the condition on the Laplacian represented as matrix.

- static void ImposeOnGrid (UniStgGrid1D &grid, const Real &epsilon, const Real &omega)

  Enforces the condition on the grid.

Public Attributes

- int highest_order_diff_west_

  Highest order of differentiation for west.
- int highest_order_diff_east_

  Highest order of differentiation for east.
- std::vector < CoefficientFunction2D > west_coefficients_

  Coeffs. west.
- std::vector < CoefficientFunction2D > east_coefficients_

  Coeffs. east.
- Real(∗ west_condition_)(Real xx, Real yy)

  Condition for west.
- Real(∗ east_condition_)(Real xx, Real yy)

  Condition for east.

16.1.1 Detailed Description

This class presents an interface for the user to specify boundary conditions on 1D mimetic operators and the grids they are acting on.

Def. Let f be any scalar or vector field defined over a domain Ω. We can specify any linear combination of f and its n derivatives to fulfill a condition, which we define as a **boundary condition**:

$$\forall \mathbf{x} \in \partial\Omega : \sum_{i=0}^{n} c_i(\mathbf{x}) < \hat{\mathbf{n}}, \frac{\partial^i f}{\partial x^i}(\mathbf{x}) >= \beta(\mathbf{x}).$$

This class receives information about all possible coefficient functions, $c_i(\mathbf{x})$ for any subset of the boundary (west and east), and each condition for any subset of the boundary, and takes care of assigning them to both, the differentiation matrices and the grids. The highest order of differentiation is computed based on how many coefficients have been given at run-time.

Definition at line 128 of file mtk_robin_bc_descriptor_1d.h.

16.1.2 Member Function Documentation

16.1.2.1 void mtk::BCDescriptor1D::ImposeOnGrid (mtk::UniStgGrid1D & *grid*, const Real & *epsilon*, const Real & *omega*) [static]

Parameters

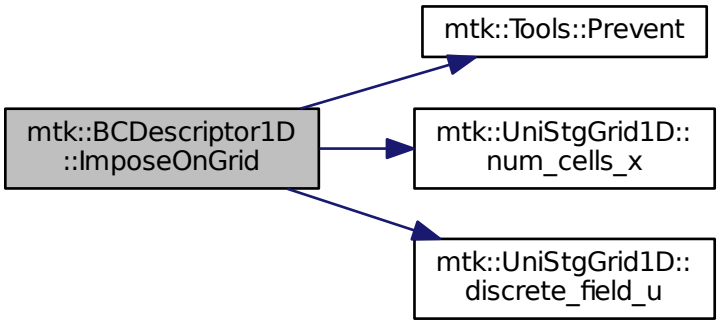
in,out	*grid*	Input grid.
in	*epsilon*	Actual BC for the east.
in	*omega*	Actual BC for the west.

1. Assign the west condition.

2. Assign the east condition.

Definition at line 89 of file mtk_bc_descriptor_1d.cc.

Here is the call graph for this function:



16.1.2.2 void mtk::BCDescriptor1D::ImposeOnLaplacianMatrix (mtk::DenseMatrix & *matrix*, const std::vector< Real > & *west*, const std::vector< Real > & *east*) [static]

Parameters

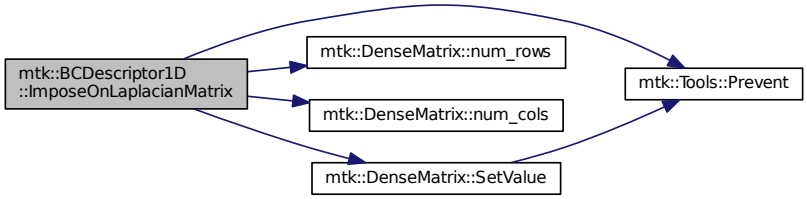
in,out	*matrix*	Input operator.
in	*west*	Array of values for the west boundary.
in	*east*	Array of values for the east boundary.

1. Assign the west array.

2. Assign the east array.

Definition at line 61 of file mtk_bc_descriptor_1d.cc.

Here is the call graph for this function:



16.1.3 Member Data Documentation

16.1.3.1 std::vector<CoefficientFunction2D> mtk::BCDescriptor1D::east_coefficients_

Definition at line 157 of file mtk_robin_bc_descriptor_1d.h.

16.1.3.2 Real(* mtk::BCDescriptor1D::east_condition_)(Real xx, Real yy)

Definition at line 160 of file mtk_robin_bc_descriptor_1d.h.

16.1.3.3 int mtk::BCDescriptor1D::highest_order_diff_east_

Definition at line 154 of file mtk_robin_bc_descriptor_1d.h.

16.1.3.4 int mtk::BCDescriptor1D::highest_order_diff_west_

Definition at line 153 of file mtk_robin_bc_descriptor_1d.h.

16.1.3.5 std::vector<CoefficientFunction2D> mtk::BCDescriptor1D::west_coefficients_

Definition at line 156 of file mtk_robin_bc_descriptor_1d.h.

16.1.3.6 Real(* mtk::BCDescriptor1D::west_condition_)(Real xx, Real yy)

Definition at line 159 of file mtk_robin_bc_descriptor_1d.h.

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

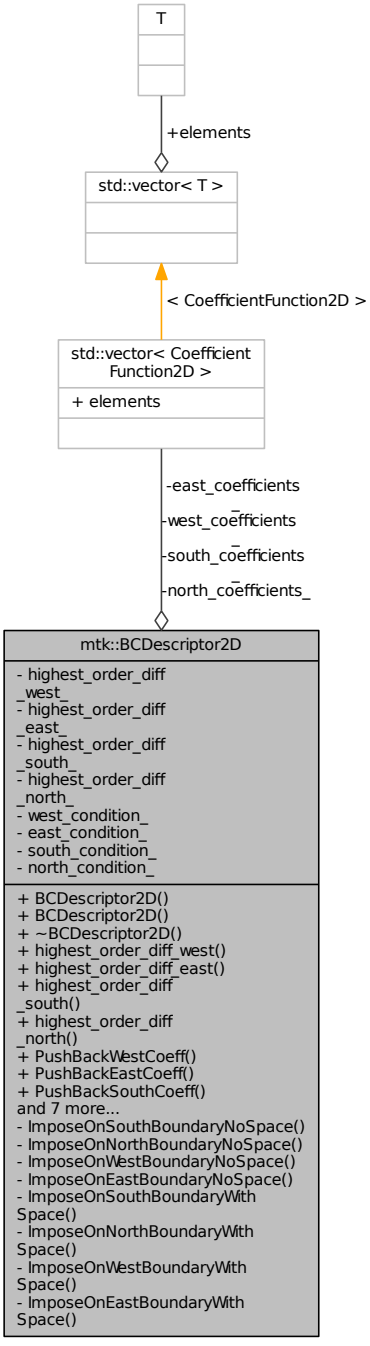
- include/mtk_robin_bc_descriptor_1d.h
- src/mtk_bc_descriptor_1d.cc

16.2 mtk::BCDescriptor2D Class Reference

Enforces boundary conditions in either the operator or the grid.

`#include <mtk_bc_descriptor_2d.h>`

Collaboration diagram for mtk::BCDescriptor2D:



Public Member Functions

- BCDescriptor2D ()

  Default constructor.
- BCDescriptor2D (const BCDescriptor2D &desc)

  Copy constructor.
- ∼BCDescriptor2D () noexcept

  Destructor.
- int highest_order_diff_west () const noexcept

  Getter for the highest order of differentiation in the west boundary.
- int highest_order_diff_east () const noexcept

  Getter for the highest order of differentiation in the east boundary.
- int highest_order_diff_south () const noexcept

  Getter for the highest order of differentiation in the south boundary.
- int highest_order_diff_north () const noexcept

  Getter for the highest order of differentiation in the north boundary.
- void PushBackWestCoeff (CoefficientFunction2D cw)

  Push back coefficient function at west of lowest order diff. available.
- void PushBackEastCoeff (CoefficientFunction2D ce)

  Push back coefficient function at east of lowest order diff. available.
- void PushBackSouthCoeff (CoefficientFunction2D cs)

  Push back coefficient function south of lowest order diff. available.
- void PushBackNorthCoeff (CoefficientFunction2D cn)

  Push back coefficient function north of lowest order diff. available.
- void set_west_condition (Real(∗west_condition)(Real xx, Real yy)) noexcept

  Set boundary condition at west.
- void set_east_condition (Real(∗east_condition)(Real xx, Real yy)) noexcept

  Set boundary condition at east.
- void set_south_condition (Real(∗south_condition)(Real xx, Real yy)) noexcept

  Set boundary condition at south.
- void set_north_condition (Real(∗north_condition)(Real xx, Real yy)) noexcept

  Set boundary condition at north.
- void ImposeOnLaplacianMatrix (const UniStgGrid2D &grid, DenseMatrix &matrix, const int &order_accuracy=2) const

  Imposes the condition on the operator represented as matrix.
- void ImposeOnGrid (UniStgGrid2D &grid) const

  Imposes the condition on the grid.

Private Member Functions

- void ImposeOnSouthBoundaryNoSpace (const mtk::UniStgGrid2D &grid, mtk::DenseMatrix &matrix, const int &order_accuracy) const

  Imposes the condition on the south boundary.
- void ImposeOnNorthBoundaryNoSpace (const mtk::UniStgGrid2D &grid, mtk::DenseMatrix &matrix, const int &order_accuracy) const

  Imposes the condition on the north boundary.
- void ImposeOnWestBoundaryNoSpace (const mtk::UniStgGrid2D &grid, mtk::DenseMatrix &matrix, const int &order_accuracy) const

Imposes the condition on the west boundary.

- void ImposeOnEastBoundaryNoSpace (const mtk::UniStgGrid2D &grid, mtk::DenseMatrix &matrix, const int &order_accuracy) const

  Imposes the condition on the east boundary.

- void ImposeOnSouthBoundaryWithSpace (const mtk::UniStgGrid2D &grid, mtk::DenseMatrix &matrix, const int &order_accuracy) const

  Imposes the condition on the south boundary.

- void ImposeOnNorthBoundaryWithSpace (const mtk::UniStgGrid2D &grid, mtk::DenseMatrix &matrix, const int &order_accuracy) const

  Imposes the condition on the north boundary.

- void ImposeOnWestBoundaryWithSpace (const mtk::UniStgGrid2D &grid, mtk::DenseMatrix &matrix, const int &order_accuracy) const

  Imposes the condition on the west boundary.

- void ImposeOnEastBoundaryWithSpace (const mtk::UniStgGrid2D &grid, mtk::DenseMatrix &matrix, const int &order_accuracy) const

  Imposes the condition on the east boundary.

Private Attributes

- int highest_order_diff_west_

  Highest order of differentiation for west.

- int highest_order_diff_east_

  Highest order of differentiation for east.

- int highest_order_diff_south_

  Highest order differentiation for south.

- int highest_order_diff_north_

  Highest order differentiation for north.

- std::vector
  < CoefficientFunction2D > west_coefficients_

  Coeffs. west.

- std::vector
  < CoefficientFunction2D > east_coefficients_

  Coeffs. east.

- std::vector
  < CoefficientFunction2D > south_coefficients_

  Coeffs. south.

- std::vector
  < CoefficientFunction2D > north_coefficients_

  Coeffs. south.

- Real(* west_condition_)(Real xx, Real yy)

  Condition for west.

- Real(* east_condition_)(Real xx, Real yy)

  Condition for east.

- Real(* south_condition_)(Real xx, Real yy)

  Condition for south.

- Real(* north_condition_)(Real xx, Real yy)

  Condition for north.

16.2.1 Detailed Description

This class presents an interface for the user to specify boundary conditions on 2D mimetic operators and the grids they are acting on.

Def. Let f be any scalar or vector field defined over a domain Ω. We can specify any linear combination of f and its n derivatives to fulfill a condition, which we define as a **boundary condition**:

$$\forall \mathbf{x} \in \partial\Omega : \sum_{i=0}^{n} c_i(\mathbf{x}) < \hat{\mathbf{n}}, \frac{\partial^i f}{\partial x^i}(\mathbf{x}) >= \beta(\mathbf{x}).$$

This class receives information about all possible coefficient functions, $c_i(\mathbf{x})$ for any subset of the boundary (south, north, west and east), and each condition for any subset of the boundary, and takes care of assigning them to both, the differentiation matrices and the grids. The highest order of differentiation is computed based on how many coefficients have been given at run-time.

Definition at line 130 of file mtk_bc_descriptor_2d.h.

16.2.2 Constructor & Destructor Documentation

16.2.2.1 mtk::BCDescriptor2D::BCDescriptor2D ( )

Definition at line 80 of file mtk_bc_descriptor_2d.cc.

16.2.2.2 mtk::BCDescriptor2D::BCDescriptor2D (const BCDescriptor2D & *desc*)

Parameters

in	*desc*	Given 2D descriptor.

Definition at line 90 of file mtk_bc_descriptor_2d.cc.

16.2.2.3 mtk::BCDescriptor2D::~BCDescriptor2D ( ) [noexcept]

Definition at line 92 of file mtk_bc_descriptor_2d.cc.

16.2.3 Member Function Documentation

16.2.3.1 int mtk::BCDescriptor2D::highest_order_diff_east ( ) const [noexcept]

Returns

Integer highest order of differentiation in the east boundary.

Definition at line 99 of file mtk_bc_descriptor_2d.cc.

16.2.3.2 int mtk::BCDescriptor2D::highest_order_diff_north ( ) const [noexcept]

Returns

Integer highest order of differentiation in the north boundary.

Definition at line 109 of file mtk_bc_descriptor_2d.cc.

16.2.3.3 int mtk::BCDescriptor2D::highest_order_diff_south ( ) const [noexcept]

Returns

Integer highest order of differentiation in the south boundary.

Definition at line 104 of file mtk_bc_descriptor_2d.cc.

16.2.3.4 int mtk::BCDescriptor2D::highest_order_diff_west ( ) const [noexcept]

Returns

Integer highest order of differentiation in the west boundary.

Definition at line 94 of file mtk_bc_descriptor_2d.cc.

16.2.3.5 void mtk::BCDescriptor2D::ImposeOnEastBoundaryNoSpace (const mtk::UniStgGrid2D & *grid*, mtk::DenseMatrix & *matrix*, const int & *order_accuracy*) const [private]

Parameters

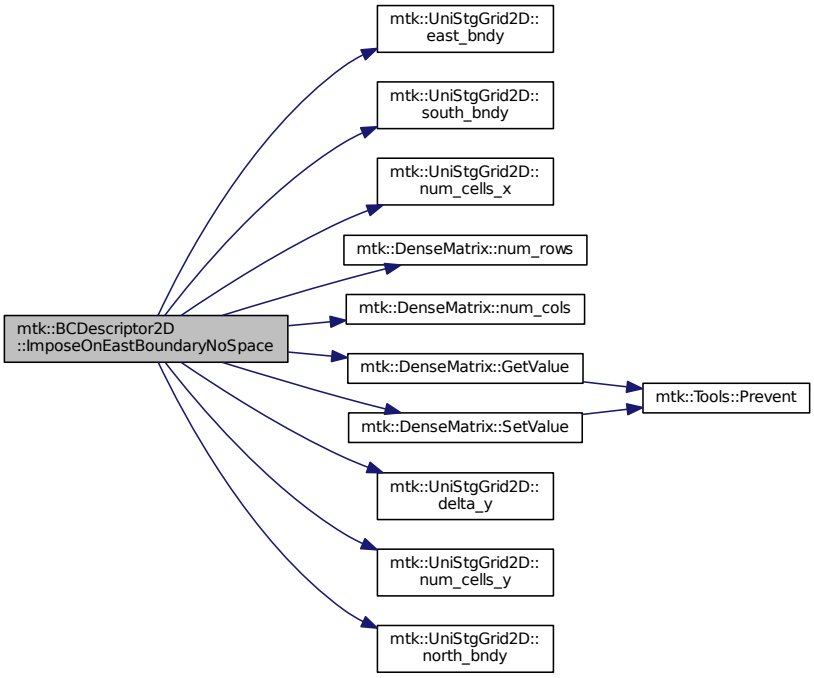
in	*grid*	Grid upon which impose the desired boundary condition.
in,out	*matrix*	Input Laplacian operator.
in	*order_accuracy*	Order of accuracy of the operator in the Matrix.

1. Impose the Dirichlet condition first.

2. Impose the Neumann condition second.

Definition at line 466 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



16.2.3.6 void mtk::BCDescriptor2D::ImposeOnEastBoundaryWithSpace (const mtk::UniStgGrid2D & *grid*, mtk::DenseMatrix & *matrix*, const int & *order_accuracy*) const [private]

Parameters

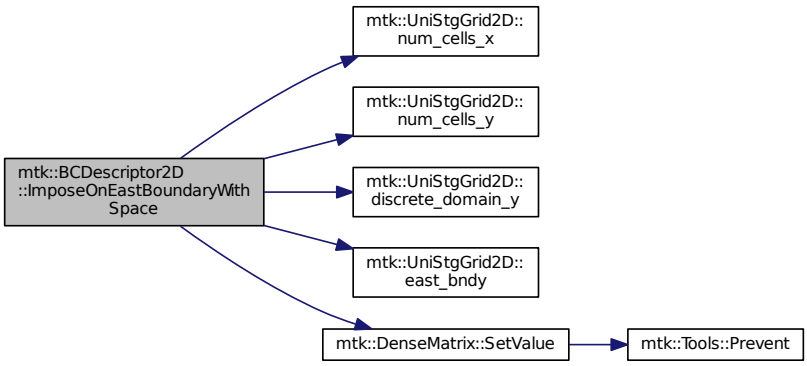
in	*grid*	Grid upon which impose the desired boundary condition.
in,out	*matrix*	Input Laplacian operator.
in	*order_accuracy*	Order of accuracy of the operator in the Matrix.

1. Impose the Dirichlet condition first.

2. Impose the Neumann condition second.

Definition at line 536 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



16.2.3.7 void mtk::BCDescriptor2D::ImposeOnGrid (mtk::UniStgGrid2D & *grid*) const

Parameters

in,out	*grid*	Grid upon which impose the desired boundary condition.

1. Impose assuming an scalar grid.

1.1. Impose south condition.

1.1.1. Impose south-west corner.

1.1.2. Impose south border.

1.1.3. Impose south-east corner.

1.2. Impose north condition.

1.2.1. Impose north-west corner.

1.2.2. Impose north border.

1.2.3. Impose north-east corner.

1.3. Impose west condition.

1.3.1. Impose south-west corner.

Note

As per discussion with Otilio, we will take the **arithmetic mean** of the values of the BCs at the corners.

1.3.2. Impose west border.

1.3.3. Impose north-west corner.

1.4. Impose east condition.

1.4.1. Impose south-east corner.

1.4.2. Impose east border.

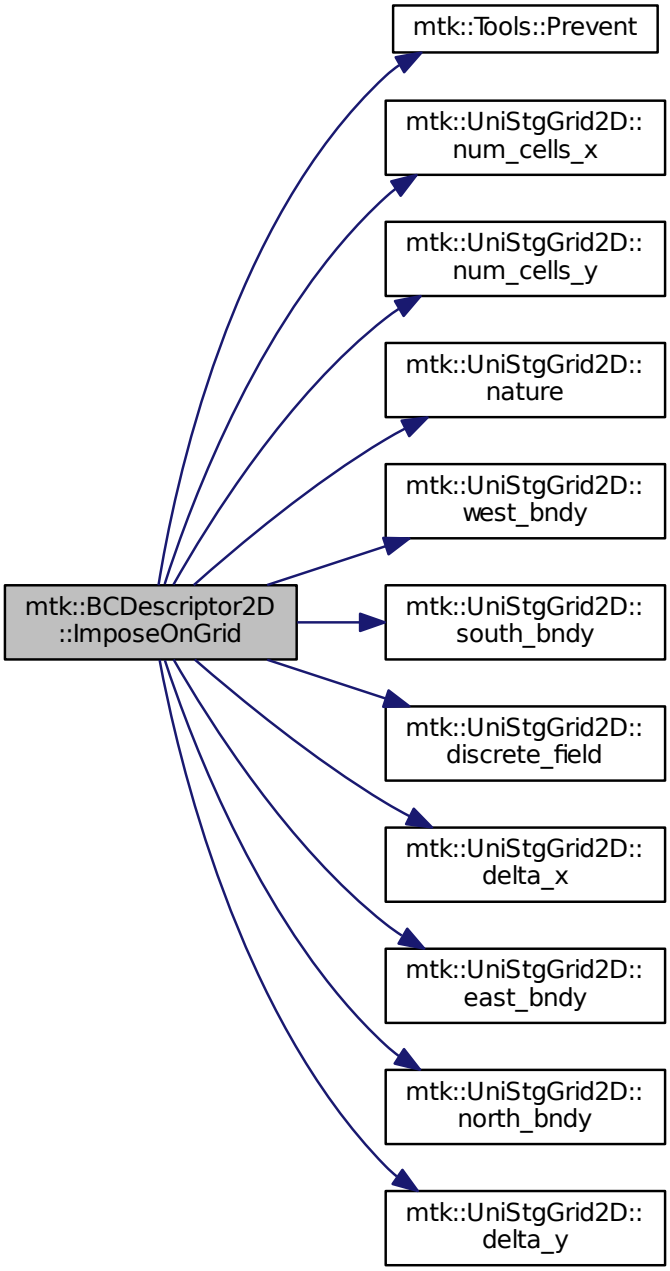
1.4.3. Impose north-east corner.

1. Impose assuming a vector grid.

Todo Implement imposition for vector-valued grids.

Definition at line 595 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



16.2.3.8 void mtk::BCDescriptor2D::ImposeOnLaplacianMatrix (const **UniStgGrid2D** & *grid,* **mtk::DenseMatrix** & *matrix,* const int & *order_accuracy* = 2) const

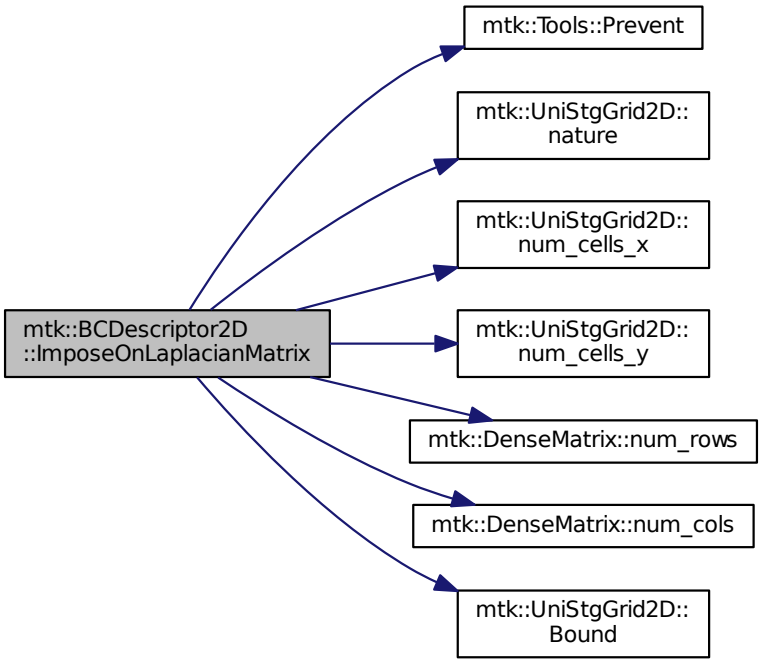
Parameters

in	*grid*	Grid upon which impose the desired boundary condition.
in,out	*matrix*	Input Laplacian operator.
in	*order_accuracy*	Order of accuracy of the operator in the Matrix.

If we have not bound anything to the grid, then we have to generate our collection of spatial coordinates, as we evaluate the coefficients.

Definition at line 557 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



16.2.3.9 void mtk::BCDescriptor2D::ImposeOnNorthBoundaryNoSpace (const mtk::UniStgGrid2D & *grid*, mtk::DenseMatrix & *matrix*, const int & *order_accuracy*) const [private]

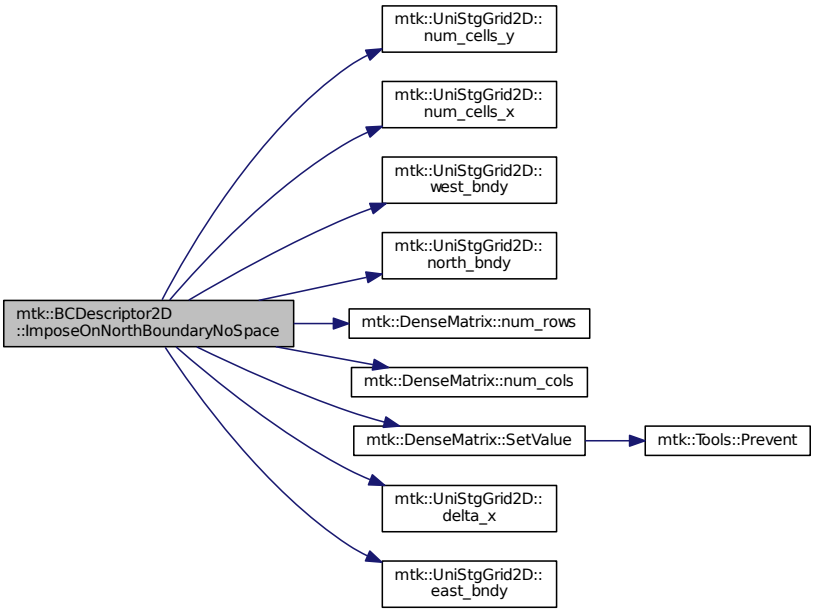
Parameters

in	*grid*	Grid upon which impose the desired boundary condition.
in,out	*matrix*	Input Laplacian operator.
in	*order_accuracy*	Order of accuracy of the operator in the Matrix.

1. Impose the Dirichlet condition first.
2. Impose the Neumann condition second.

Definition at line 290 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



16.2.3.10 void mtk::BCDescriptor2D::ImposeOnNorthBoundaryWithSpace (const mtk::UniStgGrid2D & *grid*, mtk::DenseMatrix & *matrix*, const int & *order_accuracy*) const [private]

Parameters

in	*grid*	Grid upon which impose the desired boundary condition.
in,out	*matrix*	Input Laplacian operator.
in	*order_accuracy*	Order of accuracy of the operator in the Matrix.

1. Impose Dirichlet condition.

For each entry on the diagonal:

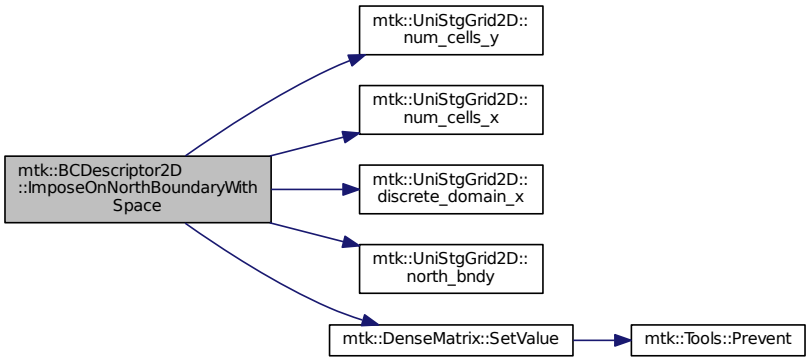
Evaluate next set spatial coordinates to evaluate the coefficient.

Evaluate and assign the Dirichlet coefficient.

1. Impose the Neumann condition.

Definition at line 354 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



16.2.3.11 void mtk::BCDescriptor2D::ImposeOnSouthBoundaryNoSpace (const mtk::UniStgGrid2D & *grid*, mtk::DenseMatrix & *matrix*, const int & *order_accuracy*) const [private]

Parameters

in	*grid*	Grid upon which impose the desired boundary condition.
in,out	*matrix*	Input Laplacian operator.
in	*order_accuracy*	Order of accuracy of the operator in the Matrix.

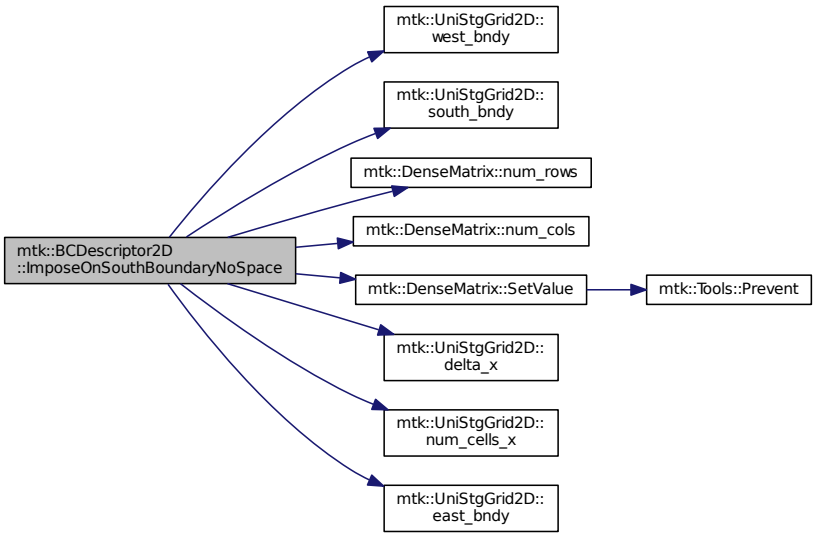
1. Impose the Dirichlet condition first.

2. Impose the Neumann condition second.

Todo Impose the Neumann conditions on every pole, for every scenario.

Definition at line 208 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



16.2.3.12 void mtk::BCDescriptor2D::ImposeOnSouthBoundaryWithSpace (const mtk::UniStgGrid2D & *grid*, mtk::DenseMatrix & *matrix*, const int & *order_accuracy*) const [private]

Parameters

in	*grid*	Grid upon which impose the desired boundary condition.
in,out	*matrix*	Input Laplacian operator.
in	*order_accuracy*	Order of accuracy of the operator in the Matrix.

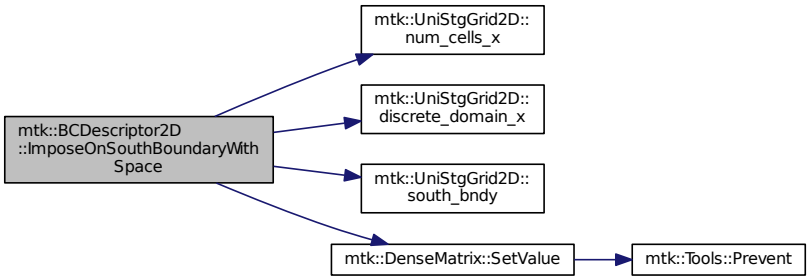
1. Impose the Dirichlet condition first.

Todo Impose Harmonic mean on the corners for the case when the generated space is available, for all poles.

1. Impose the Neumann condition second.

Definition at line 268 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



16.2.3.13 void mtk::BCDescriptor2D::ImposeOnWestBoundaryNoSpace (const mtk::UniStgGrid2D & *grid,* mtk::DenseMatrix & *matrix,* const int & *order_accuracy*) const [private]

Parameters

in	*grid*	Grid upon which impose the desired boundary condition.
in,out	*matrix*	Input Laplacian operator.
in	*order_accuracy*	Order of accuracy of the operator in the Matrix.

1. Impose the Dirichlet condition first.

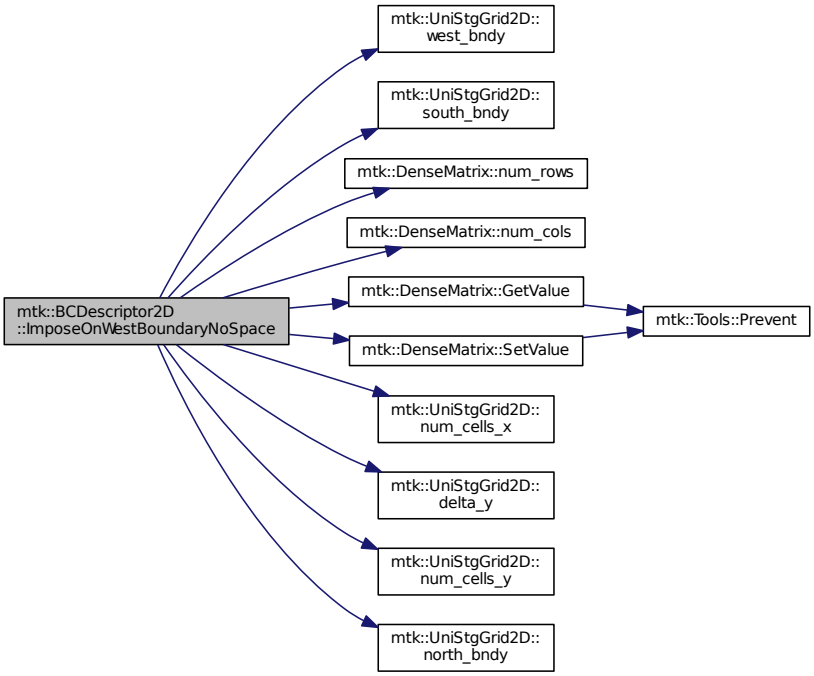
Note

As it can be seen, we must adopt a convention about how to treat the corners. Based on a reasoning with Otilio, we will take the **harmonic mean**.

1. Impose the Neumann condition second.

Definition at line 375 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



16.2.3.14 void mtk::BCDescriptor2D::ImposeOnWestBoundaryWithSpace (const mtk::UniStgGrid2D & *grid*, mtk::DenseMatrix & *matrix*, const int & *order_accuracy*) const [private]

Parameters

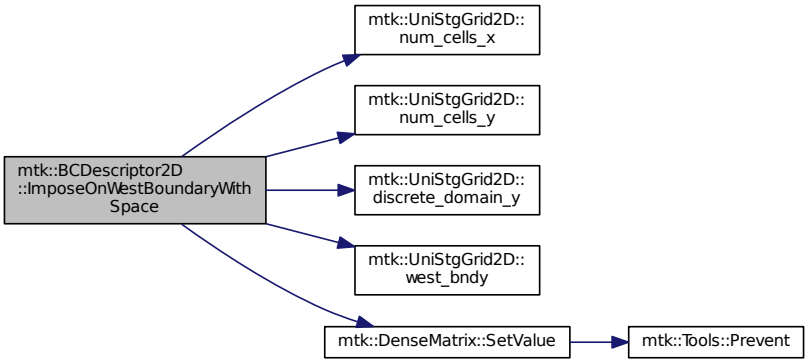
in	*grid*	Grid upon which impose the desired boundary condition.
in,out	*matrix*	Input Laplacian operator.
in	*order_accuracy*	Order of accuracy of the operator in the Matrix.

1. Impose the Dirichlet condition first.

2. Impose the Neumann condition second.

Definition at line 445 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



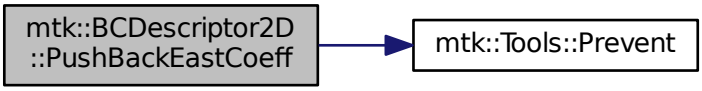
16.2.3.15 void mtk::BCDescriptor2D::PushBackEastCoeff (mtk::CoefficientFunction2D *ce*)

Parameters

in	*ce*	Function $c_e(x,y) : \Omega \mapsto \mathbb{R}$.

Definition at line 127 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



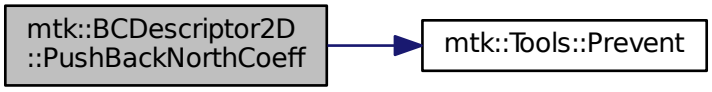
16.2.3.16 void mtk::BCDescriptor2D::PushBackNorthCoeff (mtk::CoefficientFunction2D *cn*)

Parameters

in	*cn*	Function $c_n(x,y) : \Omega \mapsto \mathbb{R}$.

Definition at line 153 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



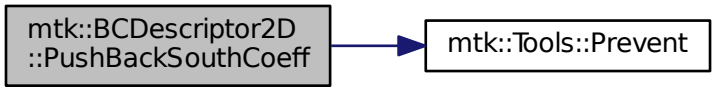
16.2.3.17 void mtk::BCDescriptor2D::PushBackSouthCoeff (mtk::CoefficientFunction2D *cs*)

Parameters

in	*cs*	Function $c_s(x,y) : \Omega \mapsto \mathbb{R}$.

Definition at line 140 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



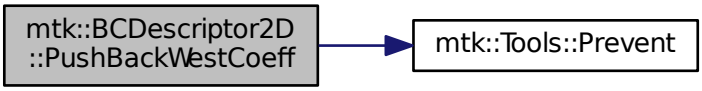
16.2.3.18 void mtk::BCDescriptor2D::PushBackWestCoeff (mtk::CoefficientFunction2D *cw*)

Parameters

in	*cw*	Function $c_w(x,y) : \Omega \mapsto \mathbb{R}$.

Definition at line 114 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



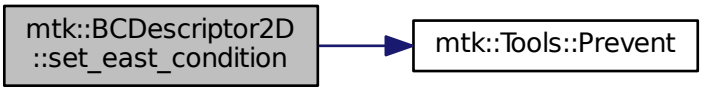
16.2.3.19 void mtk::BCDescriptor2D::set_east_condition (Real(∗)(Real xx, Real yy) *east_condition*) [noexcept]

Parameters

in	*east_condition*	$\beta_e(x,y) : \Omega \mapsto \mathbb{R}$.

Definition at line 176 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



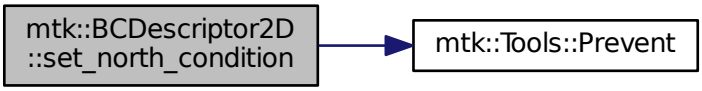
16.2.3.20 void mtk::BCDescriptor2D::set_north_condition (Real(∗)(Real xx, Real yy) *north_condition*) [noexcept]

Parameters

in	*north_condition*	$\beta_n(x,y) : \Omega \mapsto \mathbb{R}$.

Definition at line 197 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



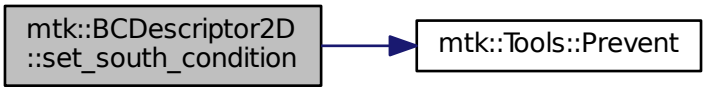
16.2.3.21 void mtk::BCDescriptor2D::set_south_condition (Real(∗)(Real xx, Real yy) *south_condition*) [noexcept]

Parameters

in	*south_condition*	$\beta_s(x,y) : \Omega \mapsto \mathbb{R}$.

Definition at line 186 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



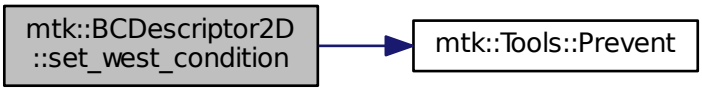
16.2.3.22 void mtk::BCDescriptor2D::set_west_condition (Real(∗)(Real xx, Real yy) *west_condition*) [noexcept]

Parameters

in	*west_condition*	$\beta_w(x,y) : \Omega \mapsto \mathbb{R}$.

Definition at line 166 of file mtk_bc_descriptor_2d.cc.

Here is the call graph for this function:



16.2.4 Member Data Documentation

16.2.4.1 std::vector<CoefficientFunction2D> mtk::BCDescriptor2D::east_coefficients_ [private]

Definition at line 335 of file mtk_bc_descriptor_2d.h.

16.2.4.2 Real(* mtk::BCDescriptor2D::east_condition_)(Real xx, Real yy) [private]

Definition at line 340 of file mtk_bc_descriptor_2d.h.

16.2.4.3 int mtk::BCDescriptor2D::highest_order_diff_east_ [private]

Definition at line 330 of file mtk_bc_descriptor_2d.h.

16.2.4.4 int mtk::BCDescriptor2D::highest_order_diff_north_ [private]

Definition at line 332 of file mtk_bc_descriptor_2d.h.

16.2.4.5 int mtk::BCDescriptor2D::highest_order_diff_south_ [private]

Definition at line 331 of file mtk_bc_descriptor_2d.h.

16.2.4.6 int mtk::BCDescriptor2D::highest_order_diff_west_ [private]

Definition at line 329 of file mtk_bc_descriptor_2d.h.

16.2.4.7 std::vector<CoefficientFunction2D> mtk::BCDescriptor2D::north_coefficients_ [private]

Definition at line 337 of file mtk_bc_descriptor_2d.h.

16.2.4.8 Real(* mtk::BCDescriptor2D::north_condition_)(Real xx, Real yy) [private]

Definition at line 342 of file mtk_bc_descriptor_2d.h.

16.2.4.9 std::vector<CoefficientFunction2D> mtk::BCDescriptor2D::south_coefficients_ [private]

Definition at line 336 of file mtk_bc_descriptor_2d.h.

16.2.4.10 Real(* mtk::BCDescriptor2D::south_condition_)(Real xx, Real yy) [private]

Definition at line 341 of file mtk_bc_descriptor_2d.h.

16.2.4.11 std::vector<CoefficientFunction2D> mtk::BCDescriptor2D::west_coefficients_ [private]

Definition at line 334 of file mtk_bc_descriptor_2d.h.

16.2.4.12 Real(* mtk::BCDescriptor2D::west_condition_)(Real xx, Real yy) [private]

Definition at line 339 of file mtk_bc_descriptor_2d.h.

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

- include/mtk_bc_descriptor_2d.h
- src/mtk_bc_descriptor_2d.cc

16.3 mtk::BLASAdapter Class Reference

Adapter class for the BLAS API.

`#include <mtk_blas_adapter.h>`

Collaboration diagram for mtk::BLASAdapter:

Static Public Member Functions

- static Real RealNRM2 (Real *in, int &in_length)

  Compute the $||\mathbf{x}||_2$ *of given array* $\mathbf{x}$.

- static void RealAXPY (Real alpha, Real ∗xx, Real ∗yy, int &in_length)

  Real-Arithmetic Scalar-Vector plus a Vector.

- static Real RelNorm2Error (Real ∗computed, Real ∗known, int length)

  Computes the relative norm-2 of the error.

- static void RealDenseMV (Real &alpha, DenseMatrix &aa, Real ∗xx, Real &beta, Real ∗yy)

  Real-Arithmetic General (Dense matrices) Matrix-Vector Multiplier.

- static DenseMatrix RealDenseMM (DenseMatrix &aa, DenseMatrix &bb)

  Real-Arithmetic General (Dense matrices) Matrix-Matrix multiplier.

16.3.1 Detailed Description

This class contains a collection of static classes, that posses direct access to the underlying structure of the matrices, thus allowing programmers to exploit some of the numerical methods implemented in the BLAS.

The **BLAS (Basic Linear Algebra Subprograms)** are routines that provide standard building blocks for performing basic vector and matrix operations. The Level 1 BLAS perform scalar, vector and vector-vector operations, the Level 2 BLAS perform matrix-vector operations, and the Level 3 BLAS perform matrix-matrix operations.

See also

http://www.netlib.org/blas/

Definition at line 96 of file mtk_blas_adapter.h.

16.3.2 Member Function Documentation

16.3.2.1 void mtk::BLASAdapter::RealAXPY (mtk::Real *alpha*, mtk::Real ∗ *xx*, mtk::Real ∗ *yy*, int & *in_length*) [static]

Performs

$$\mathbf{y} := \alpha \mathbf{A} mathbf x + \mathbf{y}$$

Parameters

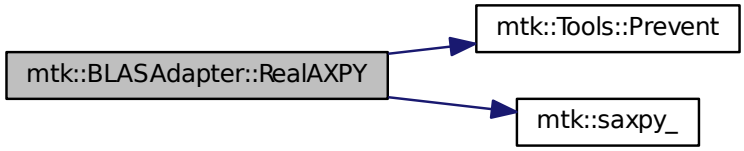
in	*alpha*	Scalar of the first array.
in	*xx*	First array.
in	*yy*	Second array.
in	*in_length*	Lengths of the given arrays.

Returns

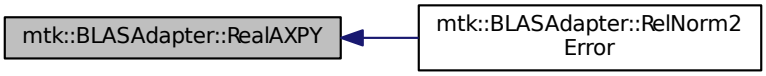
Norm-2 of the given array.

Definition at line 339 of file mtk_blas_adapter.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



16.3.2.2 mtk::DenseMatrix mtk::BLASAdapter::RealDenseMM (mtk::DenseMatrix & *aa*, mtk::DenseMatrix & *bb*) [static]

Performs:

$$\mathbf{C} := \mathbf{AB}$$

Parameters

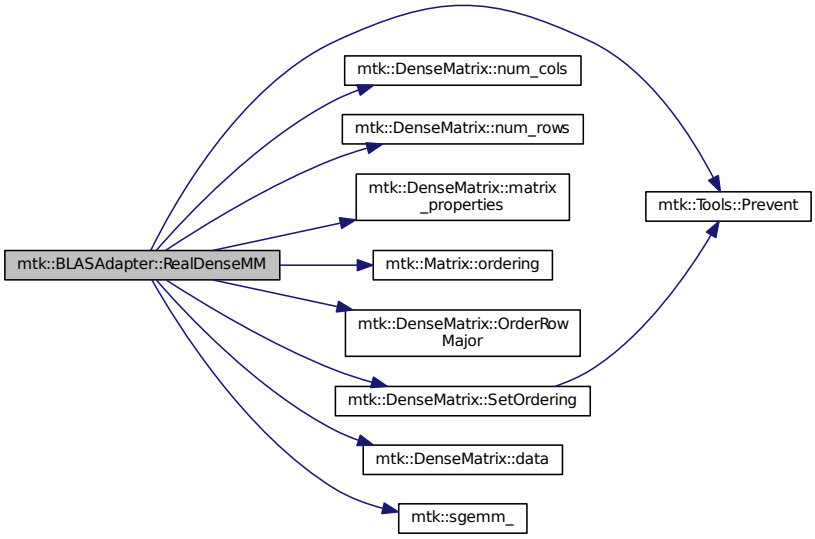
in	*aa*	First matrix.
in	*bb*	Second matrix.

See also

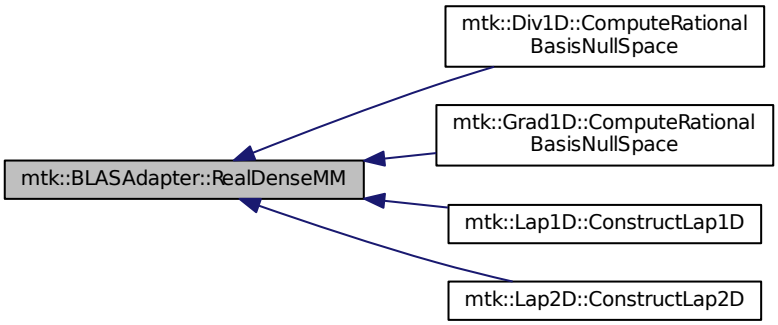
http://ejspeiro.github.io/Netlib-and-CPP/

Definition at line 409 of file mtk_blas_adapter.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



16.3.2.3 void mtk::BLASAdapter::RealDenseMV (mtk::Real & *alpha*, mtk::DenseMatrix & *aa*, mtk::Real * *xx*, mtk::Real & *beta*, mtk::Real * *yy*) [static]

Performs

$$\mathbf{y} := \alpha\mathbf{A}\mathbf{x} + \beta\mathbf{y}$$

Parameters

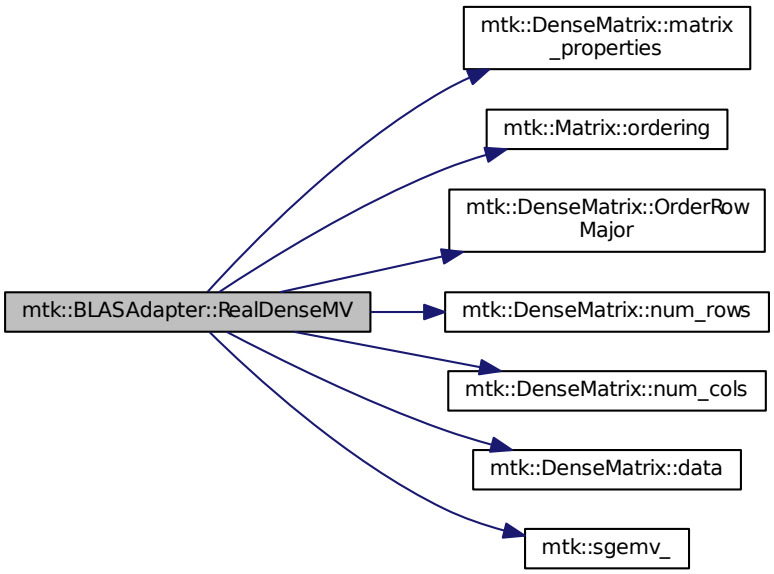
in	*alpha*	First scalar.
in	*aa*	Given matrix.
in	*xx*	First vector.
in	*beta*	Second scalar.
in,out	*yy*	Second vector (output).

See also

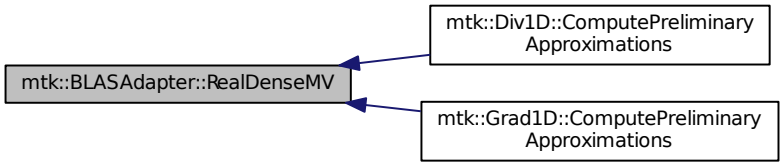
http://ejspeiro.github.io/Netlib-and-CPP/

Definition at line 378 of file mtk_blas_adapter.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



16.3.2.4 mtk::Real mtk::BLASAdapter::RealNRM2 (Real * *in*, int & *in_length*) [static]

Parameters

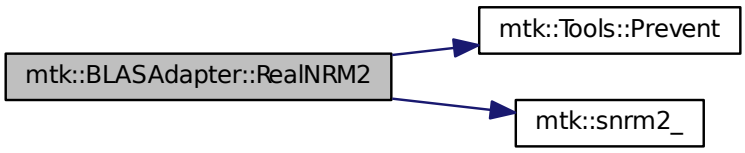
in	*in*	Input array.
in	*in_length*	Length of the array.

Returns

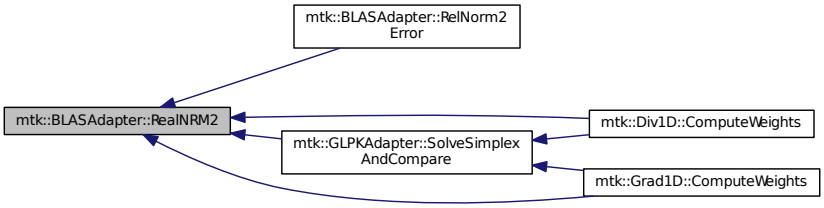
Norm-2 of the given array.

Definition at line 324 of file mtk_blas_adapter.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



16.3.2.5 mtk::Real mtk::BLASAdapter::RelNorm2Error (mtk::Real * *computed,* mtk::Real * *known,* int *length*) [static]

We compute

$$\frac{||\tilde{\mathbf{x}} - \mathbf{x}||_2}{||\mathbf{x}||_2}.$$

Parameters

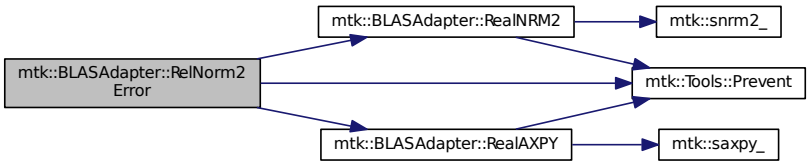
in	*known*	Array containing the computed solution.
in	*computed*	Array containing the known solution (ref. solution).

Returns

Relative norm-2 of the error, aka, the difference between the arrays.

Definition at line 358 of file mtk_blas_adapter.cc.

Here is the call graph for this function:



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

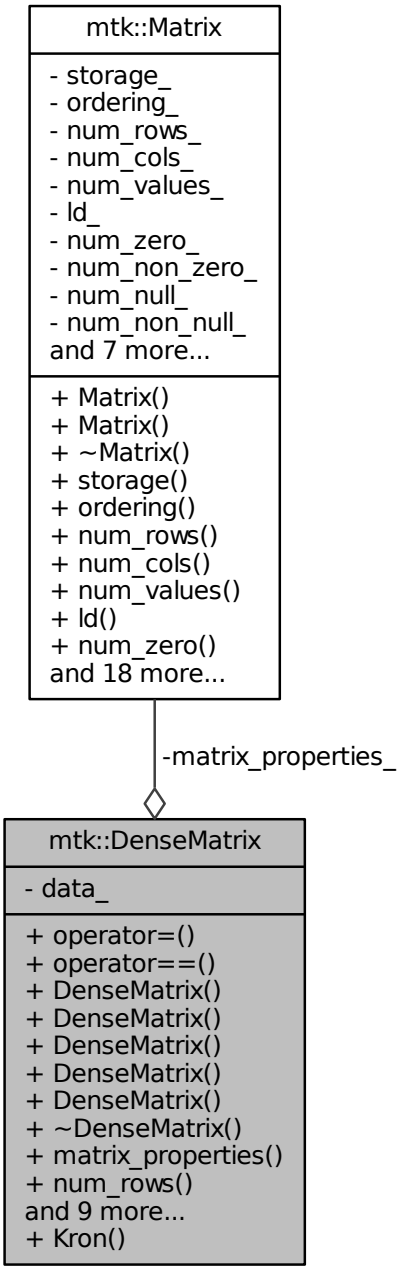
- include/mtk_blas_adapter.h
- src/mtk_blas_adapter.cc

16.4 mtk::DenseMatrix Class Reference

Defines a common dense matrix, using a 1D array.

`#include <mtk_dense_matrix.h>`

Collaboration diagram for mtk::DenseMatrix:



Public Member Functions

- DenseMatrix & operator= (const DenseMatrix &in)

Overloaded assignment operator.

- bool operator== (const DenseMatrix &in)

  Am I equal to the in matrix?

- DenseMatrix ()

  Default constructor.

- DenseMatrix (const DenseMatrix &in)

  Copy constructor.

- DenseMatrix (const int &num_rows, const int &num_cols)

  Construct a dense matrix based on the given dimensions.

- DenseMatrix (const int &rank, const bool &padded, const bool &transpose)

  Construct a zero-rows-padded identity matrix.

- DenseMatrix (const Real *const gen, const int &gen_length, const int &pro_length, const bool &transpose)

  Construct a dense Vandermonde matrix.

- ~DenseMatrix ()

  Destructor.

- Matrix matrix_properties () const noexcept

  Provides access to the matrix data.

- int num_rows () const noexcept

  Gets the number of rows.

- int num_cols () const noexcept

  Gets the number of columns.

- Real * data () const noexcept

  Provides access to the matrix value array.

- void SetOrdering (mtk::MatrixOrdering oo) noexcept

  Sets the ordering of the matrix.

- Real GetValue (const int &row_coord, const int &col_coord) const noexcept

  Gets a value on the given coordinates.

- void SetValue (const int &row_coord, const int &col_coord, const Real &val) noexcept

  Sets a value on the given coordinates.

- void Transpose ()

  Transpose this matrix.

- void OrderRowMajor ()

  Make the matrix row-wise ordered.

- void OrderColMajor ()

  Make the matrix column-wise ordered.

- bool WriteToFile (const std::string &filename) const

  Writes matrix to a file compatible with Gnuplot 4.6.

Static Public Member Functions

- static DenseMatrix Kron (const DenseMatrix &aa, const DenseMatrix &bb)

  Construct a dense matrix based on the Kronecker product of arguments.

Private Attributes

- Matrix matrix_properties_

  Data related to the matrix nature.

- Real * data_

  Array holding the data in contiguous position in memory.

Friends

- std::ostream & operator<< (std::ostream &stream, DenseMatrix &in)

  Prints the matrix as a block of numbers (standard way).

16.4.1 Detailed Description

For developing purposes, it is better to have a not-so-intrincated data structure implementing matrices. This is the purpose of this class: to be used for prototypes of new code for small test cases. In every other instance, this should be replaced by the most appropriate sparse matrix.

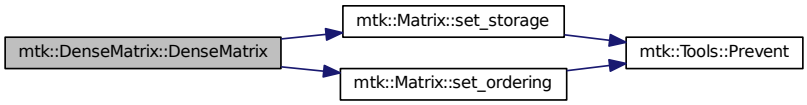
Definition at line 92 of file mtk_dense_matrix.h.

16.4.2 Constructor & Destructor Documentation

16.4.2.1 mtk::DenseMatrix::DenseMatrix ( )

Definition at line 162 of file mtk_dense_matrix.cc.

Here is the call graph for this function:



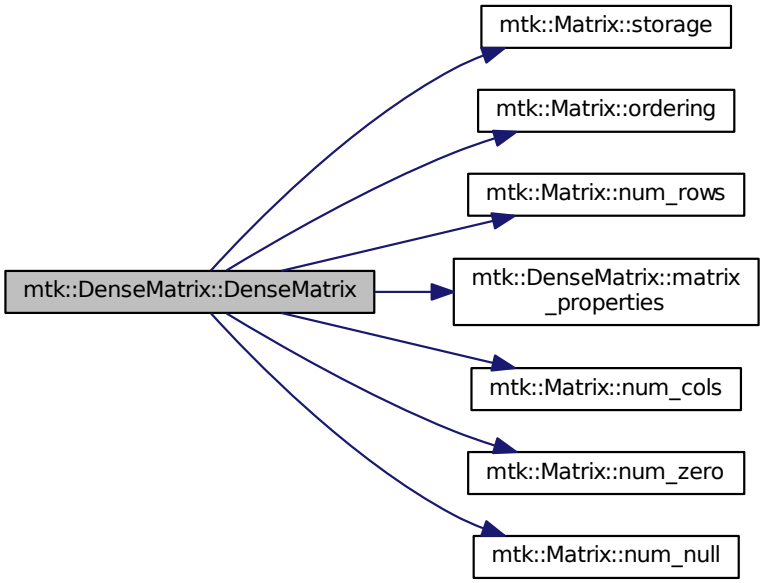
16.4.2.2 mtk::DenseMatrix::DenseMatrix (const DenseMatrix & *in*)

Parameters

in	*in*	Given matrix.

Definition at line 168 of file mtk_dense_matrix.cc.

Here is the call graph for this function:



16.4.2.3 mtk::DenseMatrix::DenseMatrix (const int & *num_rows*, const int & *num_cols*)

Parameters

in	*num_rows*	Number of rows of the required matrix.
in	*num_cols*	Number of rows of the required matrix.

Exceptions

std::bad_alloc	

Definition at line 201 of file mtk_dense_matrix.cc.

Here is the call graph for this function:

16.4.2.4 mtk::DenseMatrix::DenseMatrix (const int & *rank*, const bool & *padded*, const bool & *transpose*)

Used in the construction of the mimetic operators.

Def$**$. A **padded matrix** is a matrix with its first and last rows initialized to only zero values:

$$\bar{\mathbf{I}} = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}$$

Parameters

in	*rank*	Rank or number of rows/cols in square matrix.
in	*padded*	Should it be padded?
in	*transpose*	Should I return the transpose of the requested matrix?

Exceptions

std::bad_alloc	

Definition at line 223 of file mtk_dense_matrix.cc.

Here is the call graph for this function:

16.4.2.5 mtk::DenseMatrix::DenseMatrix (const Real *const *gen*, const int & *gen_length*, const int & *pro_length*, const bool & *transpose*)

Def$**$. In linear algebra, a **Vandermonde matrix** is a matrix with terms of a geometric progression in each row. This progression uses the terms of a given **generator vector**:

$$\mathbf{V} = \begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^{n-1} \\ 1 & \alpha_3 & \alpha_3^2 & \dots & \alpha_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_m & \alpha_m^2 & \dots & \alpha_m^{n-1} \end{pmatrix}$$

This constructor generates a Vandermonde matrix, as defined above.

Obs$**$. It in important to understand that the generator vectors to be used are nothing but a very particular instance of a grid. These are little chunks, little samples, if you will, of a grid which is rectangular and uniform. So the selected samples, on the mtk::Div1D and mtk::Grad1D, basically represent the entire space, the entire grid. This is why nor the CRS nor the CBS algorithms may work for irregular geometries, such as curvilinear grids.

Parameters

in	*gen*	Given generator vector.
in	*gen_length*	Length generator vector.
in	*pro_length*	Length the progression.
in	*transpose*	Should the transpose be created instead?

Exceptions

std::bad_alloc	

Definition at line 264 of file mtk_dense_matrix.cc.

Here is the call graph for this function:

16.4.2.6 mtk::DenseMatrix::~DenseMatrix ( )

Definition at line 312 of file mtk_dense_matrix.cc.

16.4.3 Member Function Documentation

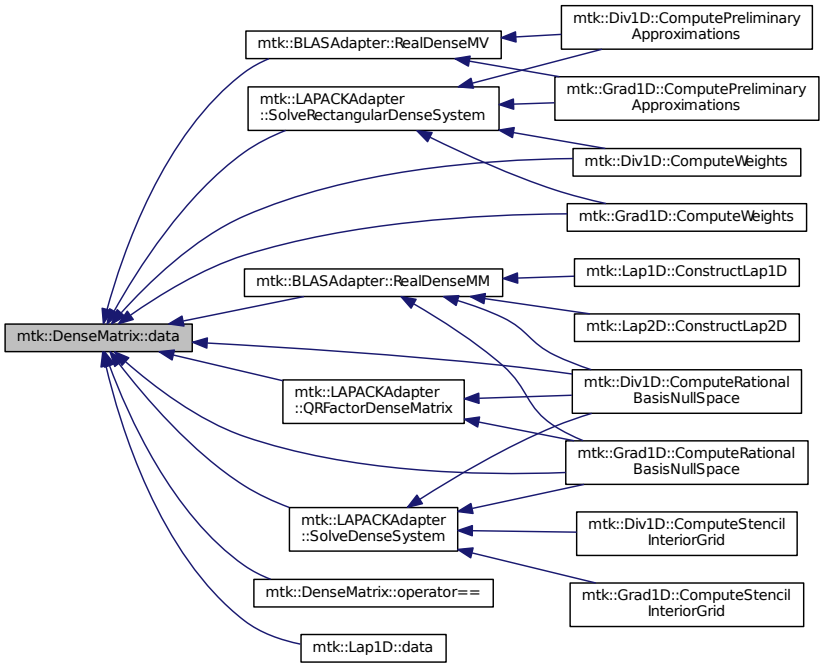
16.4.3.1 mtk::Real * mtk::DenseMatrix::data ( ) const [noexcept]

Returns

Pointer to an array of mtk::Real.

Definition at line 343 of file mtk_dense_matrix.cc.

Here is the caller graph for this function:



16.4.3.2 mtk::Real mtk::DenseMatrix::GetValue (const int & *row_coord*, const int & *col_coord*) const [noexcept]

Parameters

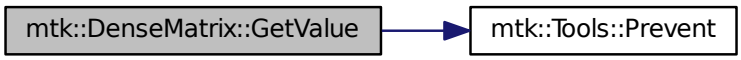
in	*row_coord*	Row coordinate.
in	*col_coord*	Column coordinate.

Returns

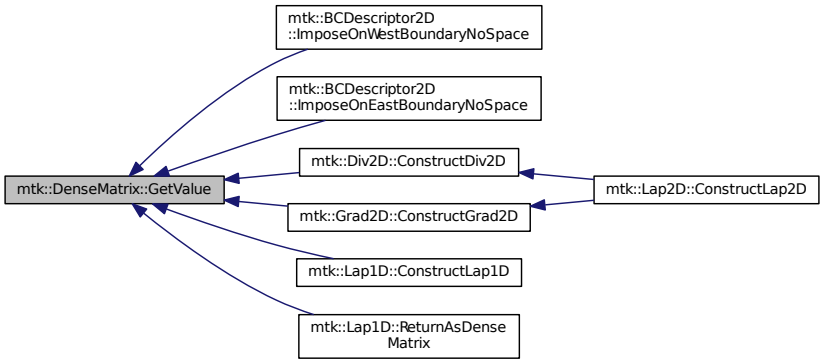
The required value at the specified coordinates.

Definition at line 348 of file mtk_dense_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



16.4.3.3 mtk::DenseMatrix mtk::DenseMatrix::Kron (const DenseMatrix & *aa*, const DenseMatrix & *bb*) [static]

Parameters

in	*aa*	First matrix.
in	*bb*	Second matrix.

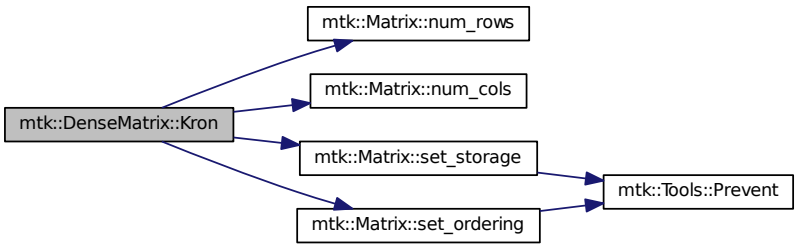
Exceptions

std::bad_alloc	

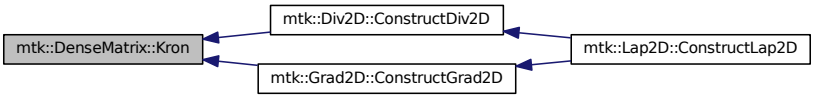
Todo Implement Kronecker product using the BLAS.

Definition at line 490 of file mtk_dense_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



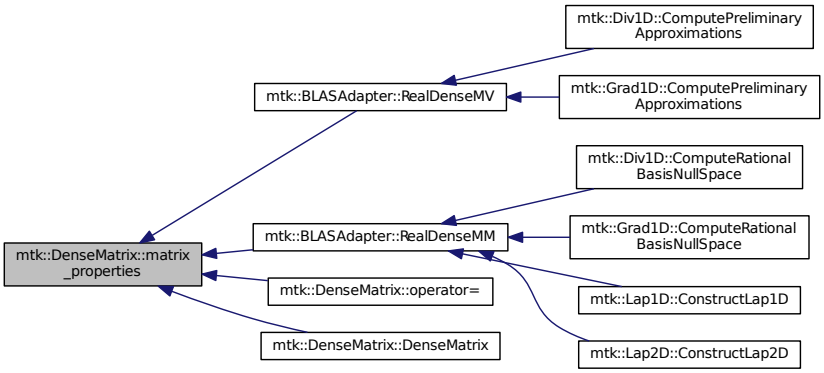
16.4.3.4 mtk::Matrix mtk::DenseMatrix::matrix_properties ( ) const [noexcept]

Returns

Pointer to a Matrix.

Definition at line 318 of file mtk_dense_matrix.cc.

Here is the caller graph for this function:



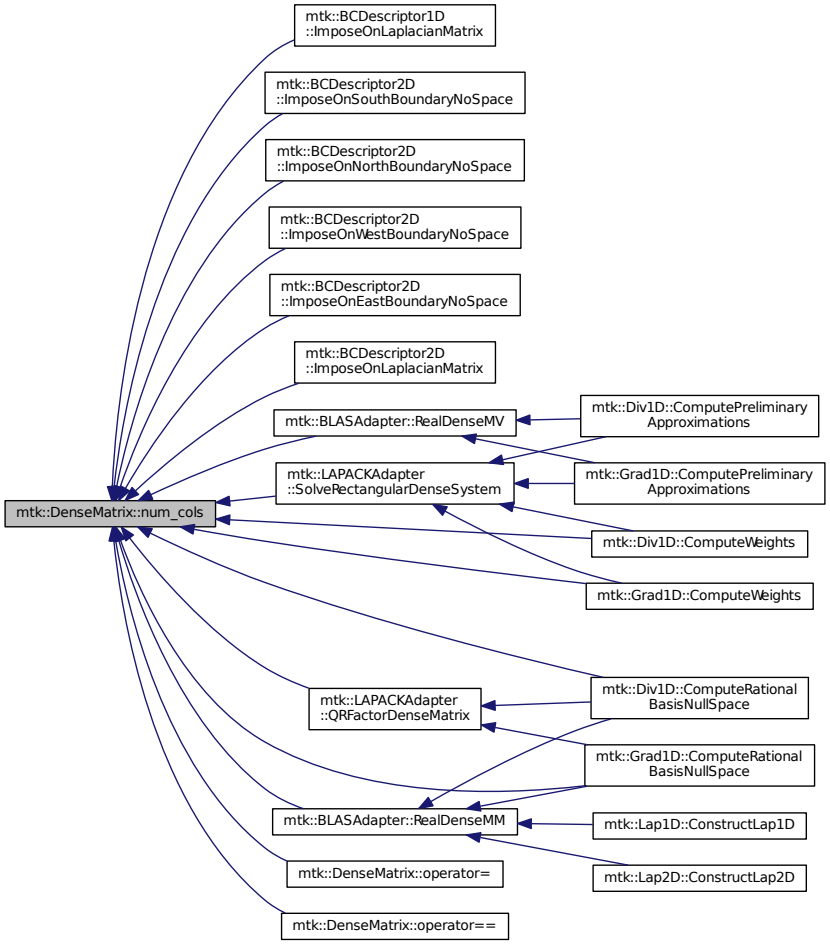
16.4.3.5 int mtk::DenseMatrix::num_cols ( ) const [noexcept]

Returns

Number of columns of the matrix.

Definition at line 338 of file mtk_dense_matrix.cc.

Here is the caller graph for this function:



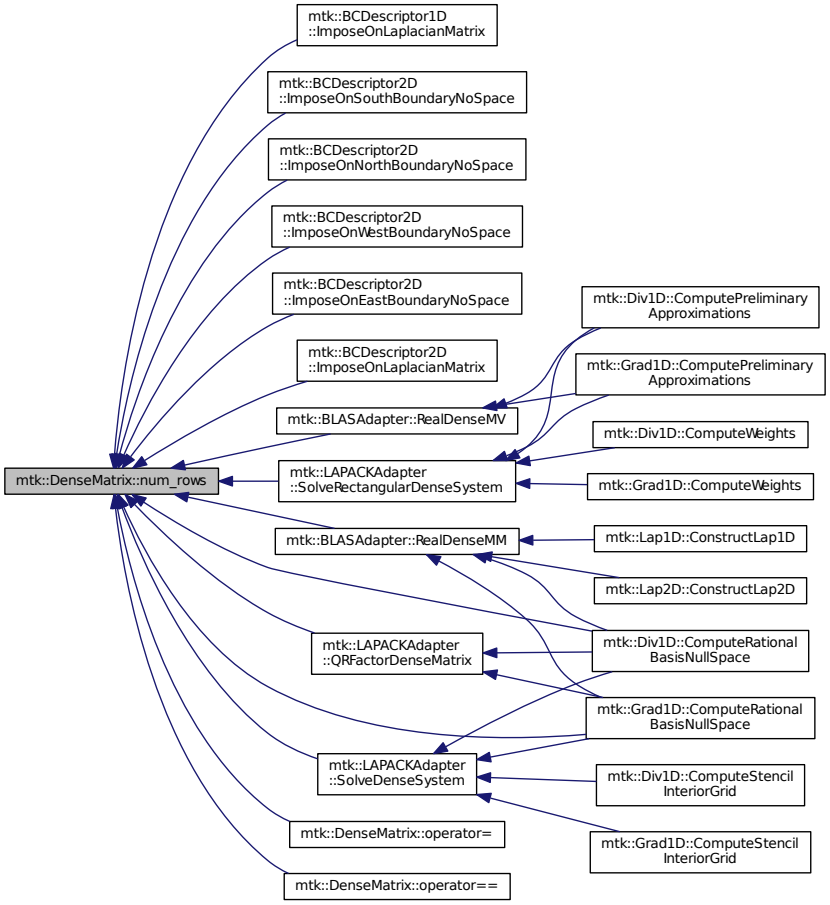
16.4.3.6 int mtk::DenseMatrix::num_rows ( ) const [noexcept]

Returns

Number of rows of the matrix.

Definition at line 333 of file mtk_dense_matrix.cc.

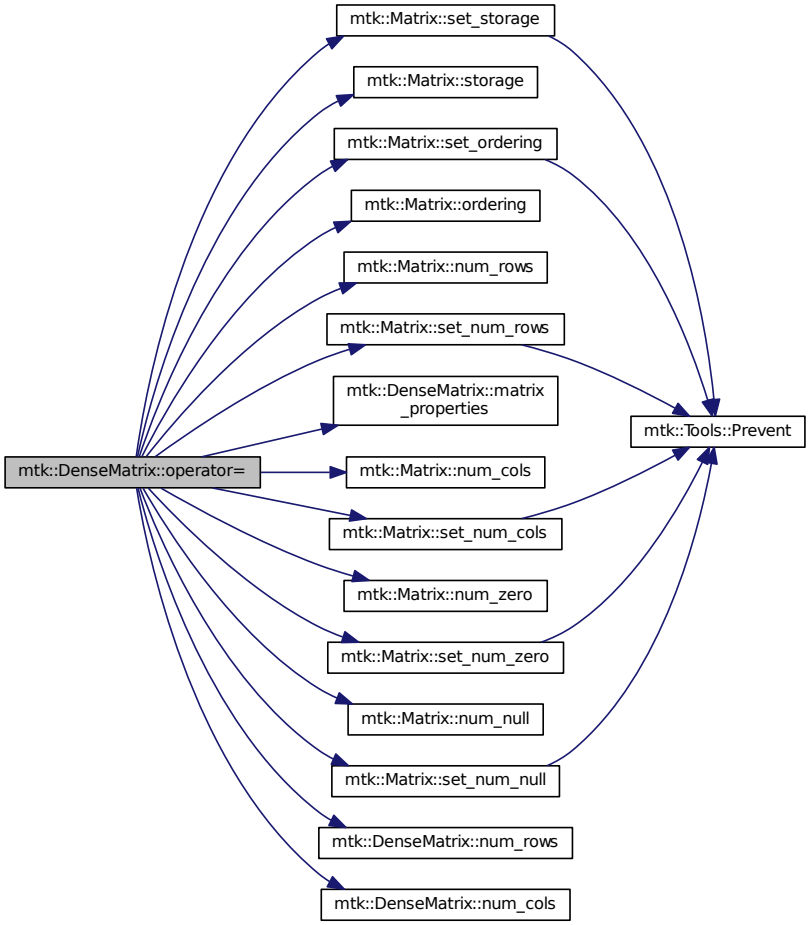
Here is the caller graph for this function:



16.4.3.7 mtk::DenseMatrix & mtk::DenseMatrix::operator= (const DenseMatrix & *in*)

Definition at line 100 of file mtk_dense_matrix.cc.

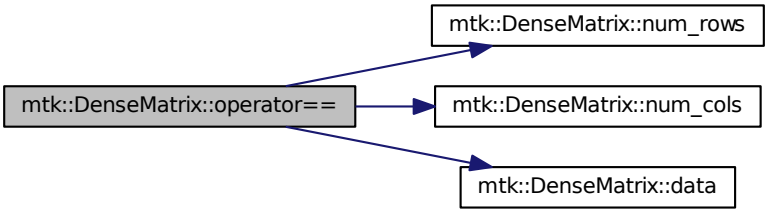
Here is the call graph for this function:



16.4.3.8 bool mtk::DenseMatrix::operator== (const DenseMatrix & *in*)

Definition at line 141 of file mtk_dense_matrix.cc.

Here is the call graph for this function:

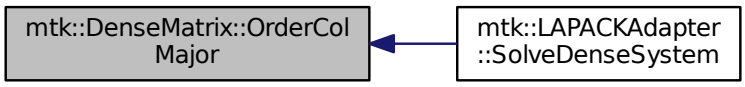


16.4.3.9 void mtk::DenseMatrix::OrderColMajor ( )

Todo Improve this so that no new arrays have to be created.

Definition at line 451 of file mtk_dense_matrix.cc.

Here is the caller graph for this function:

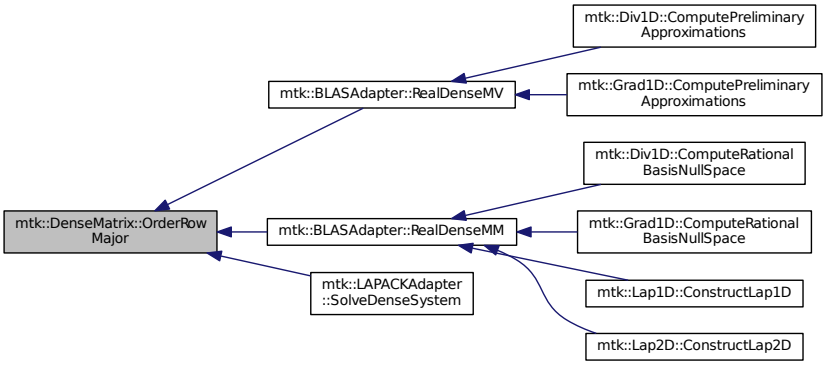


16.4.3.10 void mtk::DenseMatrix::OrderRowMajor ( )

Todo Improve this so that no new arrays have to be created.

Definition at line 410 of file mtk_dense_matrix.cc.

Here is the caller graph for this function:



16.4.3.11 void mtk::DenseMatrix::SetOrdering (mtk::MatrixOrdering *oo*) [noexcept]

Parameters

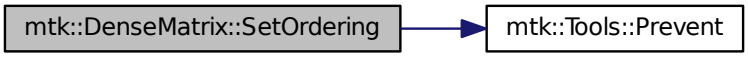
in	*oo*	Ordering.

Returns

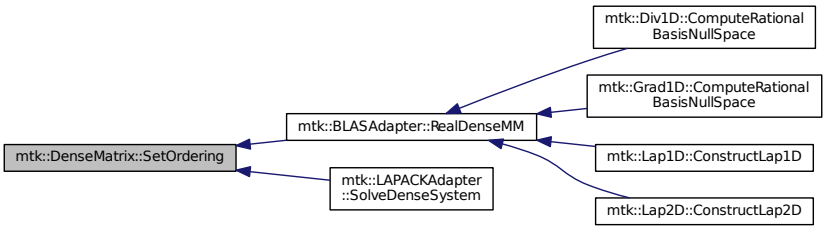
The required value at the specified coordinates.

Definition at line 323 of file mtk_dense_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



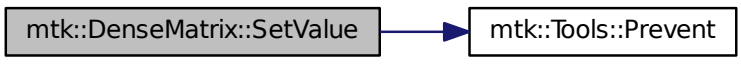
16.4.3.12 void mtk::DenseMatrix::SetValue (const int & *row_coord,* const int & *col_coord,* const **Real** & *val*) [noexcept]

Parameters

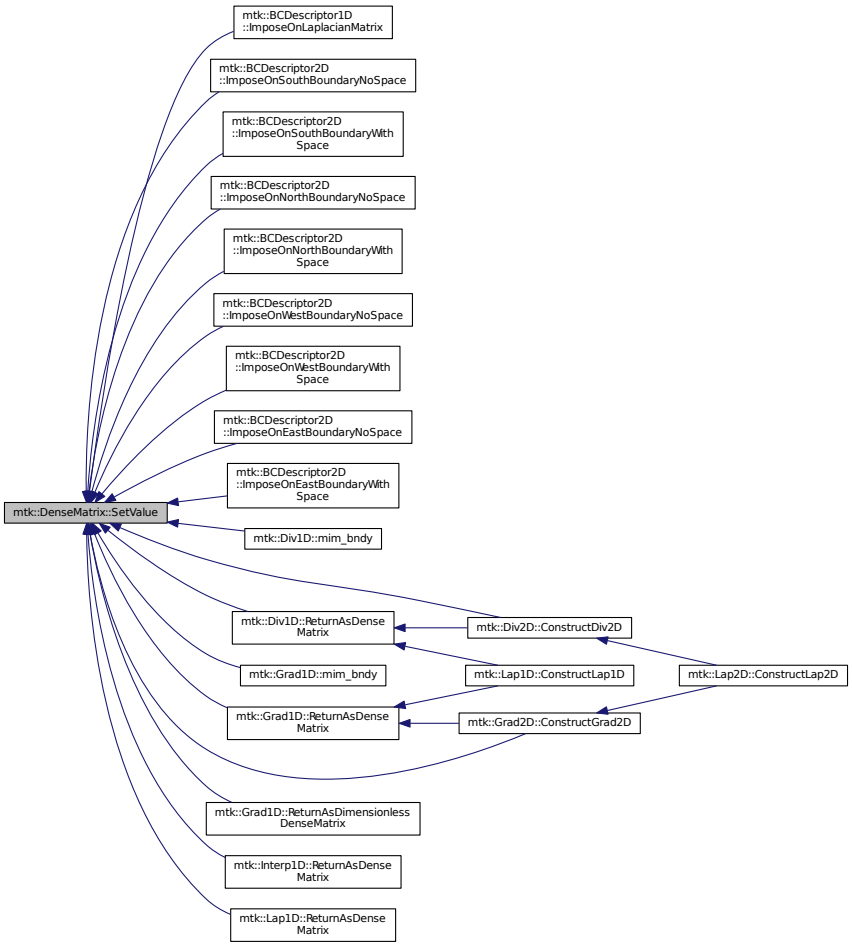
in	*row_coord*	Row coordinate.
in	*col_coord*	Column coordinate.
in	*val*	Row Actual value to be inserted.

Definition at line 360 of file mtk_dense_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:

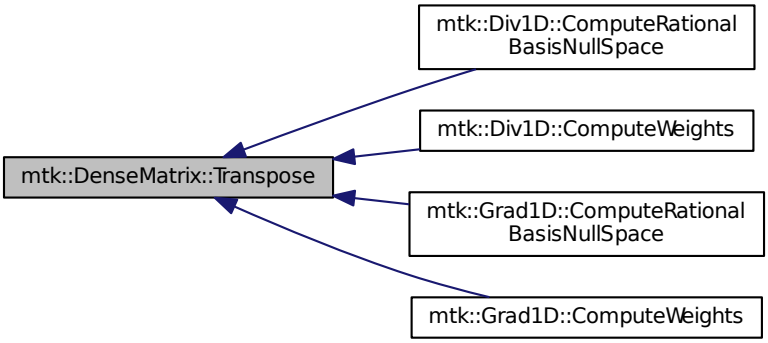


16.4.3.13 void mtk::DenseMatrix::Transpose ( )

Todo Improve this so that no extra arrays have to be created.

Definition at line 373 of file mtk_dense_matrix.cc.

Here is the caller graph for this function:



16.4.3.14 bool mtk::DenseMatrix::WriteToFile (const std::string & *filename*) const

Parameters

in	*filename*	Name of the output file.

Returns

Success of the file writing process.

See also

http://www.gnuplot.info/

Definition at line 531 of file mtk_dense_matrix.cc.

16.4.4 Friends And Related Function Documentation

16.4.4.1 std::ostream& operator<< (std::ostream & *stream,* mtk::DenseMatrix & *in*) [friend]

Definition at line 77 of file mtk_dense_matrix.cc.

16.4.5 Member Data Documentation

16.4.5.1 Real* mtk::DenseMatrix::data_ [private]

Definition at line 285 of file mtk_dense_matrix.h.

16.4.5.2 Matrix mtk::DenseMatrix::matrix_properties_ [private]

Definition at line 283 of file mtk_dense_matrix.h.

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

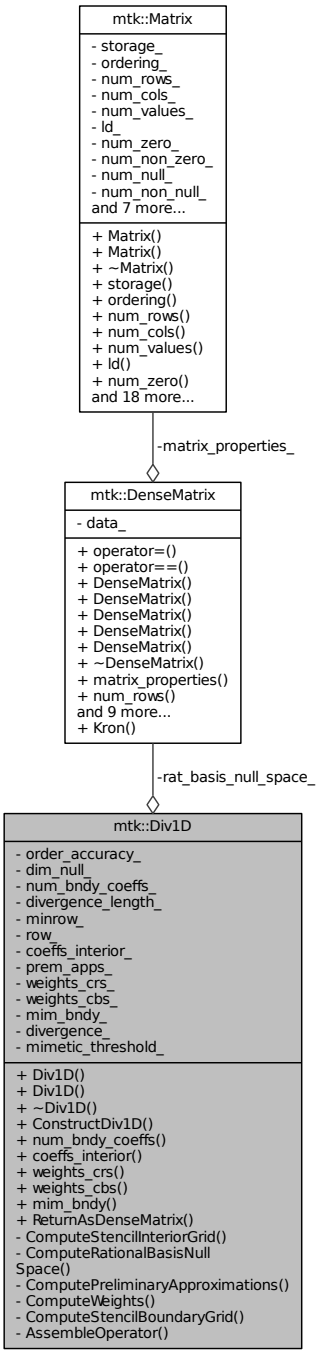
- include/mtk_dense_matrix.h
- src/mtk_dense_matrix.cc

16.5 mtk::Div1D Class Reference

Implements a 1D mimetic divergence operator.

`#include <mtk_div_1d.h>`

Collaboration diagram for mtk::Div1D:



Public Member Functions

- Div1D ()

Default constructor.

- Div1D (const Div1D &div)

  Copy constructor.

- ~Div1D ()

  Destructor.

- bool ConstructDiv1D (int order_accuracy=kDefaultOrderAccuracy, Real mimetic_threshold=kDefaultMimeticThreshold)

  Factory method implementing the CBS Algorithm to build operator.

- int num_bndy_coeffs () const

  Returns how many coefficients are approximating at the boundary.

- Real * coeffs_interior () const

  Returns coefficients for the interior of the grid.

- Real * weights_crs (void) const

  Return collection of weights as computed by the CRSA.

- Real * weights_cbs (void) const

  Return collection of weights as computed by the CBSA.

- DenseMatrix mim_bndy () const

  Return collection of mimetic approximations at the boundary.

- DenseMatrix ReturnAsDenseMatrix (const UniStgGrid1D &grid) const

  Return the operator as a dense matrix.

Private Member Functions

- bool ComputeStencilInteriorGrid (void)

  Stage 1 of the CBS Algorithm.

- bool ComputeRationalBasisNullSpace (void)

  Stage 2.1 of the CBS Algorithm.

- bool ComputePreliminaryApproximations (void)

  Stage 2.2 of the CBS Algorithm.

- bool ComputeWeights (void)

  Stage 2.3 of the CBS Algorithm.

- bool ComputeStencilBoundaryGrid (void)

  Stage 2.4 of the CBS Algorithm.

- bool AssembleOperator (void)

  Stage 3 of the CBS Algorithm.

Private Attributes

- int order_accuracy_

  Order of numerical accuracy of the operator.

- int dim_null_

  Dim. null-space for boundary approximations.

- int num_bndy_coeffs_

  Req. coeffs. per bndy pt. uni. order accuracy.

- int divergence_length_

  Length of the output array.

- int minrow_

Row from the optimizer with the minimum rel. nor.

- int row_

  Row currently processed by the optimizer.
- DenseMatrix rat_basis_null_space_

  Rational b. null-space w. bndy.
- Real * coeffs_interior_

  Interior stencil.
- Real * prem_apps_

  2D array of boundary preliminary approximations.
- Real * weights_crs_

  Array containing weights from CRSA.
- Real * weights_cbs_

  Array containing weights from CBSA.
- Real * mim_bndy_

  Array containing mimetic boundary approximations.
- Real * divergence_

  Output array containing the operator and weights.
- Real mimetic_threshold_

  < Mimetic threshold.

Friends

- std::ostream & operator<< (std::ostream &stream, Div1D &in)

  Output stream operator for printing.

16.5.1 Detailed Description

This class implements a 1D divergence operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm (CBSA).

Definition at line 81 of file mtk_div_1d.h.

16.5.2 Constructor & Destructor Documentation

16.5.2.1 mtk::Div1D::Div1D ( )

Definition at line 125 of file mtk_div_1d.cc.

16.5.2.2 mtk::Div1D::Div1D (const Div1D & *div*)

Parameters

in	*div*	Given divergence.

Definition at line 140 of file mtk_div_1d.cc.

16.5.2.3 mtk::Div1D::~Div1D ( )

Definition at line 155 of file mtk_div_1d.cc.

16.5.3 Member Function Documentation

16.5.3.1 bool mtk::Div1D::AssembleOperator (void) [private]

Construct the output array with the operator and its weights.

1. The first entry of the array will contain the order of accuracy.
2. The second entry the collection of coefficients for interior of grid.
3. If order_accuracy_ $>$ 2, then third entry is the collection of weights.
4. If order_accuracy_ $>$ 2, next dim_null_ entries is approximating coefficients for the west boundary of the grid.

Definition at line 1334 of file mtk_div_1d.cc.

16.5.3.2 mtk::Real * mtk::Div1D::coeffs_interior ( ) const

Returns

Coefficients for the interior of the grid.

Definition at line 320 of file mtk_div_1d.cc.

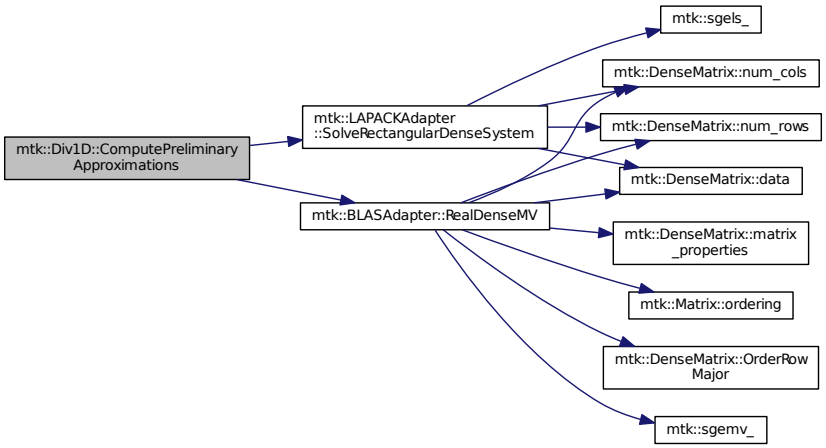
16.5.3.3 bool mtk::Div1D::ComputePreliminaryApproximations (void) [private]

Compute the set of preliminary approximations on the boundary neighborhood.

1. Create generator vector for the first approximation.
2. Compute the dim_null near-the-boundary columns of the pi matrix.
3. Create the Vandermonde matrix for this iteration.
4. New order-selector vector (gets re-written with LAPACK solutions).
5. Solving TT*rr = ob yields the columns rr of the KK matrix.
6. Scale the KK matrix to make it a rational basis for null-space.
7. Extract the last dim_null values of the pre-scaled ob.
8. Once we posses the bottom elements, we proceed with the scaling.

Definition at line 689 of file mtk_div_1d.cc.

Here is the call graph for this function:



16.5.3.4 bool mtk::Div1D::ComputeRationalBasisNullSpace (void) [private]

Compute a rational basis for the null-space of the Vandermonde matrix approximating at the west boundary.

1. Create generator vector for the first approximation.

2. Create Vandermonde matrix.

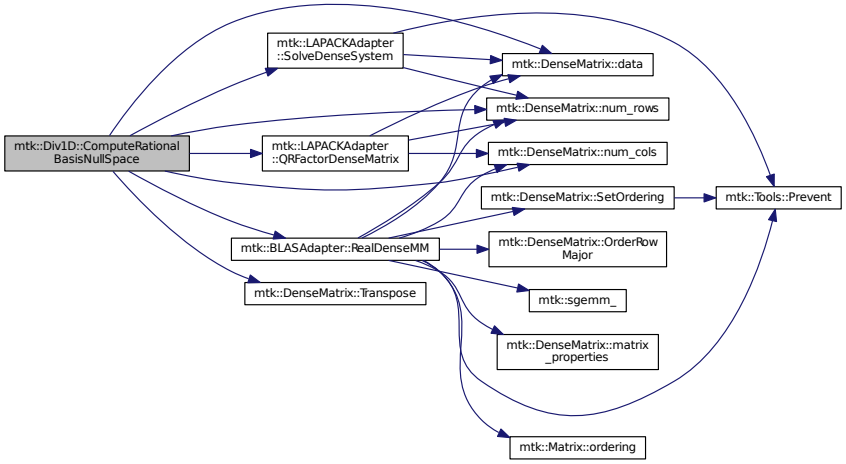
3. QR-factorize the Vandermonde matrix.

4. Extract the basis for the null-space from Q matrix.

5. Scale null-space to make it rational.

Definition at line 513 of file mtk_div_1d.cc.

Here is the call graph for this function:



16.5.3.5 bool mtk::Div1D::ComputeStencilBoundaryGrid (void) [private]

Compute mimetic stencil approximating at boundary.

1. Collect lambda values.
2. Compute alpha values.
3. Compute the mimetic boundary approximations.

Definition at line 1235 of file mtk_div_1d.cc.

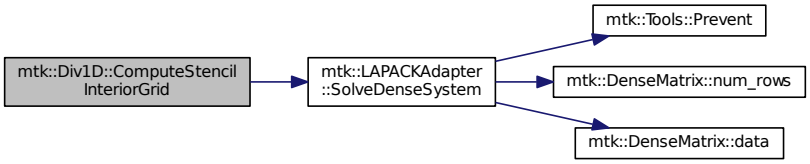
16.5.3.6 bool mtk::Div1D::ComputeStencilInteriorGrid (void) [private]

Compute the stencil approximating the interior of the staggered grid.

1. Create vector for interior spatial coordinates.
2. Create Vandermonde matrix (using interior coordinates as generator).
3. Create order-selector vector.
4. Solve dense Vandermonde system to attain the interior coefficients.

Definition at line 414 of file mtk_div_1d.cc.

Here is the call graph for this function:



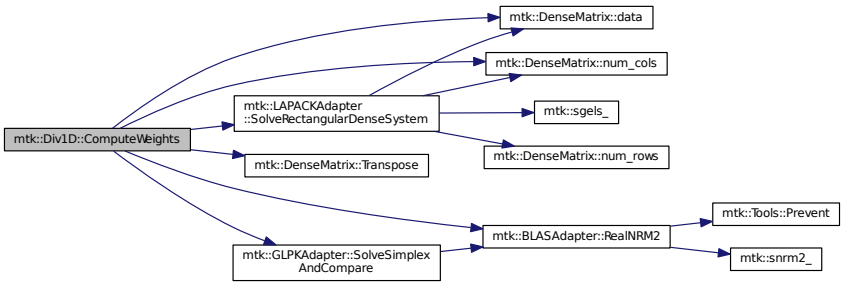
16.5.3.7 bool mtk::Div1D::ComputeWeights (void) [private]

Compute the set of mimetic weights to impose the mimetic condition.

1. Construct the ■ matrix.
2. Use interior stencil to build proper RHS vector $\mathbf{h}$.
3. Get weights (as **CRSA**): $\blacksquare\mathbf{q} = \mathbf{h}$.
4. If required order is greater than critical order, start the **CBSA**.
5. Create ■ matrix from ■.
6. Prepare constraint vector as in the CBSA: ■.
7. Brute force search through all the rows of the Φ matrix.
8. Apply solution found from brute force search.

Definition at line 909 of file mtk_div_1d.cc.

Here is the call graph for this function:



16.5.3.8 bool mtk::Div1D::ConstructDiv1D (int *order_accuracy* = kDefaultOrderAccuracy, mtk::Real *mimetic_threshold* = kDefaultMimeticThreshold)

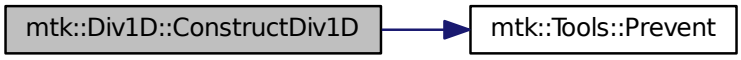
Returns

Success of the construction.

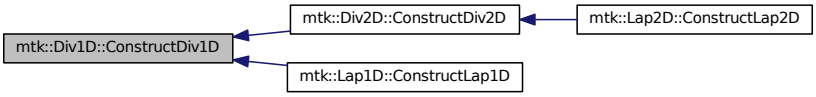
1. Compute stencil for the interior cells.
2. Compute a rational basis for the null-space for the first matrix.
3. Compute preliminary approximation (non-mimetic) on the boundaries.
4. Compute quadrature weights to impose the mimetic conditions.
5. Compute real approximation (mimetic) on the boundaries.
6. Assemble operator.

Definition at line 176 of file mtk_div_1d.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



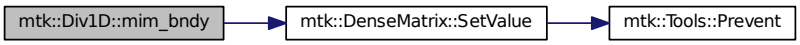
16.5.3.9 mtk::DenseMatrix mtk::Div1D::mim_bndy ( ) const

Returns

Collection of mimetic approximations at the boundary.

Definition at line 336 of file mtk_div_1d.cc.

Here is the call graph for this function:



16.5.3.10 int mtk::Div1D::num_bndy_coeffs ( ) const

Returns

How many coefficients are approximating at the boundary.

Definition at line 315 of file mtk_div_1d.cc.

16.5.3.11 mtk::DenseMatrix mtk::Div1D::ReturnAsDenseMatrix (const UniStgGrid1D & *grid*) const

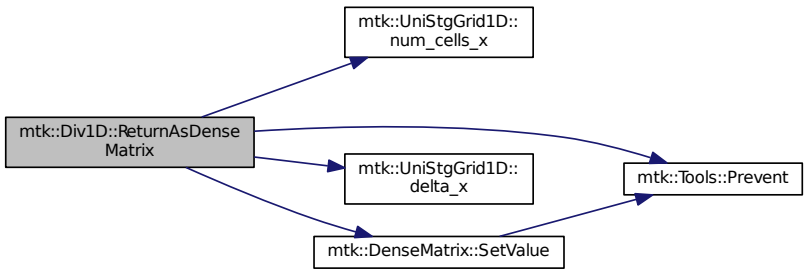
Returns

The operator as a dense matrix.

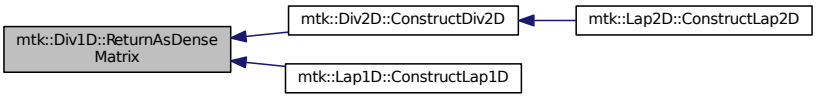
1. Insert mimetic boundary at the west.
2. Insert coefficients for the interior of the grid.
3. Impose center-skew symmetry by permuting the mimetic boundaries.

Definition at line 351 of file mtk_div_1d.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



16.5.3.12 mtk::Real * mtk::Div1D::weights_cbs (void) const

Returns

Collection of weights as computed by the CBSA.

Definition at line 330 of file mtk_div_1d.cc.

16.5.3.13 mtk::Real * mtk::Div1D::weights_crs (void) const

Returns

Collection of weights as computed by the CRSA.

Definition at line 325 of file mtk_div_1d.cc.

16.5.4 Friends And Related Function Documentation

16.5.4.1 std::ostream& operator<< (std::ostream & *stream,* mtk::Div1D & *in*) [friend]

1. Print order of accuracy.
2. Print approximating coefficients for the interior.
3. Print mimetic weights.
4. Print mimetic approximations at the boundary.

Definition at line 79 of file mtk_div_1d.cc.

16.5.5 Member Data Documentation

16.5.5.1 Real* mtk::Div1D::coeffs_interior_ [private]

Definition at line 202 of file mtk_div_1d.h.

16.5.5.2 int mtk::Div1D::dim_null_ [private]

Definition at line 194 of file mtk_div_1d.h.

16.5.5.3 Real* mtk::Div1D::divergence_ [private]

Definition at line 207 of file mtk_div_1d.h.

16.5.5.4 int mtk::Div1D::divergence_length_ [private]

Definition at line 196 of file mtk_div_1d.h.

16.5.5.5 Real* mtk::Div1D::mim_bndy_ [private]

Definition at line 206 of file mtk_div_1d.h.

16.5.5.6 Real mtk::Div1D::mimetic_threshold_ [private]

Definition at line 209 of file mtk_div_1d.h.

16.5.5.7 int mtk::Div1D::minrow_ [private]

Definition at line 197 of file mtk_div_1d.h.

16.5.5.8 int mtk::Div1D::num_bndy_coeffs_ [private]

Definition at line 195 of file mtk_div_1d.h.

16.5.5.9 int mtk::Div1D::order_accuracy_ [private]

Definition at line 193 of file mtk_div_1d.h.

16.5.5.10 Real* mtk::Div1D::prem_apps_ [private]

Definition at line 203 of file mtk_div_1d.h.

16.5.5.11 DenseMatrix mtk::Div1D::rat_basis_null_space_ [private]

Definition at line 200 of file mtk_div_1d.h.

16.5.5.12 int mtk::Div1D::row_ [private]

Definition at line 198 of file mtk_div_1d.h.

16.5.5.13 Real* mtk::Div1D::weights_cbs_ [private]

Definition at line 205 of file mtk_div_1d.h.

16.5.5.14 Real* mtk::Div1D::weights_crs_ [private]

Definition at line 204 of file mtk_div_1d.h.

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

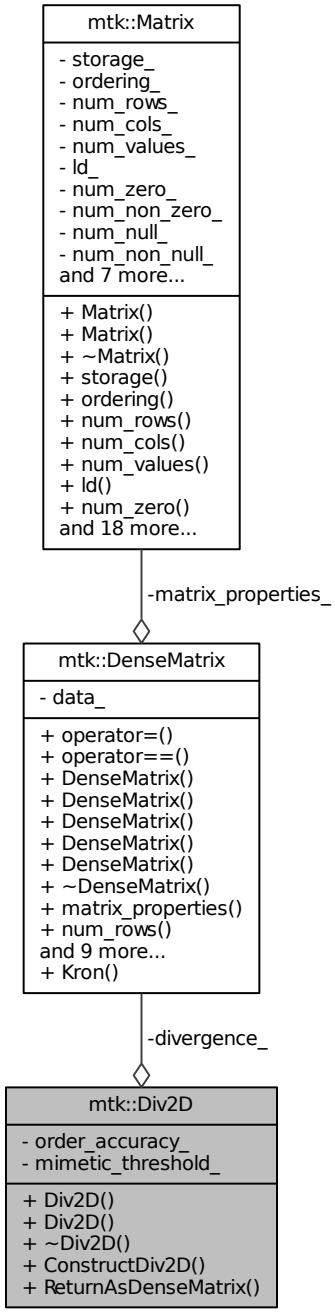
- include/mtk_div_1d.h
- src/mtk_div_1d.cc

16.6 mtk::Div2D Class Reference

Implements a 2D mimetic divergence operator.

`#include <mtk_div_2d.h>`

Collaboration diagram for mtk::Div2D:



Public Member Functions

- Div2D ()

Default constructor.

- Div2D (const Div2D &div)

  Copy constructor.

- ~Div2D ()

  Destructor.

- bool ConstructDiv2D (const UniStgGrid2D &grid, int order_accuracy=kDefaultOrderAccuracy, Real mimetic_threshold=kDefaultMimeticThreshold)

  Factory method implementing the CBS Algorithm to build operator.

- DenseMatrix ReturnAsDenseMatrix () const

  Return the operator as a dense matrix.

Private Attributes

- DenseMatrix divergence_

  Actual operator.

- int order_accuracy_

  Order of accuracy.

- Real mimetic_threshold_

  Mimetic Threshold.

16.6.1 Detailed Description

This class implements a 2D divergence operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm (CBSA).

Definition at line 76 of file mtk_div_2d.h.

16.6.2 Constructor & Destructor Documentation

16.6.2.1 mtk::Div2D::Div2D ( )

Definition at line 69 of file mtk_div_2d.cc.

16.6.2.2 mtk::Div2D::Div2D (const Div2D & *div*)

Parameters

in	*div*	Given divergence.

Definition at line 73 of file mtk_div_2d.cc.

16.6.2.3 mtk::Div2D::~Div2D ( )

Definition at line 77 of file mtk_div_2d.cc.

16.6.3 Member Function Documentation

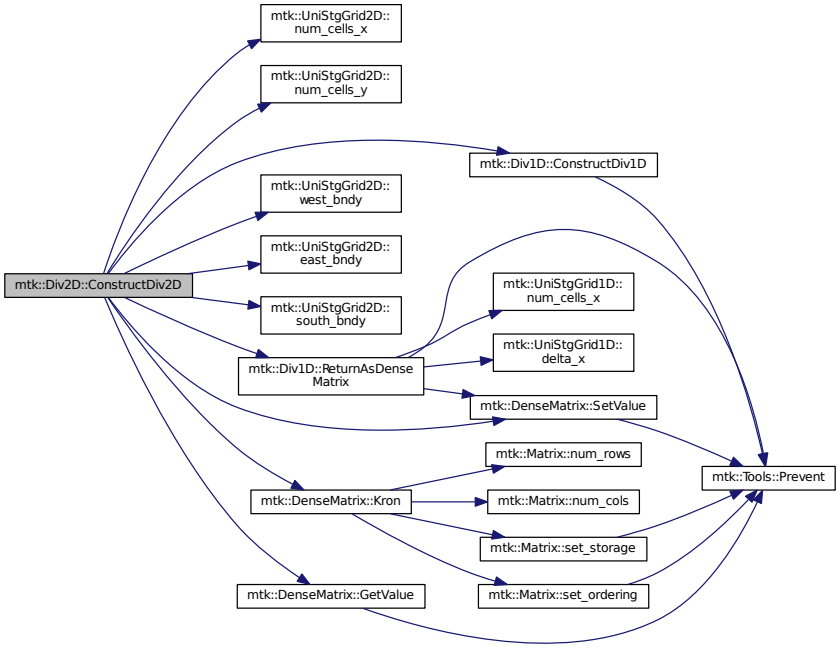
16.6.3.1 bool mtk::Div2D::ConstructDiv2D (const UniStgGrid2D & *grid*, int *order_accuracy* = kDefaultOrderAccuracy, mtk::Real *mimetic_threshold* = kDefaultMimeticThreshold)

Returns

Success of the construction.

Definition at line 79 of file mtk_div_2d.cc.

Here is the call graph for this function:



Here is the caller graph for this function:

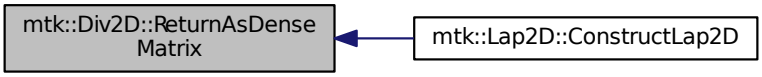
16.6.3.2 mtk::DenseMatrix mtk::Div2D::ReturnAsDenseMatrix ( ) const

Returns

The operator as a dense matrix.

Definition at line 145 of file mtk_div_2d.cc.

Here is the caller graph for this function:



16.6.4 Member Data Documentation

16.6.4.1 DenseMatrix mtk::Div2D::divergence_ [private]

Definition at line 108 of file mtk_div_2d.h.

16.6.4.2 Real mtk::Div2D::mimetic_threshold_ [private]

Definition at line 112 of file mtk_div_2d.h.

16.6.4.3 int mtk::Div2D::order_accuracy_ [private]

Definition at line 110 of file mtk_div_2d.h.

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

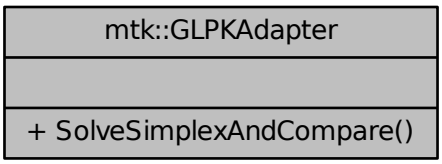
- include/mtk_div_2d.h
- src/mtk_div_2d.cc

16.7 mtk::GLPKAdapter Class Reference

Adapter class for the GLPK API.

`#include <mtk_glpk_adapter.h>`

Collaboration diagram for mtk::GLPKAdapter:



Static Public Member Functions

- static mtk::Real SolveSimplexAndCompare (mtk::Real *A, int nrows, int ncols, int kk, mtk::Real *hh, mtk::Real *qq, int robjective, mtk::Real mimetic_tol, int copy)

  Solves a CLO problem and compares the solution to a reference solution.

16.7.1 Detailed Description

This class contains a collection of static classes, that posses direct access to the underlying structure of the matrices, thus allowing programmers to exploit some of the numerical methods implemented in the GLPK.

The **GLPK (GNU Linear Programming Kit)** package is intended for solving large-scale linear programming (LP), mixed integer programming (MIP), and other related problems. It is a set of routines written in ANSI C and organized in the form of a callable library.

Warning

We use the GLPK temporarily in order to test the CBSA, but it will be removed due to potential licensing issues.

See also

http://www.gnu.org/software/glpk/

Todo Rescind from the GLPK as the numerical core for CLO problems.

Definition at line 101 of file mtk_glpk_adapter.h.

16.7.2 Member Function Documentation

16.7.2.1 mtk::Real mtk::GLPKAdapter::SolveSimplexAndCompare (mtk::Real * *A*, int *nrows*, int *ncols*, int *kk*, mtk::Real * *hh*, mtk::Real * *qq*, int *robjective*, mtk::Real *mimetic_tol*, int *copy*) [static]

This routine is the pivot of the CBSA. It solves a Constrained Linear Optimization (CLO) problem, and it compares the attained solution to a given reference solution. This comparison is done computing the norm-2 relative error.

Parameters

in	*alpha*	First scalar.
in	*AA*	Given matrix.
in	*xx*	First vector.
in	*beta*	Second scalar.
in	*beta*	Second scalar.
in,out	*yy*	Second vector (output).
in	*xx*	First vector.
in	*beta*	Second scalar.
in	*beta*	Second scalar.

Returns

Relative error computed between attained solution and provided ref.

Warning

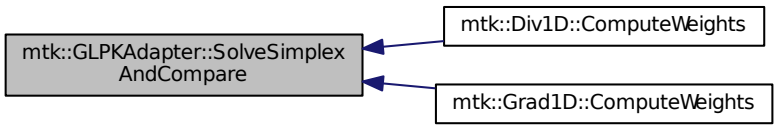
GLPK indexes in [1,n], so we must get the extra space needed.

1. Memory allocation.
2. Fill the problem.
3. Copy the row to the vector objective.
4. Forming the RHS.
5. Setting up the objective function.
6. Setting up constraints.
7. Copy the matrix minus the row objective to the glpk problem.
8. Solve problem.

Definition at line 76 of file mtk_glpk_adapter.cc.

Here is the call graph for this function:

Here is the caller graph for this function:



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

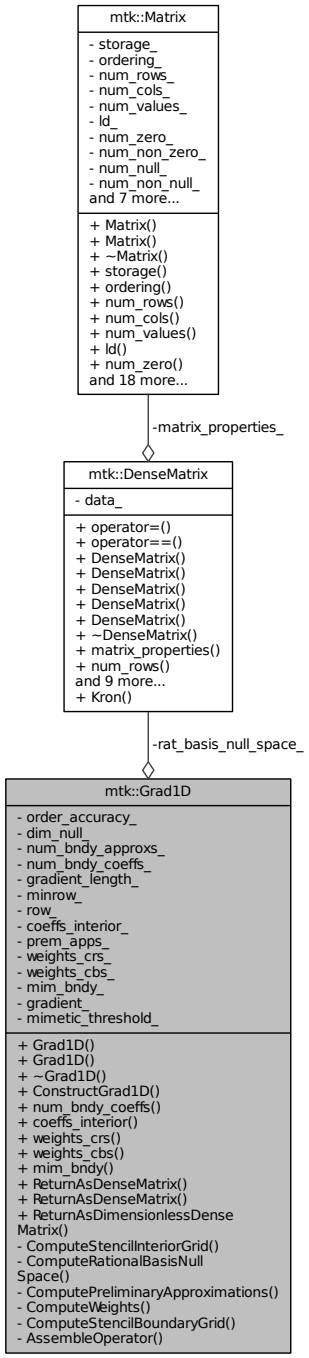
- include/mtk_glpk_adapter.h
- src/mtk_glpk_adapter.cc

16.8 mtk::Grad1D Class Reference

Implements a 1D mimetic gradient operator.

`#include <mtk_grad_1d.h>`

Collaboration diagram for mtk::Grad1D:



Public Member Functions

- Grad1D ()

Default constructor.

- Grad1D (const Grad1D &grad)

  Copy constructor.

- ~Grad1D ()

  Destructor.

- bool ConstructGrad1D (int order_accuracy=kDefaultOrderAccuracy, Real mimetic_threshold=kDefaultMimetic↩ Threshold)

  Factory method implementing the CBS Algorithm to build operator.

- int num_bndy_coeffs () const

  Returns how many coefficients are approximating at the boundary.

- Real * coeffs_interior () const

  Returns coefficients for the interior of the grid.

- Real * weights_crs (void) const

  Returns collection of weights as computed by the CRSA.

- Real * weights_cbs (void) const

  Returns collection of weights as computed by the CBSA.

- DenseMatrix mim_bndy () const

  Return collection of mimetic approximations at the boundary.

- DenseMatrix ReturnAsDenseMatrix (Real west, Real east, int num_cells_x) const

  Returns the operator as a dense matrix.

- DenseMatrix ReturnAsDenseMatrix (const UniStgGrid1D &grid) const

  Returns the operator as a dense matrix.

- DenseMatrix ReturnAsDimensionlessDenseMatrix (int num_cells_x) const

  Returns the operator as a dimensionless dense matrix.

Private Member Functions

- bool ComputeStencilInteriorGrid (void)

  Stage 1 of the CBS Algorithm.

- bool ComputeRationalBasisNullSpace (void)

  Stage 2.1 of the CBS Algorithm.

- bool ComputePreliminaryApproximations (void)

  Stage 2.2 of the CBS Algorithm.

- bool ComputeWeights (void)

  Stage 2.3 of the CBS Algorithm.

- bool ComputeStencilBoundaryGrid (void)

  Stage 2.4 of the CBS Algorithm.

- bool AssembleOperator (void)

  Stage 3 of the CBS Algorithm.

Private Attributes

- int order_accuracy_

  Order of numerical accuracy of the operator.

- int dim_null_

  Dim. null-space for boundary approximations.

- int num_bndy_approxs_

Req. approximations at and near the boundary.

- int num_bndy_coeffs_

  Req. coeffs. per bndy pt. uni. order accuracy.

- int gradient_length_

  Length of the output array.

- int minrow_

  Row from the optimizer with the minimum rel. nor.

- int row_

  Row currently processed by the optimizer.

- DenseMatrix rat_basis_null_space_

  Rational b. null-space w. bndy.

- Real * coeffs_interior_

  Interior stencil.

- Real * prem_apps_

  2D array of boundary preliminary approximations.

- Real * weights_crs_

  Array containing weights from CRSA.

- Real * weights_cbs_

  Array containing weights from CBSA.

- Real * mim_bndy_

  Array containing mimetic boundary approximations.

- Real * gradient_

  Output array containing the operator and weights.

- Real mimetic_threshold_

  < Mimetic threshold.

Friends

- std::ostream & operator<< (std::ostream &stream, Grad1D &in)

  Output stream operator for printing.

16.8.1 Detailed Description

This class implements a 1D gradient operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm (CBSA).

Definition at line 81 of file mtk_grad_1d.h.

16.8.2 Constructor & Destructor Documentation

16.8.2.1 mtk::Grad1D::Grad1D ( )

Definition at line 129 of file mtk_grad_1d.cc.

16.8.2.2 mtk::Grad1D::Grad1D (const Grad1D & *grad*)

Parameters

in	*div*	Given divergence.

Definition at line 145 of file mtk_grad_1d.cc.

16.8.2.3 mtk::Grad1D::~Grad1D ( )

Definition at line 161 of file mtk_grad_1d.cc.

16.8.3 Member Function Documentation

16.8.3.1 bool mtk::Grad1D::AssembleOperator (void) [private]

Construct the output array with the operator and its weights.

1. The first entry of the array will contain the order of accuracy.
2. The second entry of the array will contain the collection of approximating coefficients for the interior of the grid.
3. The third entry will contain the collection of weights.
4. The next dim_null + 1 entries will contain the collections of approximating coefficients for the west boundary of the grid.

Definition at line 1499 of file mtk_grad_1d.cc.

16.8.3.2 mtk::Real * mtk::Grad1D::coeffs_interior ( ) const

Returns

Coefficients for the interior of the grid.

Definition at line 330 of file mtk_grad_1d.cc.

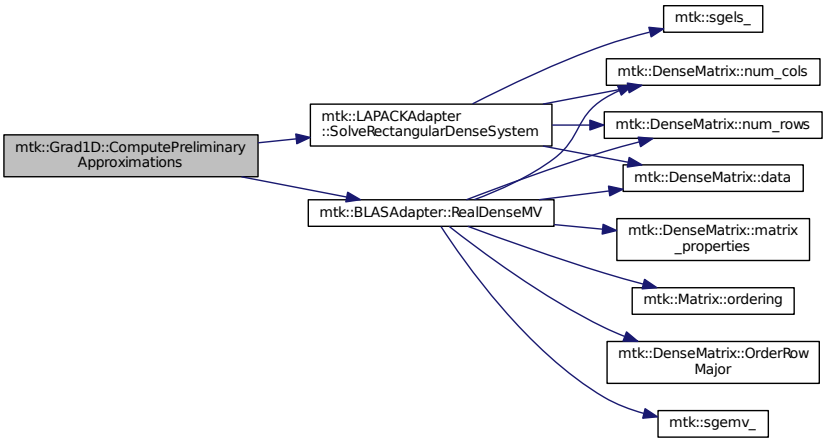
16.8.3.3 bool mtk::Grad1D::ComputePreliminaryApproximations (void) [private]

Compute the set of preliminary approximations on the boundary neighborhood.

1. Create generator vector for the first approximation.
2. Compute the dim_null near-the-boundary columns of the pi matrix.
3. Create the Vandermonde matrix for this iteration.
4. New order-selector vector (gets re-written with LAPACK solutions).
5. Solving TT*rr = ob yields the columns rr of the kk matrix.
6. Scale the kk matrix to make it a rational basis for null-space.
7. Extract the last dim_null values of the pre-scaled ob.
8. Once we posses the bottom elements, we proceed with the scaling.

Definition at line 833 of file mtk_grad_1d.cc.

Here is the call graph for this function:



16.8.3.4 bool mtk::Grad1D::ComputeRationalBasisNullSpace (void) [private]

Compute a rational basis for the null-space of the Vandermonde matrix approximating at the west boundary.

1. Create generator vector for the first approximation.

2. Create Vandermonde matrix.

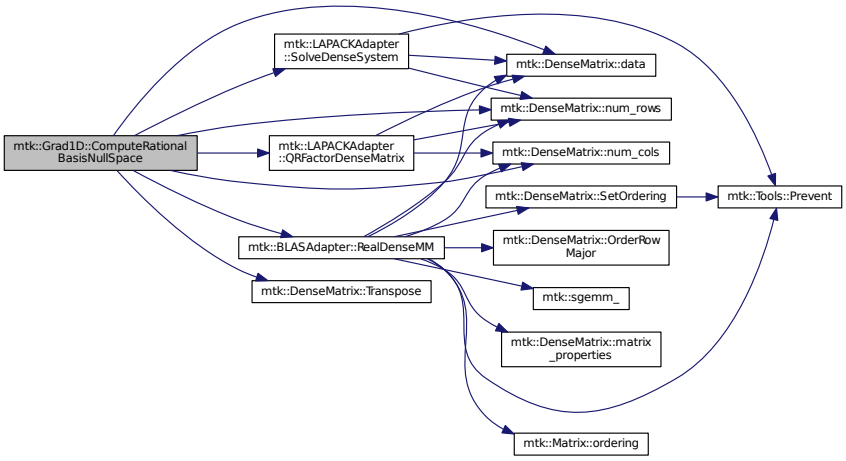
3. QR-factorize the Vandermonde matrix.

4. Extract the basis for the null-space from Q matrix.

5. Scale null-space to make it rational.

Definition at line 650 of file mtk_grad_1d.cc.

Here is the call graph for this function:



16.8.3.5 bool mtk::Grad1D::ComputeStencilBoundaryGrid (void) [private]

Compute mimetic stencil approximating at boundary.

1. Collect lambda values.
2. Compute alpha values.
3. Compute the mimetic boundary approximations.

Definition at line 1393 of file mtk_grad_1d.cc.

16.8.3.6 bool mtk::Grad1D::ComputeStencilInteriorGrid (void) [private]

Compute the stencil approximating the interior of the staggered grid.

1. Create vector for interior spatial coordinates.
2. Create Vandermonde matrix (using interior coordinates as generator).
3. Create order-selector vector.
4. Solve dense Vandermonde system to attain the interior coefficients.

Definition at line 554 of file mtk_grad_1d.cc.

Here is the call graph for this function:

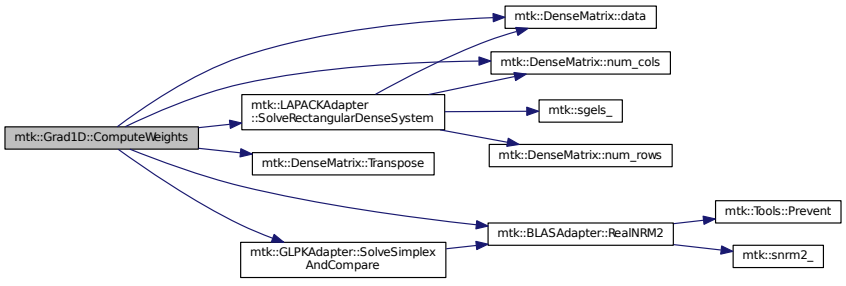
16.8.3.7 bool mtk::Grad1D::ComputeWeights (void) [private]

Compute the set of mimetic weights to impose the mimetic condition.

1. Construct the ■ matrix.
2. Use interior stencil to build proper RHS vector $\mathbf{h}$.
3. Get weights (as **CRSA**): $\blacksquare\mathbf{q} = \mathbf{h}$.
4. If required order is greater than critical order, start the **CBSA**.
5. Create ■ matrix from ■.
6. Prepare constraint vector as in the CBSA: ■.
7. Brute force search through all the rows of the Φ matrix.
8. Apply solution found from brute force search.

Definition at line 1053 of file mtk_grad_1d.cc.

Here is the call graph for this function:



16.8.3.8 bool mtk::Grad1D::ConstructGrad1D (int *order_accuracy* = kDefaultOrderAccuracy, Real *mimetic_threshold* = kDefaultMimeticThreshold)

Returns

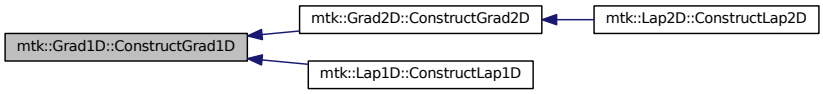
Success of the solution.

1. Compute stencil for the interior cells.
2. Compute a rational null-space from the first matrix transposed.
3. Compute preliminary approximation (non-mimetic) on the boundaries.
4. Compute quadrature weights to impose the mimetic conditions.
5. Compute real approximation (mimetic) on the boundaries.
6. Assemble operator.

Definition at line 182 of file mtk_grad_1d.cc.

Here is the call graph for this function:

Here is the caller graph for this function:



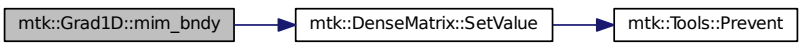
16.8.3.9 mtk::DenseMatrix mtk::Grad1D::mim_bndy ( ) const

Returns

Collection of mimetic approximations at the boundary.

Definition at line 345 of file mtk_grad_1d.cc.

Here is the call graph for this function:



16.8.3.10 int mtk::Grad1D::num_bndy_coeffs ( ) const

Returns

How many coefficients are approximating at the boundary.

Definition at line 325 of file mtk_grad_1d.cc.

16.8.3.11 mtk::DenseMatrix mtk::Grad1D::ReturnAsDenseMatrix (mtk::Real *west*, mtk::Real *east*, int *num_cells_x*) const

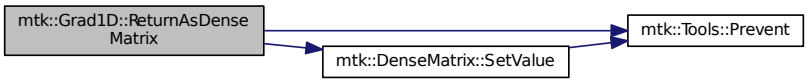
Returns

The operator as a dense matrix.

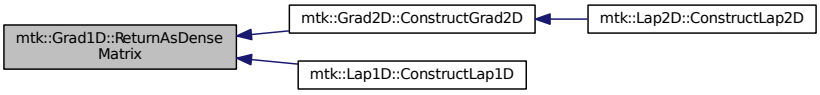
1. Insert mimetic boundary at the west.
2. Insert coefficients for the interior of the grid.
3. Impose center-skew symmetry by permuting the mimetic boundaries.

Definition at line 360 of file mtk_grad_1d.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



16.8.3.12 mtk::DenseMatrix mtk::Grad1D::ReturnAsDenseMatrix (const UniStgGrid1D & *grid*) const

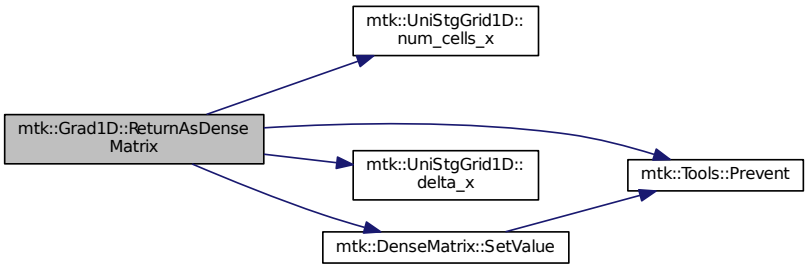
Returns

The operator as a dense matrix.

1. Insert mimetic boundary at the west.
2. Insert coefficients for the interior of the grid.
3. Impose center-skew symmetry by permuting the mimetic boundaries.

Definition at line 428 of file mtk_grad_1d.cc.

Here is the call graph for this function:



16.8.3.13 mtk::DenseMatrix mtk::Grad1D::ReturnAsDimensionlessDenseMatrix (int *num_cells_x*) const

Returns

The operator as a dimensionless dense matrix.

1. Insert mimetic boundary at the west.
2. Insert coefficients for the interior of the grid.
3. Impose center-skew symmetry by permuting the mimetic boundaries.

Definition at line 492 of file mtk_grad_1d.cc.

Here is the call graph for this function:

16.8.3.14 mtk::Real * mtk::Grad1D::weights_cbs (void) const

Returns

Collection of weights as computed by the CBSA.

Definition at line 340 of file mtk_grad_1d.cc.

16.8.3.15 mtk::Real * mtk::Grad1D::weights_crs (void) const

Returns

Success of the solution.

Definition at line 335 of file mtk_grad_1d.cc.

16.8.4 Friends And Related Function Documentation

16.8.4.1 std::ostream& operator<< (std::ostream & *stream,* mtk::Grad1D & *in*) [friend]

1. Print order of accuracy.
2. Print approximating coefficients for the interior.
3. Print mimetic weights.
4. Print mimetic approximations at the boundary.

Definition at line 79 of file mtk_grad_1d.cc.

16.8.5 Member Data Documentation

16.8.5.1 Real* mtk::Grad1D::coeffs_interior_ [private]

Definition at line 217 of file mtk_grad_1d.h.

16.8.5.2 int mtk::Grad1D::dim_null_ [private]

Definition at line 208 of file mtk_grad_1d.h.

16.8.5.3 Real* mtk::Grad1D::gradient_ [private]

Definition at line 222 of file mtk_grad_1d.h.

16.8.5.4 int mtk::Grad1D::gradient_length_ [private]

Definition at line 211 of file mtk_grad_1d.h.

16.8.5.5 Real* mtk::Grad1D::mim_bndy_ [private]

Definition at line 221 of file mtk_grad_1d.h.

16.8.5.6 Real mtk::Grad1D::mimetic_threshold_ [private]

Definition at line 224 of file mtk_grad_1d.h.

16.8.5.7 int mtk::Grad1D::minrow_ [private]

Definition at line 212 of file mtk_grad_1d.h.

16.8.5.8 int mtk::Grad1D::num_bndy_approxs_ [private]

Definition at line 209 of file mtk_grad_1d.h.

16.8.5.9 int mtk::Grad1D::num_bndy_coeffs_ [private]

Definition at line 210 of file mtk_grad_1d.h.

16.8.5.10 int mtk::Grad1D::order_accuracy_ [private]

Definition at line 207 of file mtk_grad_1d.h.

16.8.5.11 Real* mtk::Grad1D::prem_apps_ [private]

Definition at line 218 of file mtk_grad_1d.h.

16.8.5.12 DenseMatrix mtk::Grad1D::rat_basis_null_space_ [private]

Definition at line 215 of file mtk_grad_1d.h.

16.8.5.13 int mtk::Grad1D::row_ [private]

Definition at line 213 of file mtk_grad_1d.h.

16.8.5.14 Real* mtk::Grad1D::weights_cbs_ [private]

Definition at line 220 of file mtk_grad_1d.h.

16.8.5.15 Real* mtk::Grad1D::weights_crs_ [private]

Definition at line 219 of file mtk_grad_1d.h.

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

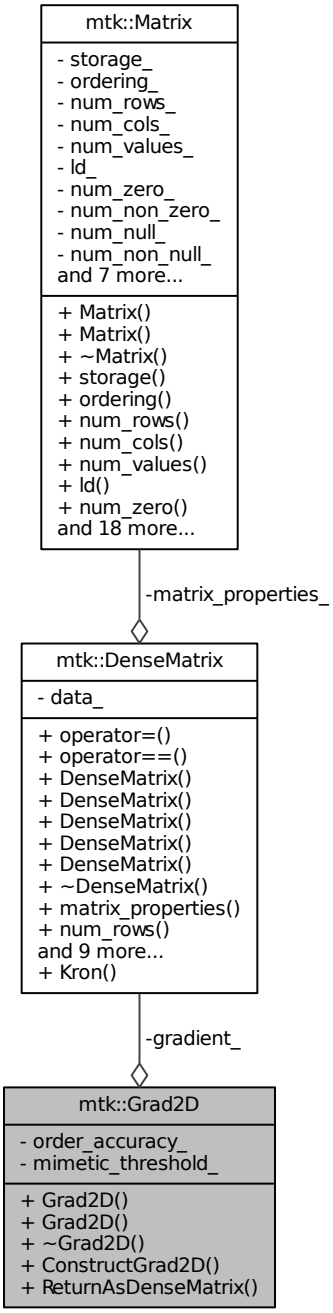
- include/mtk_grad_1d.h
- src/mtk_grad_1d.cc

16.9 mtk::Grad2D Class Reference

Implements a 2D mimetic gradient operator.

`#include <mtk_grad_2d.h>`

Collaboration diagram for mtk::Grad2D:



Public Member Functions

- Grad2D ()

Default constructor.

- Grad2D (const Grad2D &grad)

  Copy constructor.

- ~Grad2D ()

  Destructor.

- bool ConstructGrad2D (const UniStgGrid2D &grid, int order_accuracy=kDefaultOrderAccuracy, Real mimetic_threshold=kDefaultMimeticThreshold)

  Factory method implementing the CBS Algorithm to build operator.

- DenseMatrix ReturnAsDenseMatrix () const

  Return the operator as a dense matrix.

Private Attributes

- DenseMatrix gradient_

  Actual operator.

- int order_accuracy_

  Order of accuracy.

- Real mimetic_threshold_

  Mimetic Threshold.

16.9.1 Detailed Description

This class implements a 2D gradient operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm (CBSA).

Definition at line 76 of file mtk_grad_2d.h.

16.9.2 Constructor & Destructor Documentation

16.9.2.1 mtk::Grad2D::Grad2D ( )

Definition at line 67 of file mtk_grad_2d.cc.

16.9.2.2 mtk::Grad2D::Grad2D (const Grad2D & *grad*)

Parameters

in	*div*	Given divergence.

Definition at line 71 of file mtk_grad_2d.cc.

16.9.2.3 mtk::Grad2D::~Grad2D ( )

Definition at line 75 of file mtk_grad_2d.cc.

16.9.3 Member Function Documentation

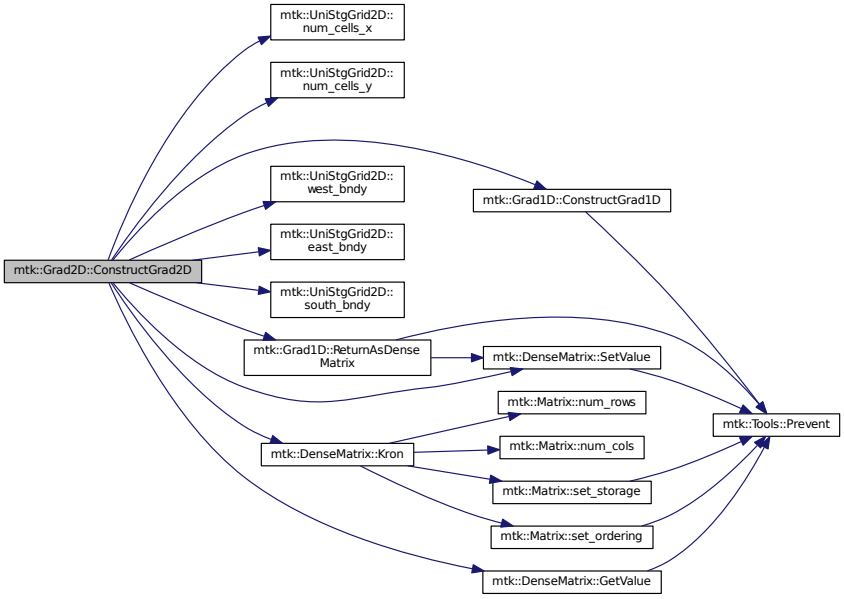
16.9.3.1 bool mtk::Grad2D::ConstructGrad2D (const UniStgGrid2D & *grid*, int *order_accuracy* = kDefaultOrderAccuracy, mtk::Real *mimetic_threshold* = kDefaultMimeticThreshold)

Returns

Success of the construction.

Definition at line 77 of file mtk_grad_2d.cc.

Here is the call graph for this function:



Here is the caller graph for this function:

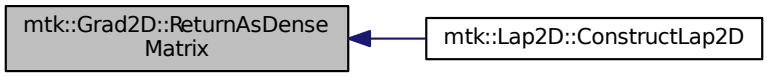
16.9.3.2 mtk::DenseMatrix mtk::Grad2D::ReturnAsDenseMatrix ( ) const

Returns

The operator as a dense matrix.

Definition at line 143 of file mtk_grad_2d.cc.

Here is the caller graph for this function:



16.9.4 Member Data Documentation

16.9.4.1 DenseMatrix mtk::Grad2D::gradient_ [private]

Definition at line 108 of file mtk_grad_2d.h.

16.9.4.2 Real mtk::Grad2D::mimetic_threshold_ [private]

Definition at line 112 of file mtk_grad_2d.h.

16.9.4.3 int mtk::Grad2D::order_accuracy_ [private]

Definition at line 110 of file mtk_grad_2d.h.

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

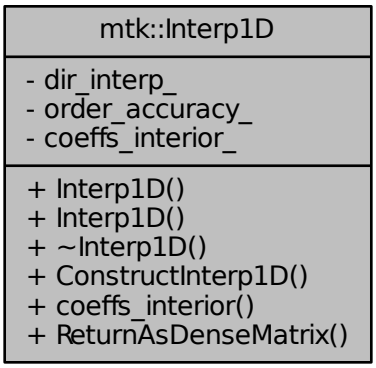
- include/mtk_grad_2d.h
- src/mtk_grad_2d.cc

16.10 mtk::Interp1D Class Reference

Implements a 1D interpolation operator.

`#include <mtk_interp_1d.h>`

Collaboration diagram for mtk::Interp1D:



Public Member Functions

- Interp1D ()

  Default constructor.
- Interp1D (const Interp1D &interp)

  Copy constructor.
- ~Interp1D ()

  Destructor.
- bool ConstructInterp1D (int order_accuracy=kDefaultOrderAccuracy, mtk::DirInterp dir=SCALAR_TO_VECTOR)

  Factory method to build operator.
- Real * coeffs_interior () const

  Returns coefficients for the interior of the grid.
- DenseMatrix ReturnAsDenseMatrix (const UniStgGrid1D &grid) const

  Returns the operator as a dense matrix.

Private Attributes

- DirInterp dir_interp_

  Direction of interpolation.
- int order_accuracy_

  Order of numerical accuracy of the operator.
- Real * coeffs_interior_

  Interior stencil.

Friends

- std::ostream & operator<< (std::ostream &stream, Interp1D &in)

  Output stream operator for printing.

16.10.1 Detailed Description

This class implements a 1D interpolation operator.

Definition at line 82 of file mtk_interp_1d.h.

16.10.2 Constructor & Destructor Documentation

16.10.2.1 mtk::Interp1D::Interp1D ( )

Definition at line 80 of file mtk_interp_1d.cc.

16.10.2.2 mtk::Interp1D::Interp1D (const Interp1D & *interp*)

Parameters

in	*interp*	Given interpolation operator.

Definition at line 85 of file mtk_interp_1d.cc.

16.10.2.3 mtk::Interp1D::~Interp1D ( )

Definition at line 90 of file mtk_interp_1d.cc.

16.10.3 Member Function Documentation

16.10.3.1 mtk::Real * mtk::Interp1D::coeffs_interior ( ) const

Returns

Coefficients for the interior of the grid.

Definition at line 130 of file mtk_interp_1d.cc.

16.10.3.2 bool mtk::Interp1D::ConstructInterp1D (int *order_accuracy* = kDefaultOrderAccuracy, mtk::DirInterp *dir* = SCALAR_TO_VECTOR)

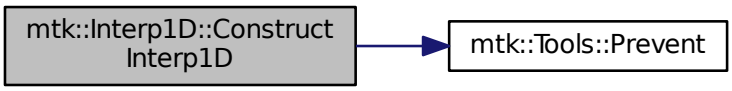
Returns

Success of the solution.

1. Compute stencil for the interior cells.

Definition at line 96 of file mtk_interp_1d.cc.

Here is the call graph for this function:



16.10.3.3 mtk::DenseMatrix mtk::Interp1D::ReturnAsDenseMatrix (const UniStgGrid1D & *grid*) const

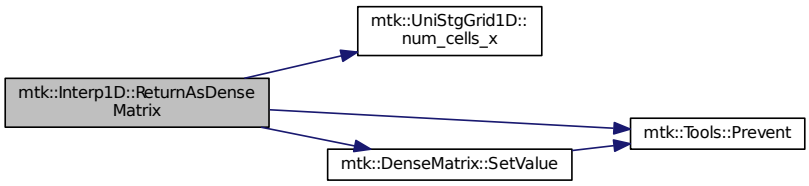
Returns

The operator as a dense matrix.

1. Preserve values at the boundary.
2. Insert coefficients for the interior of the grid.
3. Impose center-skew symmetry by permuting the boundaries.

Definition at line 135 of file mtk_interp_1d.cc.

Here is the call graph for this function:



16.10.4 Friends And Related Function Documentation

16.10.4.1 std::ostream& operator<< (std::ostream & *stream,* mtk::Interp1D & *in*) [friend]

1. Print approximating coefficients for the interior.

Definition at line 66 of file mtk_interp_1d.cc.

16.10.5 Member Data Documentation

16.10.5.1 Real* mtk::Interp1D::coeffs_interior_ [private]

Definition at line 127 of file mtk_interp_1d.h.

16.10.5.2 DirInterp mtk::Interp1D::dir_interp_ [private]

Definition at line 123 of file mtk_interp_1d.h.

16.10.5.3 int mtk::Interp1D::order_accuracy_ [private]

Definition at line 125 of file mtk_interp_1d.h.

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

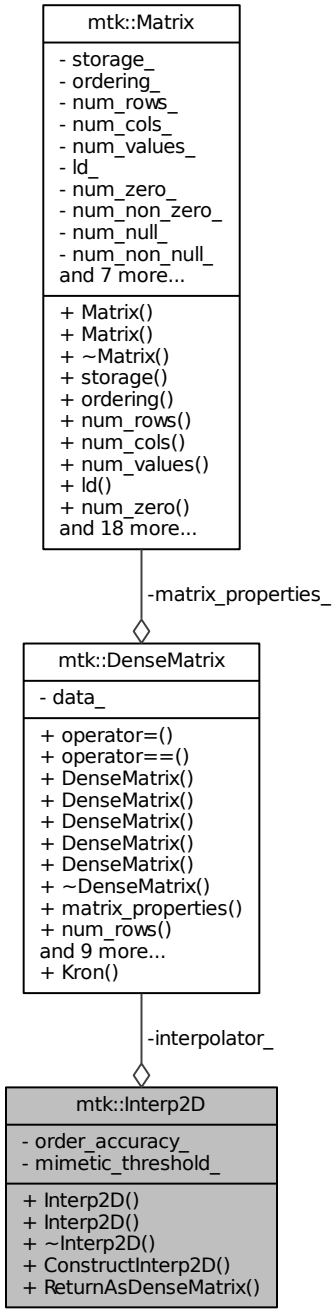
- include/mtk_interp_1d.h
- src/mtk_interp_1d.cc

16.11 mtk::Interp2D Class Reference

Implements a 2D interpolation operator.

`#include <mtk_interp_2d.h>`

Collaboration diagram for mtk::Interp2D:



Public Member Functions

- Interp2D ()

Default constructor.

- Interp2D (const Interp2D &interp)

  Copy constructor.

- ~Interp2D ()

  Destructor.

- DenseMatrix ConstructInterp2D (const UniStgGrid2D &grid, int order_accuracy=kDefaultOrderAccuracy, Real mimetic_threshold=kDefaultMimeticThreshold)

  Factory method implementing the CBS Algorithm to build operator.

- DenseMatrix ReturnAsDenseMatrix ()

  Return the operator as a dense matrix.

Private Attributes

- DenseMatrix interpolator_

  Actual operator.

- int order_accuracy_

  Order of accuracy.

- Real mimetic_threshold_

  Mimetic Threshold.

16.11.1 Detailed Description

This class implements a 2D interpolation operator.

Definition at line 76 of file mtk_interp_2d.h.

16.11.2 Constructor & Destructor Documentation

16.11.2.1 mtk::Interp2D::Interp2D ( )

16.11.2.2 mtk::Interp2D::Interp2D (const Interp2D & *interp*)

Parameters

in	*lap*	Given Laplacian.

16.11.2.3 mtk::Interp2D::~Interp2D ( )

16.11.3 Member Function Documentation

16.11.3.1 DenseMatrix mtk::Interp2D::ConstructInterp2D (const UniStgGrid2D & *grid*, int *order_accuracy* = kDefaultOrderAccuracy, Real *mimetic_threshold* = kDefaultMimeticThreshold)

Returns

Success of the construction.

16.11.3.2 DenseMatrix mtk::Interp2D::ReturnAsDenseMatrix ( )

Returns

The operator as a dense matrix.

16.11.4 Member Data Documentation

16.11.4.1 DenseMatrix mtk::Interp2D::interpolator_ [private]

Definition at line 108 of file mtk_interp_2d.h.

16.11.4.2 Real mtk::Interp2D::mimetic_threshold_ [private]

Definition at line 112 of file mtk_interp_2d.h.

16.11.4.3 int mtk::Interp2D::order_accuracy_ [private]

Definition at line 110 of file mtk_interp_2d.h.

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

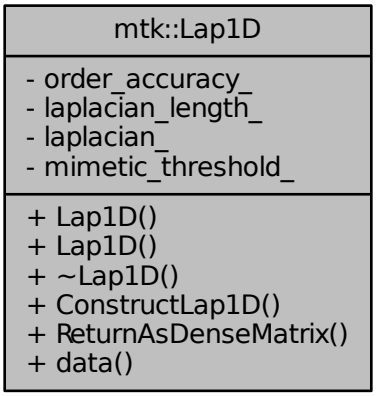
- include/mtk_interp_2d.h

16.12 mtk::Lap1D Class Reference

Implements a 1D mimetic Laplacian operator.

`#include <mtk_lap_1d.h>`

Collaboration diagram for mtk::Lap1D:



Public Member Functions

- Lap1D ()

  Default constructor.
- Lap1D (const Lap1D &lap)

  Copy constructor.
- ~Lap1D ()

  Destructor.
- bool ConstructLap1D (int order_accuracy=kDefaultOrderAccuracy, Real mimetic_threshold=kDefaultMimeticThreshold)

  Factory method implementing the CBS Algorithm to build operator.
- DenseMatrix ReturnAsDenseMatrix (const UniStgGrid1D &grid) const

  Return the operator as a dense matrix.
- const mtk::Real * data (const UniStgGrid1D &grid) const

  Return the operator as a dense array.

Private Attributes

- int order_accuracy_

  Order of numerical accuracy of the operator.
- int laplacian_length_

  Length of the output array.
- Real * laplacian_

  Output array containing the operator and weights.
- Real mimetic_threshold_

  < Mimetic threshold.

Friends

- std::ostream & operator<< (std::ostream &stream, Lap1D &in)

  Output stream operator for printing.

16.12.1 Detailed Description

This class implements a 1D Laplacian operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm (CBSA).

Definition at line 76 of file mtk_lap_1d.h.

16.12.2 Constructor & Destructor Documentation

16.12.2.1 mtk::Lap1D::Lap1D ( )

Definition at line 108 of file mtk_lap_1d.cc.

16.12.2.2 mtk::Lap1D::Lap1D (const Lap1D & *lap*)

Parameters

in	*lap*	Given Laplacian.

16.12.2.3 mtk::Lap1D::~Lap1D ( )

Definition at line 113 of file mtk_lap_1d.cc.

16.12.3 Member Function Documentation

16.12.3.1 bool mtk::Lap1D::ConstructLap1D (int *order_accuracy* = kDefaultOrderAccuracy, mtk::Real *mimetic_threshold* = kDefaultMimeticThreshold)

Returns

Success of the solution.

1. Create gradient operator using specific values for the Laplacian.
2. Create gradient operator using specific values for the Laplacian.
3. Create both operators as matrices.
4. Multiply both operators: $\breve{\mathbf{L}}_x^k = \breve{\mathbf{D}}_x^k \breve{\mathbf{G}}_x^k$
5. Extract the coefficients from the matrix and store them in the array.

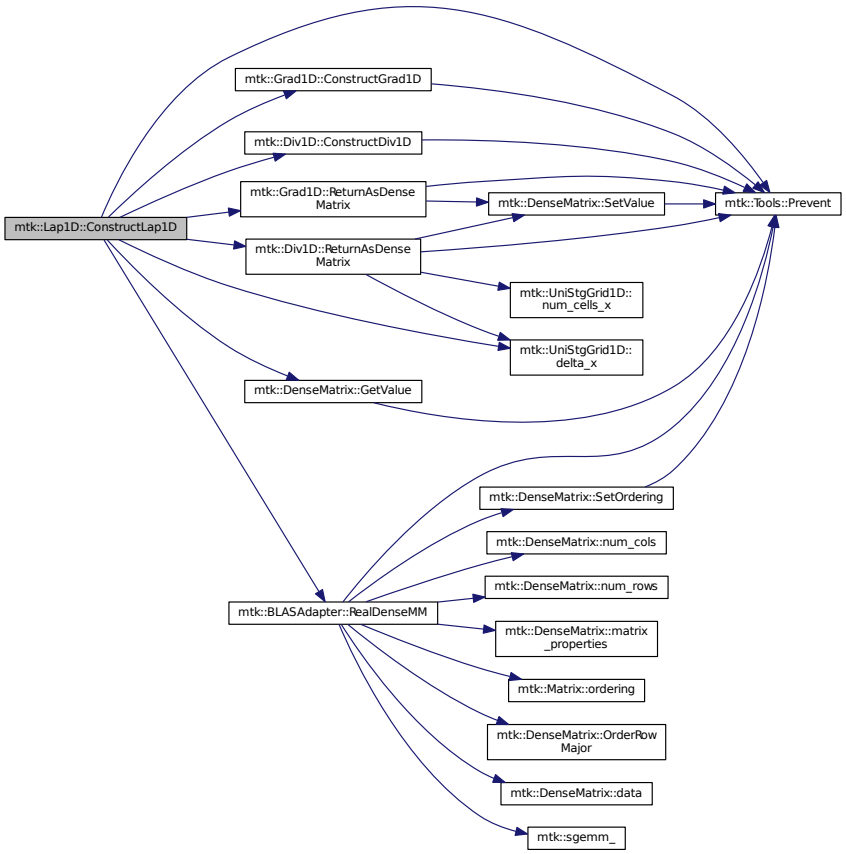
Warning

We do not compute weights for this operator.

1. The first entry of the array will contain the order of accuracy.
2. The second entry of the array will contain the collection of approximating coefficients for the interior of the grid.
3. We DO NOT have weights in this operator. Copy mimetic bndy coeffs.

Definition at line 119 of file mtk_lap_1d.cc.

Here is the call graph for this function:



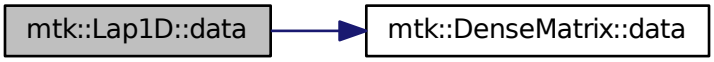
16.12.3.2 const mtk::Real * mtk::Lap1D::data (const UniStgGrid1D & *grid*) const

Returns

The operator as a dense array.

Definition at line 333 of file mtk_lap_1d.cc.

Here is the call graph for this function:



16.12.3.3 mtk::DenseMatrix mtk::Lap1D::ReturnAsDenseMatrix (const UniStgGrid1D & *grid*) const

Returns

The operator as a dense matrix.

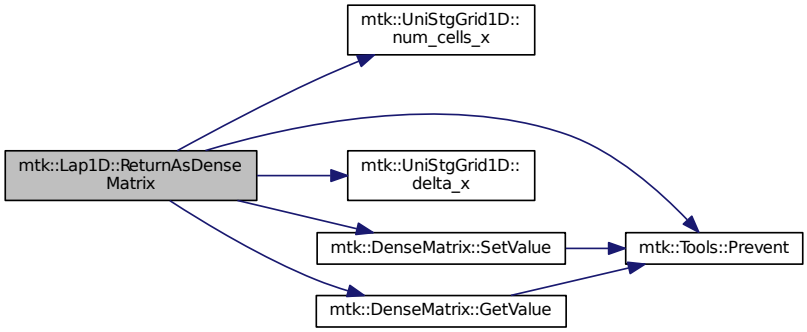
1. Extract mimetic coefficients from the west boundary.
2. Extract interior coefficients.
3. Extract mimetic coefficients from the west boundary to go east.

Note

We could create two matrices of the requested size and multiply them, but that would be inefficient, since we already have the computed coefficients stored. We just have to set them in place, in a matrix of an adequate size, and multiply them times the inverse of the square of the step size, in order for the matrix to actually represent a differential operator.

Definition at line 265 of file mtk_lap_1d.cc.

Here is the call graph for this function:



16.12.4 Friends And Related Function Documentation

16.12.4.1 std::ostream& operator<< (std::ostream & *stream,* mtk::Lap1D & *in*) [friend]

1. Print order of accuracy.
2. Print approximating coefficients for the interior.
3. No weights, thus print the mimetic boundary coefficients.

Definition at line 73 of file mtk_lap_1d.cc.

16.12.5 Member Data Documentation

16.12.5.1 Real* mtk::Lap1D::laplacian_ [private]

Definition at line 120 of file mtk_lap_1d.h.

16.12.5.2 int mtk::Lap1D::laplacian_length_ [private]

Definition at line 118 of file mtk_lap_1d.h.

16.12.5.3 Real mtk::Lap1D::mimetic_threshold_ [private]

Definition at line 122 of file mtk_lap_1d.h.

16.12.5.4 int mtk::Lap1D::order_accuracy_ [private]

Definition at line 117 of file mtk_lap_1d.h.

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

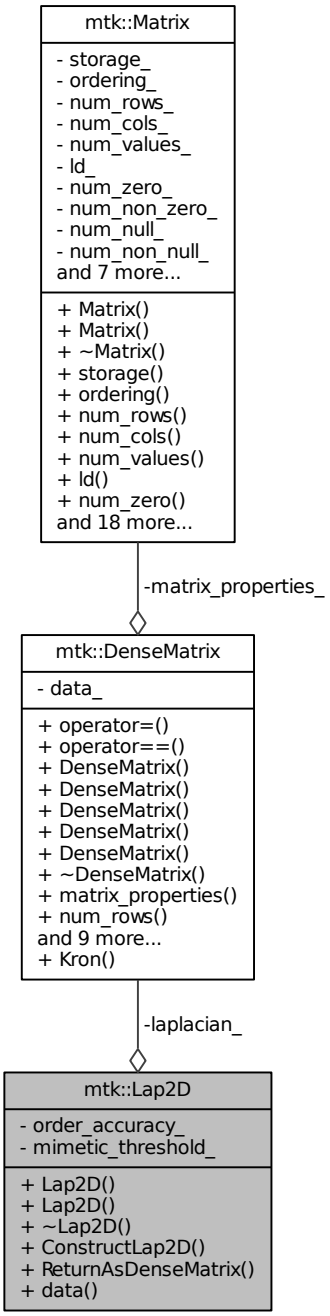
- include/mtk_lap_1d.h
- src/mtk_lap_1d.cc

16.13 mtk::Lap2D Class Reference

Implements a 2D mimetic Laplacian operator.

`#include <mtk_lap_2d.h>`

Collaboration diagram for mtk::Lap2D:



Public Member Functions

- Lap2D ()

Default constructor.

- Lap2D (const Lap2D &lap)

  Copy constructor.

- ~Lap2D ()

  Destructor.

- bool ConstructLap2D (const UniStgGrid2D &grid, int order_accuracy=kDefaultOrderAccuracy, Real mimetic_threshold=kDefaultMimeticThreshold)

  Factory method implementing the CBS Algorithm to build operator.

- DenseMatrix ReturnAsDenseMatrix () const

  Return the operator as a dense matrix.

- Real * data () const

  Return the operator as a dense array.

Private Attributes

- DenseMatrix laplacian_

  Actual operator.

- int order_accuracy_

  Order of accuracy.

- Real mimetic_threshold_

  Mimetic Threshold.

16.13.1 Detailed Description

This class implements a 2D Laplacian operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm (CBSA).

Definition at line 76 of file mtk_lap_2d.h.

16.13.2 Constructor & Destructor Documentation

16.13.2.1 mtk::Lap2D::Lap2D ( )

Definition at line 69 of file mtk_lap_2d.cc.

16.13.2.2 mtk::Lap2D::Lap2D (const Lap2D & *lap*)

Parameters

in	*lap*	Given Laplacian.

Definition at line 71 of file mtk_lap_2d.cc.

16.13.2.3 mtk::Lap2D::~Lap2D ( )

Definition at line 75 of file mtk_lap_2d.cc.

16.13.3 Member Function Documentation

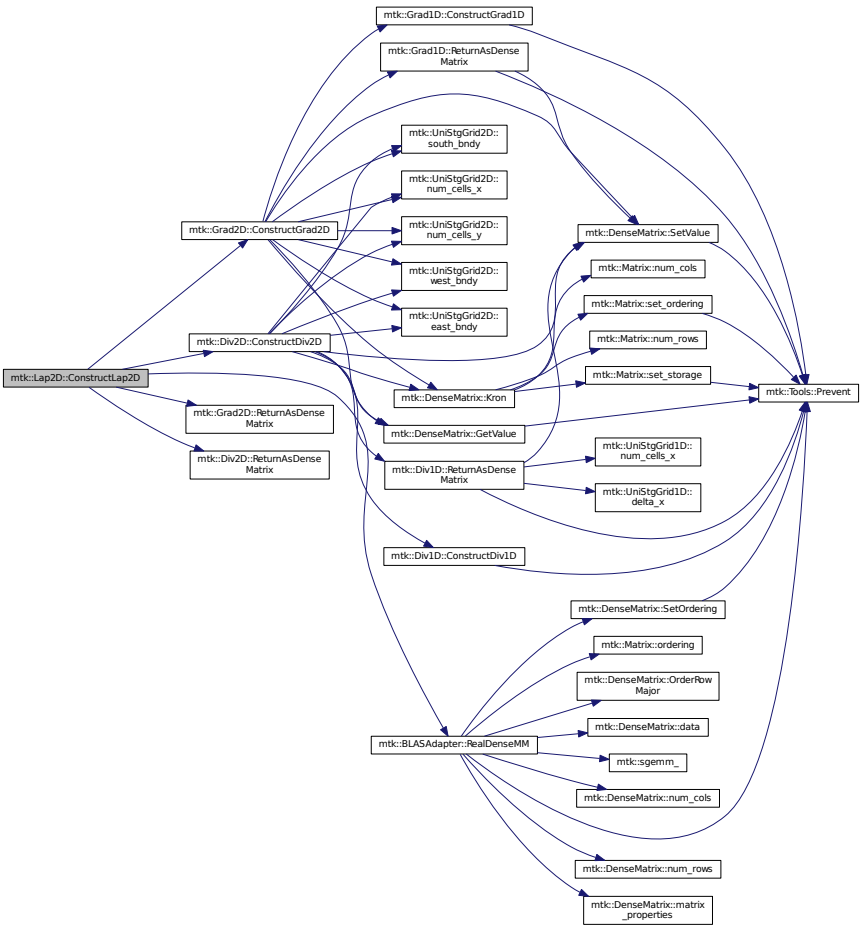
16.13.3.1 bool mtk::Lap2D::ConstructLap2D (const UniStgGrid2D & *grid*, int *order_accuracy* = kDefaultOrderAccuracy, mtk::Real *mimetic_threshold* = kDefaultMimeticThreshold)

Returns

Success of the construction.

Definition at line 77 of file mtk_lap_2d.cc.

Here is the call graph for this function:



16.13.3.2 mtk::Real * mtk::Lap2D::data ( ) const

Returns

The operator as a dense array.

Definition at line 111 of file mtk_lap_2d.cc.

16.13.3.3 DenseMatrix mtk::Lap2D::ReturnAsDenseMatrix ( ) const

Returns

The operator as a dense matrix.

Definition at line 106 of file mtk_lap_2d.cc.

16.13.4 Member Data Documentation

16.13.4.1 DenseMatrix mtk::Lap2D::laplacian_ [private]

Definition at line 115 of file mtk_lap_2d.h.

16.13.4.2 Real mtk::Lap2D::mimetic_threshold_ [private]

Definition at line 119 of file mtk_lap_2d.h.

16.13.4.3 int mtk::Lap2D::order_accuracy_ [private]

Definition at line 117 of file mtk_lap_2d.h.

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

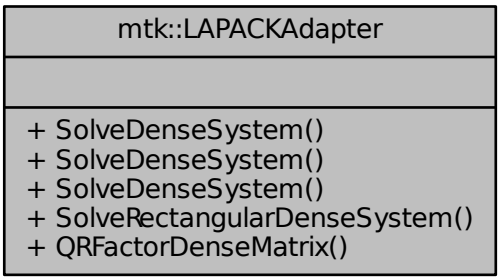
- include/mtk_lap_2d.h
- src/mtk_lap_2d.cc

16.14 mtk::LAPACKAdapter Class Reference

Adapter class for the LAPACK API.

`#include <mtk_lapack_adapter.h>`

Collaboration diagram for mtk::LAPACKAdapter:



Static Public Member Functions

- static int SolveDenseSystem (mtk::DenseMatrix &mm, mtk::Real *rhs)

  Solves a dense system of linear equations.
- static int SolveDenseSystem (mtk::DenseMatrix &mm, mtk::DenseMatrix &rr)

  Solves a dense system of linear equations.
- static int SolveDenseSystem (mtk::DenseMatrix &mm, mtk::UniStgGrid1D &rhs)

  Solves a dense system of linear equations.
- static int SolveRectangularDenseSystem (const mtk::DenseMatrix &aa, mtk::Real *ob_, int ob_ld_)

  Solves overdetermined or underdetermined real linear systems.
- static mtk::DenseMatrix QRFactorDenseMatrix (DenseMatrix &matrix)

  Performs a QR factorization on a dense matrix.

16.14.1 Detailed Description

This class contains a collection of static classes, that posses direct access to the underlying structure of the matrices, thus allowing programmers to exploit the numerical methods implemented in the LAPACK.

The **LAPACK (Linear Algebra PACKage)** is written in Fortran 90 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems.

See also

http://www.netlib.org/lapack/

Definition at line 92 of file mtk_lapack_adapter.h.

16.14.2 Member Function Documentation

16.14.2.1 mtk::DenseMatrix mtk::LAPACKAdapter::QRFactorDenseMatrix (mtk::DenseMatrix & *aa*) [static]

Adapts the MTK to LAPACK's routine.

Parameters

in,out	*matrix*	Input matrix.

Returns

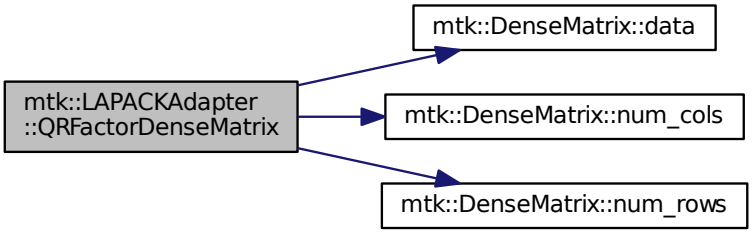
Matrix $\mathbf{Q}$.

Exceptions

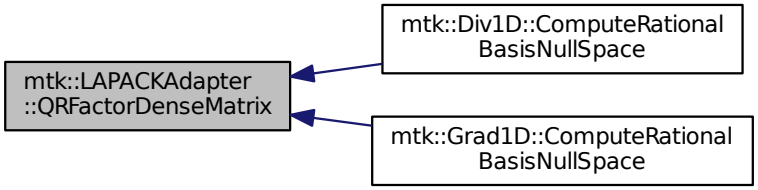
std::bad_alloc	

Definition at line 555 of file mtk_lapack_adapter.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



16.14.2.2 int mtk::LAPACKAdapter::SolveDenseSystem (mtk::DenseMatrix & *mm*, mtk::Real * *rhs*) [static]

Adapts the MTK to LAPACK's dgesv_ routine.

Parameters

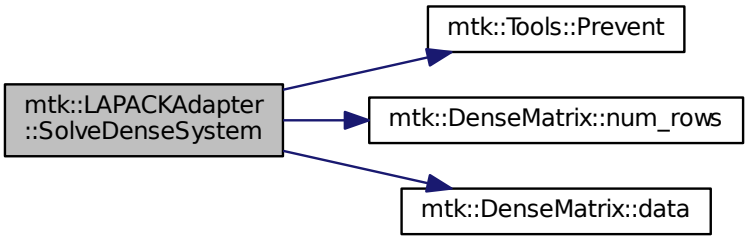
in	*matrix*	Input matrix.
in	*rhs*	Input right-hand sides vector.

Exceptions

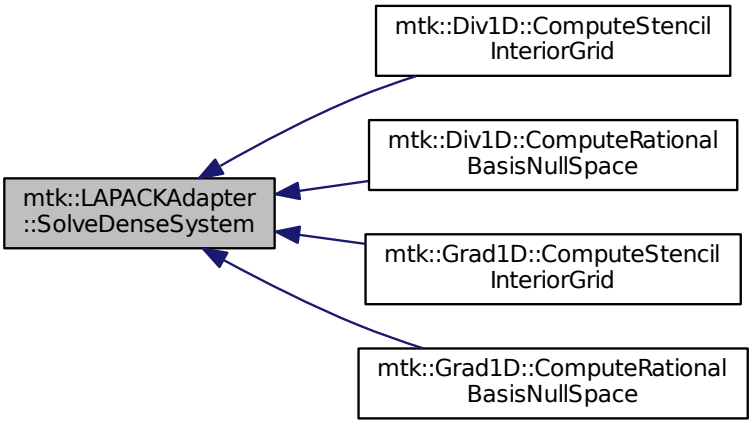
std::bad_alloc	

Definition at line 430 of file mtk_lapack_adapter.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



16.14.2.3 int mtk::LAPACKAdapter::SolveDenseSystem (mtk::DenseMatrix & *mm*, mtk::DenseMatrix & *rr*) [static]

Adapts the MTK to LAPACK's dgesv_ routine.

Parameters

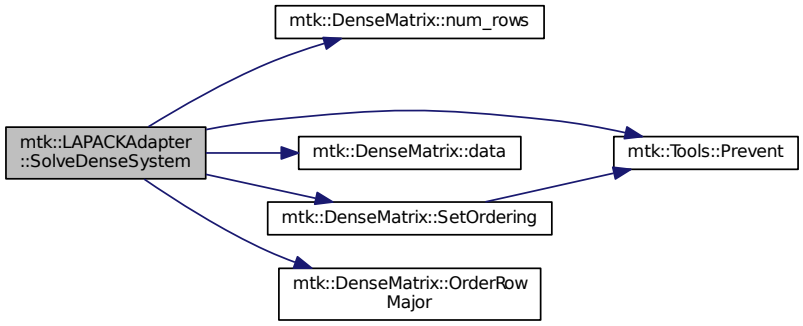
in	*matrix*	Input matrix.
in	*rr*	Input right-hand sides matrix.

Exceptions

std::bad_alloc	

Definition at line 465 of file mtk_lapack_adapter.cc.

Here is the call graph for this function:



16.14.2.4 int mtk::LAPACKAdapter::SolveDenseSystem (mtk::DenseMatrix & *mm*, mtk::UniStgGrid1D & *rhs*) [static]

Adapts the MTK to LAPACK's dgesv_ routine.

Parameters

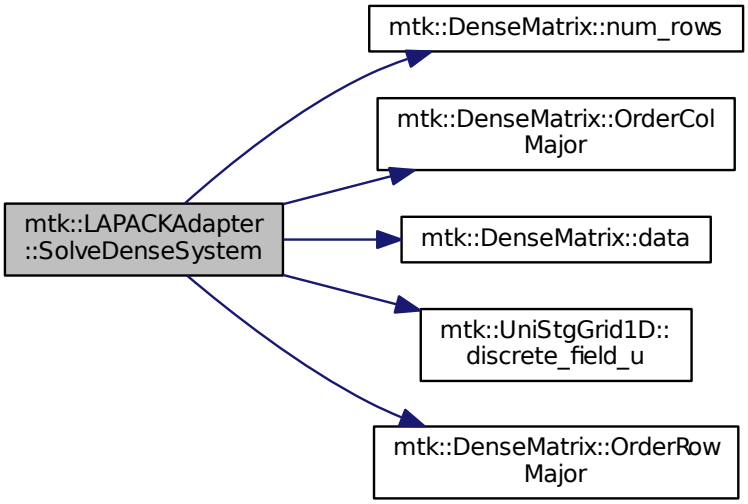
in	*matrix*	Input matrix.
in	*rr*	Input right-hand side from info on a grid.

Exceptions

std::bad_alloc	

Definition at line 517 of file mtk_lapack_adapter.cc.

Here is the call graph for this function:



16.14.2.5 int mtk::LAPACKAdapter::SolveRectangularDenseSystem (const mtk::DenseMatrix & *aa*, mtk::Real * *ob_*, int *ob_ld_*) [static]

Adapts the MTK to LAPACK's routine.

Parameters

in,out	*matrix*	Input matrix.

Returns

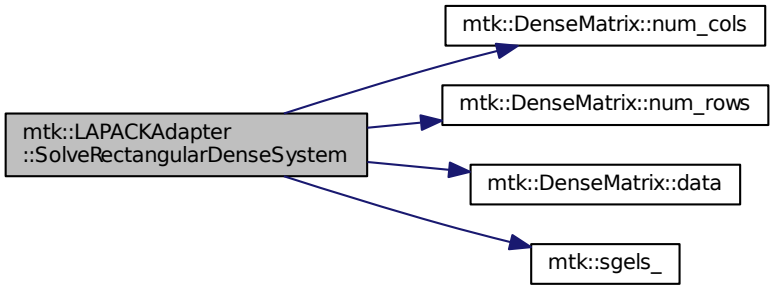
Success of the solution.

Exceptions

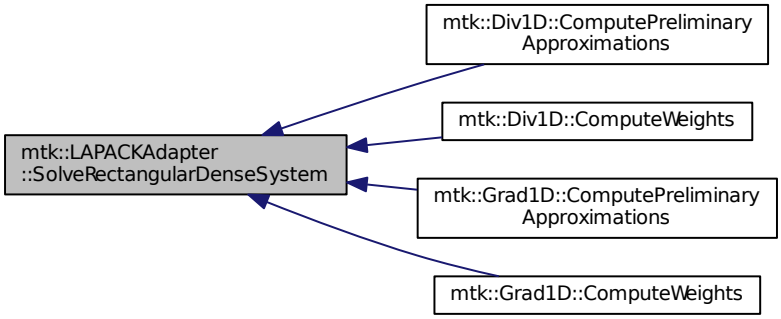
std::bad_alloc	

Definition at line 756 of file mtk_lapack_adapter.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



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

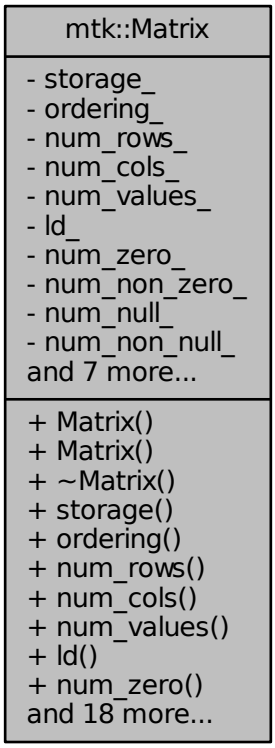
- include/mtk_lapack_adapter.h
- src/mtk_lapack_adapter.cc

16.15 mtk::Matrix Class Reference

Definition of the representation of a matrix in the MTK.

`#include <mtk_matrix.h>`

Collaboration diagram for mtk::Matrix:



Public Member Functions

- Matrix ()

  Default constructor.
- Matrix (const Matrix &in)

  Copy constructor.
- ~Matrix () noexcept

  Destructor.
- MatrixStorage storage () const noexcept

  Gets the type of storage of this matrix.
- MatrixOrdering ordering () const noexcept

  Gets the type of ordering of this matrix.
- int num_rows () const noexcept

  Gets the number of rows.
- int num_cols () const noexcept

  Gets the number of rows.

- int num_values () const noexcept

  Gets the number of values.
- int ld () const noexcept

  Gets the matrix' leading dimension.
- int num_zero () const noexcept

  Gets the number of zeros.
- int num_non_zero () const noexcept

  Gets the number of non-zero values.
- int num_null () const noexcept

  Gets the number of null values.
- int num_non_null () const noexcept

  Gets the number of non-null values.
- int kl () const noexcept

  Gets the number of lower diagonals.
- int ku () const noexcept

  Gets the number of upper diagonals.
- int bandwidth () const noexcept

  Gets the bandwidth.
- Real abs_density () const noexcept

  Gets the absolute density.
- Real rel_density () const noexcept

  Gets the relative density.
- Real abs_sparsity () const noexcept

  Gets the Absolute sparsity.
- Real rel_sparsity () const noexcept

  Gets the Relative sparsity.
- void set_storage (const MatrixStorage &tt) noexcept

  Sets the storage type of the matrix.
- void set_ordering (const MatrixOrdering &oo) noexcept

  Sets the ordering of the matrix.
- void set_num_rows (const int &num_rows) noexcept

  Sets the number of rows of the matrix.
- void set_num_cols (const int &num_cols) noexcept

  Sets the number of columns of the matrix.
- void set_num_zero (const int &in) noexcept

  Sets the number of zero values of the matrix that matter.
- void set_num_null (const int &in) noexcept

  Sets the number of zero values of the matrix that DO NOT matter.
- void IncreaseNumZero () noexcept

  Increases the number of values that equal zero but with meaning.
- void IncreaseNumNull () noexcept

  Increases the number of values that equal zero but with no meaning.

Private Attributes

- MatrixStorage storage_

  What type of matrix is this?
- MatrixOrdering ordering_

  What kind of ordering is it following?
- int num_rows_

  Number of rows.
- int num_cols_

  Number of columns.
- int num_values_

  Number of total values in matrix.
- int ld_

  Elements between successive rows when row-major.
- int num_zero_

  Number of zeros.
- int num_non_zero_

  Number of non-zero values.
- int num_null_

  Number of null (insignificant) values.
- int num_non_null_

  Number of null (significant) values.
- int kl_

  Number of lower diagonals on a banded matrix.
- int ku_

  Number of upper diagonals on a banded matrix.
- int bandwidth_

  Bandwidth of the matrix.
- Real abs_density_

  Absolute density of matrix.
- Real rel_density_

  Relative density of matrix.
- Real abs_sparsity_

  Absolute sparsity of matrix.
- Real rel_sparsity_

  Relative sparsity of matrix.

16.15.1 Detailed Description

Definition of the representation for the matrices implemented in the MTK.

Definition at line 75 of file mtk_matrix.h.

16.15.2 Constructor & Destructor Documentation

16.15.2.1 mtk::Matrix::Matrix ( )

Definition at line 67 of file mtk_matrix.cc.

16.15.2.2 mtk::Matrix::Matrix (const Matrix & *in*)

Parameters

in	*in*	Given matrix.

Definition at line 86 of file mtk_matrix.cc.

16.15.2.3 mtk::Matrix::~Matrix ( ) [noexcept]

Definition at line 105 of file mtk_matrix.cc.

16.15.3 Member Function Documentation

16.15.3.1 Real mtk::Matrix::abs_density ( ) const [noexcept]

See also

http://www.csrc.sdsu.edu/research_reports/CSRCR2013-01.pdf

Returns

Absolute density of the matrix.

16.15.3.2 mtk::Real mtk::Matrix::abs_sparsity ( ) const [noexcept]

See also

http://www.csrc.sdsu.edu/research_reports/CSRCR2013-01.pdf

Returns

Absolute sparsity of the matrix.

Definition at line 177 of file mtk_matrix.cc.

16.15.3.3 int mtk::Matrix::bandwidth ( ) const [noexcept]

Returns

Bandwidth of the matrix.

Definition at line 167 of file mtk_matrix.cc.

16.15.3.4 void mtk::Matrix::IncreaseNumNull ( ) [noexcept]

Todo Review the definition of sparse matrices properties.

Definition at line 274 of file mtk_matrix.cc.

16.15.3.5 void mtk::Matrix::IncreaseNumZero ( ) [noexcept]

Todo Review the definition of sparse matrices properties.

Definition at line 264 of file mtk_matrix.cc.

16.15.3.6 int mtk::Matrix::kl ( ) const [noexcept]

Returns

Number of lower diagonals.

Definition at line 157 of file mtk_matrix.cc.

16.15.3.7 int mtk::Matrix::ku ( ) const [noexcept]

Returns

Number of upper diagonals.

Definition at line 162 of file mtk_matrix.cc.

16.15.3.8 int mtk::Matrix::ld ( ) const [noexcept]

Leading dimension of the data array is the number of elements between successive rows (for row major storage) in memory. Most of the cases, the leading dimension is the same as the number of columns.

Returns

Leading dimension of the matrix.

Definition at line 132 of file mtk_matrix.cc.

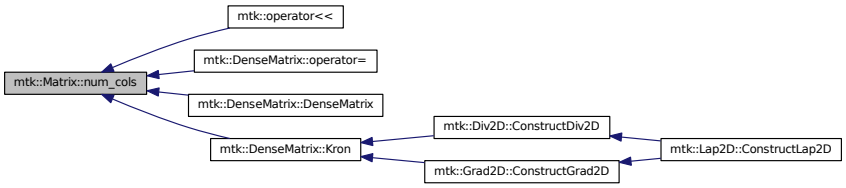
16.15.3.9 int mtk::Matrix::num_cols ( ) const [noexcept]

Returns

Number of rows of the matrix.

Definition at line 122 of file mtk_matrix.cc.

Here is the caller graph for this function:



16.15.3.10 int mtk::Matrix::num_non_null ( ) const [noexcept]

See also

http://www.csrc.sdsu.edu/research_reports/CSRCR2013-01.pdf

Returns

Number of non-null values of the matrix.

Definition at line 152 of file mtk_matrix.cc.

16.15.3.11 int mtk::Matrix::num_non_zero ( ) const [noexcept]

Returns

Number of non-zero values of the matrix.

Definition at line 142 of file mtk_matrix.cc.

16.15.3.12 int mtk::Matrix::num_null ( ) const [noexcept]

See also

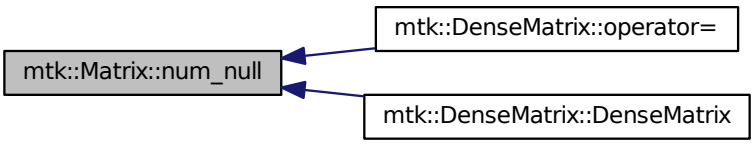
http://www.csrc.sdsu.edu/research_reports/CSRCR2013-01.pdf

Returns

Number of null values of the matrix.

Definition at line 147 of file mtk_matrix.cc.

Here is the caller graph for this function:



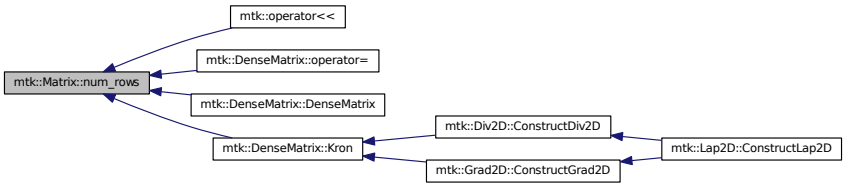
16.15.3.13 int mtk::Matrix::num_rows ( ) const [noexcept]

Returns

Number of rows of the matrix.

Definition at line 117 of file mtk_matrix.cc.

Here is the caller graph for this function:



16.15.3.14 int mtk::Matrix::num_values ( ) const [noexcept]

Returns

Number of values of the matrix.

Definition at line 127 of file mtk_matrix.cc.

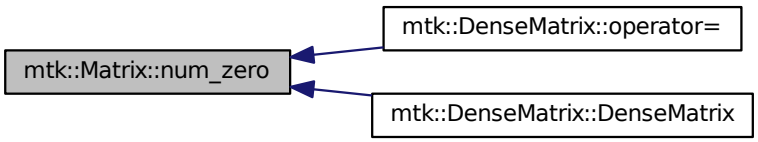
16.15.3.15 int mtk::Matrix::num_zero ( ) const [noexcept]

Returns

Number of zeros of the matrix.

Definition at line 137 of file mtk_matrix.cc.

Here is the caller graph for this function:



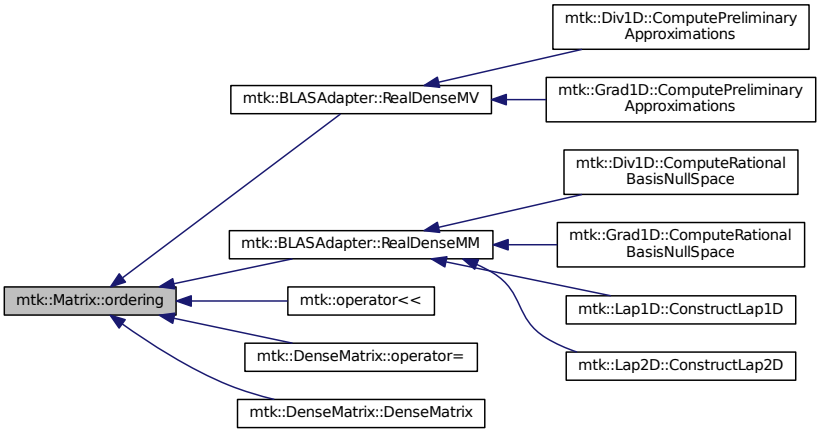
16.15.3.16 mtk::MatrixOrdering mtk::Matrix::ordering ( ) const [noexcept]

Returns

Type of ordering of this matrix.

Definition at line 112 of file mtk_matrix.cc.

Here is the caller graph for this function:



16.15.3.17 mtk::Real mtk::Matrix::rel_density ( ) const [noexcept]

See also

http://www.csrc.sdsu.edu/research_reports/CSRCR2013-01.pdf

Returns

Relative density of the matrix.

Definition at line 172 of file mtk_matrix.cc.

16.15.3.18 mtk::Real mtk::Matrix::rel_sparsity ( ) const [noexcept]

See also

http://www.csrc.sdsu.edu/research_reports/CSRCR2013-01.pdf

Returns

Relative sparsity of the matrix.

Definition at line 182 of file mtk_matrix.cc.

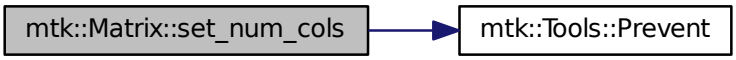
16.15.3.19 void mtk::Matrix::set_num_cols (const int & *num_cols*) [noexcept]

Parameters

in	*num_cols*	Number of columns.

Definition at line 224 of file mtk_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:

16.15.3.20 void mtk::Matrix::set_num_null (const int & *in*) [noexcept]

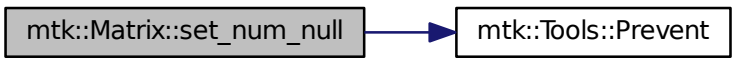
Parameters

in	*in*	Number of zero values.

Bug -nan assigned on construction time due to num_values_ being 0.

Definition at line 250 of file mtk_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:

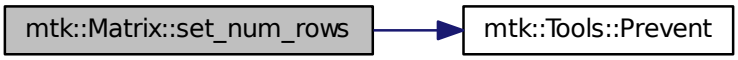
16.15.3.21 void mtk::Matrix::set_num_rows (const int & *num_rows*) [noexcept]

Parameters

in	*num_rows*	Number of rows.

Definition at line 212 of file mtk_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:

16.15.3.22 void mtk::Matrix::set_num_zero (const int & *in*) [noexcept]

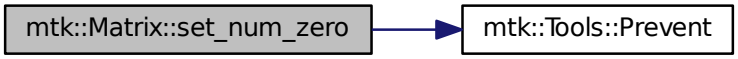
Parameters

in	*in*	Number of zero values.

Bug -nan assigned on construction time due to num_values_ being 0.

Definition at line 236 of file mtk_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:

16.15.3.23 void mtk::Matrix::set_ordering (const MatrixOrdering & *oo*) [noexcept]

See also

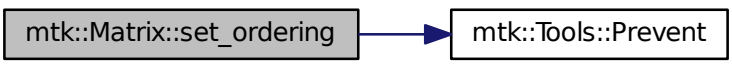
MatrixOrdering

Parameters

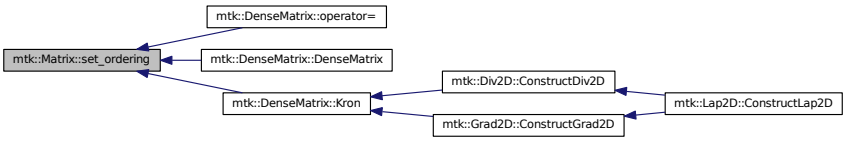
in	*oo*	Ordering of the matrix.

Definition at line 199 of file mtk_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



16.15.3.24 void mtk::Matrix::set_storage (const MatrixStorage & *tt*) [noexcept]

See also

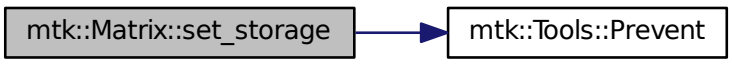
MatrixStorage

Parameters

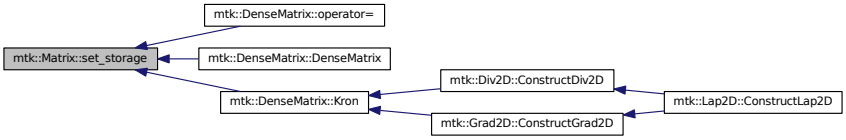
in	*tt*	Type of the matrix storage.

Definition at line 187 of file mtk_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



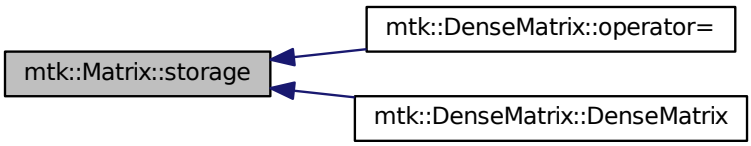
16.15.3.25 mtk::MatrixStorage mtk::Matrix::storage ( ) const [noexcept]

Returns

Type of storage of this matrix.

Definition at line 107 of file mtk_matrix.cc.

Here is the caller graph for this function:



16.15.4 Member Data Documentation

16.15.4.1 Real mtk::Matrix::abs_density_ [private]

Definition at line 296 of file mtk_matrix.h.

16.15.4.2 Real mtk::Matrix::abs_sparsity_ [private]

Definition at line 298 of file mtk_matrix.h.

16.15.4.3 int mtk::Matrix::bandwidth_ [private]

Definition at line 294 of file mtk_matrix.h.

16.15.4.4 int mtk::Matrix::kl_ [private]

Definition at line 292 of file mtk_matrix.h.

16.15.4.5 int mtk::Matrix::ku_ [private]

Definition at line 293 of file mtk_matrix.h.

16.15.4.6 int mtk::Matrix::ld_ [private]

Definition at line 285 of file mtk_matrix.h.

16.15.4.7 int mtk::Matrix::num_cols_ [private]

Definition at line 283 of file mtk_matrix.h.

16.15.4.8 int mtk::Matrix::num_non_null_ [private]

Definition at line 290 of file mtk_matrix.h.

16.15.4.9 int mtk::Matrix::num_non_zero_ [private]

Definition at line 288 of file mtk_matrix.h.

16.15.4.10 int mtk::Matrix::num_null_ [private]

Definition at line 289 of file mtk_matrix.h.

16.15.4.11 int mtk::Matrix::num_rows_ [private]

Definition at line 282 of file mtk_matrix.h.

16.15.4.12 int mtk::Matrix::num_values_ [private]

Definition at line 284 of file mtk_matrix.h.

16.15.4.13 int mtk::Matrix::num_zero_ [private]

Definition at line 287 of file mtk_matrix.h.

16.15.4.14 MatrixOrdering mtk::Matrix::ordering_ [private]

Definition at line 280 of file mtk_matrix.h.

16.15.4.15 Real mtk::Matrix::rel_density_ [private]

Definition at line 297 of file mtk_matrix.h.

16.15.4.16 Real mtk::Matrix::rel_sparsity_ [private]

Definition at line 299 of file mtk_matrix.h.

16.15.4.17 MatrixStorage mtk::Matrix::storage_ [private]

Definition at line 278 of file mtk_matrix.h.

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

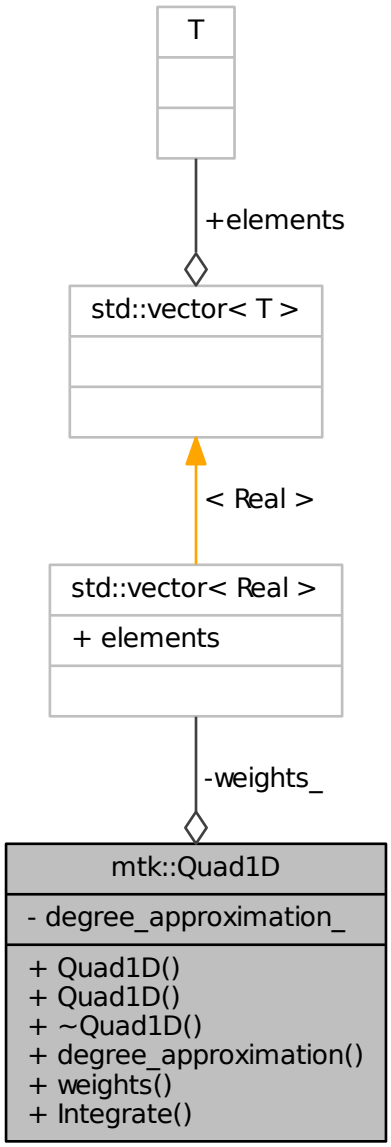
- include/mtk_matrix.h
- src/mtk_matrix.cc

16.16 mtk::Quad1D Class Reference

Implements a 1D mimetic quadrature.

`#include <mtk_quad_1d.h>`

Collaboration diagram for mtk::Quad1D:



Public Member Functions

- Quad1D ()

  Default constructor.
- Quad1D (const Quad1D &quad)

  Copy constructor.
- ~Quad1D ()

  Destructor.
- int degree_approximation () const

  Get the degree of interpolating polynomial per sub-interval of domain.
- Real * weights () const

  Return collection of weights.
- Real Integrate (Real(*Integrand)(Real xx), UniStgGrid1D grid) const

  Mimetic integration routine.

Private Attributes

- int degree_approximation_

  Degree of the interpolating polynomial.
- std::vector< Real > weights_

  Collection of weights.

Friends

- std::ostream & operator<< (std::ostream &stream, Quad1D &in)

  Output stream operator for printing.

16.16.1 Detailed Description

This class implements a 1D quadrature solver based on the mimetic discretization of the gradient operator.

Definition at line 81 of file mtk_quad_1d.h.

16.16.2 Constructor & Destructor Documentation

16.16.2.1 mtk::Quad1D::Quad1D ( )

16.16.2.2 mtk::Quad1D::Quad1D (const Quad1D & *quad*)

Parameters

in	*div*	Given quadrature.

16.16.2.3 mtk::Quad1D::∼Quad1D ( )

16.16.3 Member Function Documentation

16.16.3.1 int mtk::Quad1D::degree_approximation ( ) const

Returns

Degree of the interpolating polynomial per sub-interval of the domain.

16.16.3.2 Real mtk::Quad1D::Integrate (Real(∗)(Real xx) *Integrand,* UniStgGrid1D *grid*) const

Parameters

in	*Integrand*	Real-valued function to integrate.
in	*grid*	Given integration domain.

Returns

Result of the integration.

16.16.3.3 Real∗ mtk::Quad1D::weights ( ) const

Returns

Collection of weights.

16.16.4 Friends And Related Function Documentation

16.16.4.1 std::ostream& operator<< (std::ostream & *stream,* Quad1D & *in*) [friend]

16.16.5 Member Data Documentation

16.16.5.1 int mtk::Quad1D::degree_approximation_ [private]

Definition at line 124 of file mtk_quad_1d.h.

16.16.5.2 std::vector<Real> mtk::Quad1D::weights_ [private]

Definition at line 126 of file mtk_quad_1d.h.

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

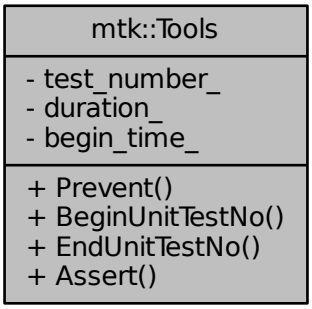
- include/mtk_quad_1d.h

16.17 mtk::Tools Class Reference

Tool manager class.

`#include <mtk_tools.h>`

Collaboration diagram for mtk::Tools:



Static Public Member Functions

- static void Prevent (const bool complement, const char *const fname, int lineno, const char *const fxname) noexcept

  Enforces preconditions by preventing their complements from occur.
- static void BeginUnitTestNo (const int &nn) noexcept

  Begins the execution of a unit test. Starts a timer.
- static void EndUnitTestNo (const int &nn) noexcept

  Ends the execution of a unit test. Stops and reports wall-clock time.
- static void Assert (const bool &condition) noexcept

  Asserts if the condition required to pass the unit test occurs.

Static Private Attributes

- static int test_number_

  Current test being executed.
- static Real duration_

  Duration of the current test.
- static clock_t begin_time_

  Elapsed time on current test.

16.17.1 Detailed Description

Basic tools to ensure execution correctness.

Definition at line 78 of file mtk_tools.h.

16.17.2 Member Function Documentation

16.17.2.1 void mtk::Tools::Assert (const bool & *condition*) [static], [noexcept]

Parameters

in	*condition*	Condition to be asserted.

Definition at line 114 of file mtk_tools.cc.

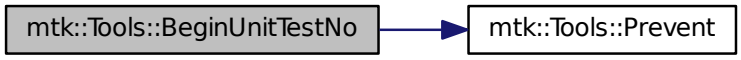
16.17.2.2 void mtk::Tools::BeginUnitTestNo (const int & *nn*) [static], [noexcept]

Parameters

in	*nn*	Number of the test.

Definition at line 91 of file mtk_tools.cc.

Here is the call graph for this function:



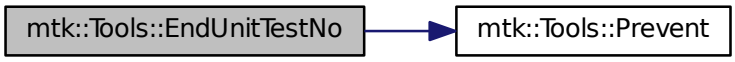
16.17.2.3 void mtk::Tools::EndUnitTestNo (const int & *nn*) [static], [noexcept]

Parameters

in	*nn*	Number of the test.

Definition at line 105 of file mtk_tools.cc.

Here is the call graph for this function:



16.17.2.4 void mtk::Tools::Prevent (const bool *complement*, const char *const *fname*, int *lineno*, const char *const *fxname*) [static], [noexcept]

See also

http://stackoverflow.com/questions/8884335/print-the-file-name-line-number-and-function

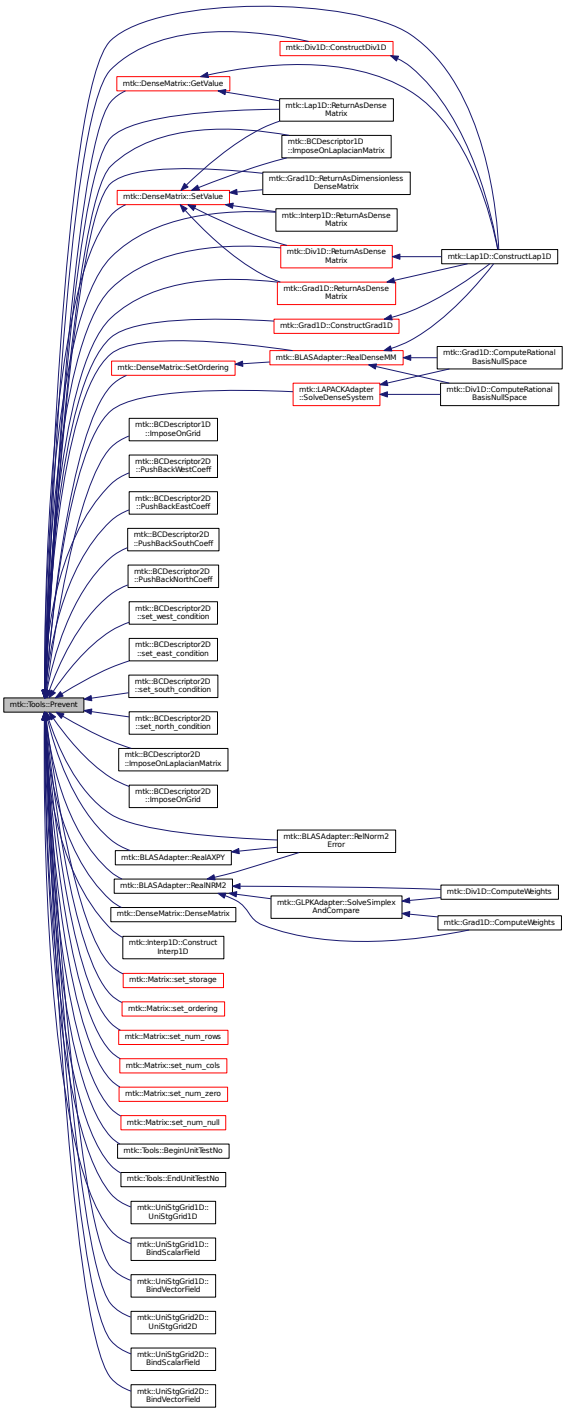
Parameters

in	*complement*	Complement of desired pre-condition.
in	*fname*	Name of the file being checked.
in	*lineno*	Number of the line where the check is executed.
in	*fxname*	Name of the module containing the check.

Todo Check if this is the best way of stalling execution.

Definition at line 61 of file mtk_tools.cc.

Here is the caller graph for this function:



16.17.3 Member Data Documentation

16.17.3.1 clock_t mtk::Tools::begin_time_ [static], [private]

Definition at line 121 of file mtk_tools.h.

16.17.3.2 mtk::Real mtk::Tools::duration_ [static], [private]

Definition at line 119 of file mtk_tools.h.

16.17.3.3 int mtk::Tools::test_number_ [static], [private]

Todo Check usage of static methods and private members.

Definition at line 117 of file mtk_tools.h.

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

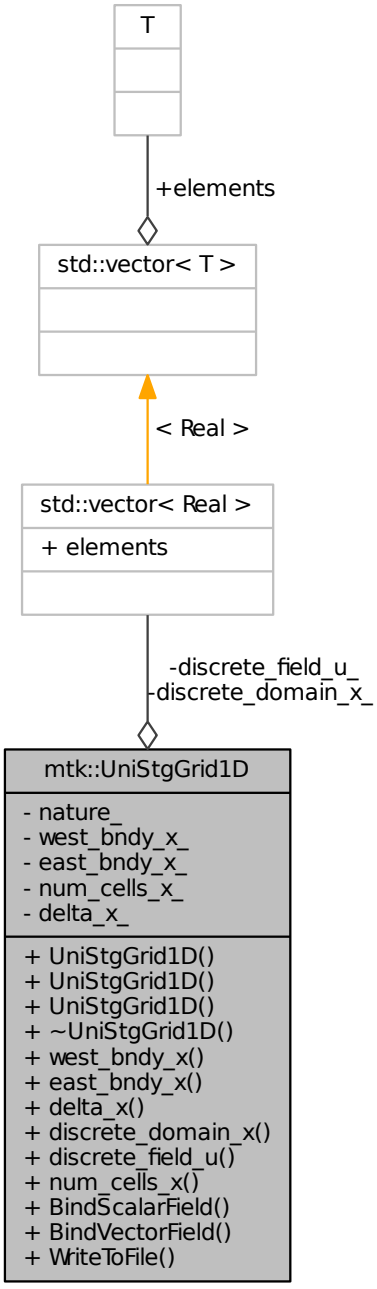
- include/mtk_tools.h
- src/mtk_tools.cc

16.18 mtk::UniStgGrid1D Class Reference

Uniform 1D Staggered Grid.

`#include <mtk_uni_stg_grid_1d.h>`

Collaboration diagram for mtk::UniStgGrid1D:



Public Member Functions

- UniStgGrid1D ()

Default constructor.

- UniStgGrid1D (const UniStgGrid1D &grid)

  Copy constructor.

- UniStgGrid1D (const Real &west_bndy_x, const Real &east_bndy_x, const int &num_cells_x, const mtk::FieldNature &nature=mtk::SCALAR)

  Construct a grid based on spatial discretization parameters.

- ~UniStgGrid1D ()

  Destructor.

- Real west_bndy_x () const

  Provides access to west boundary spatial coordinate.

- Real east_bndy_x () const

  Provides access to east boundary spatial coordinate.

- Real delta_x () const

  Provides access to the computed $ x $.

- const Real * discrete_domain_x () const

  Provides access to the grid spatial data.

- Real * discrete_field_u ()

  Provides access to the grid field data.

- int num_cells_x () const

  Provides access to the number of cells of the grid.

- void BindScalarField (Real(*ScalarField)(Real xx))

  Binds a given scalar field to the grid.

- void BindVectorField (Real(*VectorField)(Real xx))

  Binds a given vector field to the grid.

- bool WriteToFile (std::string filename, std::string space_name, std::string field_name) const

  Writes grid to a file compatible with gnuplot 4.6.

Private Attributes

- FieldNature nature_

  Nature of the discrete field.

- std::vector< Real > discrete_domain_x_

  Array of spatial data.

- std::vector< Real > discrete_field_u_

  Array of field's data.

- Real west_bndy_x_

  West boundary spatial coordinate.

- Real east_bndy_x_

  East boundary spatial coordinate.

- Real num_cells_x_

  Number of cells discretizing the domain.

- Real delta_x_

  Produced Δx.

Friends

- std::ostream & operator<< (std::ostream &stream, UniStgGrid1D &in)

  Prints the grid as a tuple of arrays.

16.18.1 Detailed Description

Uniform 1D Staggered Grid.

Definition at line 77 of file mtk_uni_stg_grid_1d.h.

16.18.2 Constructor & Destructor Documentation

16.18.2.1 mtk::UniStgGrid1D::UniStgGrid1D ( )

Definition at line 99 of file mtk_uni_stg_grid_1d.cc.

16.18.2.2 mtk::UniStgGrid1D::UniStgGrid1D (const UniStgGrid1D & *grid*)

Parameters

in	*grid*	Given grid.

Definition at line 108 of file mtk_uni_stg_grid_1d.cc.

16.18.2.3 mtk::UniStgGrid1D::UniStgGrid1D (const Real & *west_bndy_x*, const Real & *east_bndy_x*, const int & *num_cells_x*, const mtk::FieldNature & *nature* = mtk::SCALAR)

Parameters

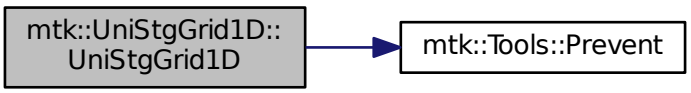
in	*west_bndy_x*	Coordinate for the west boundary.
in	*east_bndy_x*	Coordinate for the east boundary.
in	*num_cells_x*	Number of cells of the required grid.
in	*nature*	Nature of the discrete field to hold.

See also

mtk::FieldNature

Definition at line 124 of file mtk_uni_stg_grid_1d.cc.

Here is the call graph for this function:



16.18.2.4 mtk::UniStgGrid1D::~UniStgGrid1D ( )

Definition at line 144 of file mtk_uni_stg_grid_1d.cc.

16.18.3 Member Function Documentation

16.18.3.1 void mtk::UniStgGrid1D::BindScalarField (Real(∗)(Real xx) *ScalarField*)

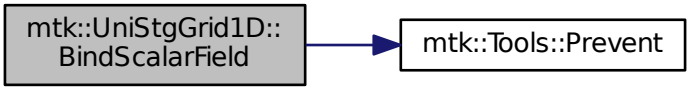
Parameters

in	*ScalarField*	Pointer to the function implementing the scalar field.

1. Create collection of spatial coordinates.
2. Create collection of field samples.

Definition at line 176 of file mtk_uni_stg_grid_1d.cc.

Here is the call graph for this function:



16.18.3.2 void mtk::UniStgGrid1D::BindVectorField (Real(∗)(Real xx) *VectorField*)

We assume the field to be of the form:

$$\mathbf{v}(x) = v(x)\hat{\mathbf{i}}$$

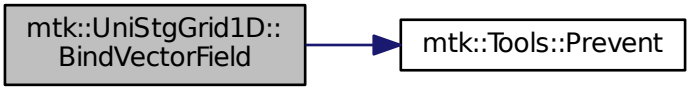
Parameters

in	*VectorField*	Pointer to the function implementing the vector field.

1. Create collection of spatial coordinates.
2. Create collection of field samples.

Definition at line 212 of file mtk_uni_stg_grid_1d.cc.

Here is the call graph for this function:



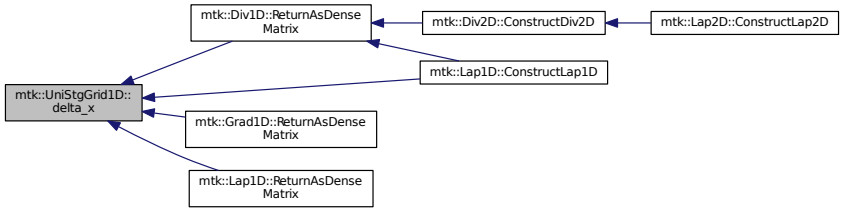
16.18.3.3 mtk::Real mtk::UniStgGrid1D::delta_x ( ) const

Returns

Computed $ x $.

Definition at line 156 of file mtk_uni_stg_grid_1d.cc.

Here is the caller graph for this function:



16.18.3.4 const mtk::Real * mtk::UniStgGrid1D::discrete_domain_x ( ) const

Returns

Pointer to the spatial data.

Todo Review const-correctness of the pointer we return.

Definition at line 161 of file mtk_uni_stg_grid_1d.cc.

16.18.3.5 mtk::Real * mtk::UniStgGrid1D::discrete_field_u ( )

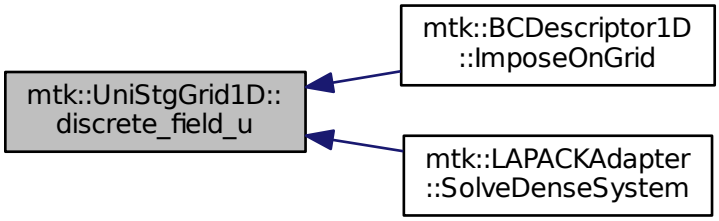
Returns

Pointer to the field data.

Todo Review const-correctness of the pointer we return. Look at the STL!

Definition at line 166 of file mtk_uni_stg_grid_1d.cc.

Here is the caller graph for this function:



16.18.3.6 mtk::Real mtk::UniStgGrid1D::east_bndy_x ( ) const

Returns

East boundary spatial coordinate.

Definition at line 151 of file mtk_uni_stg_grid_1d.cc.

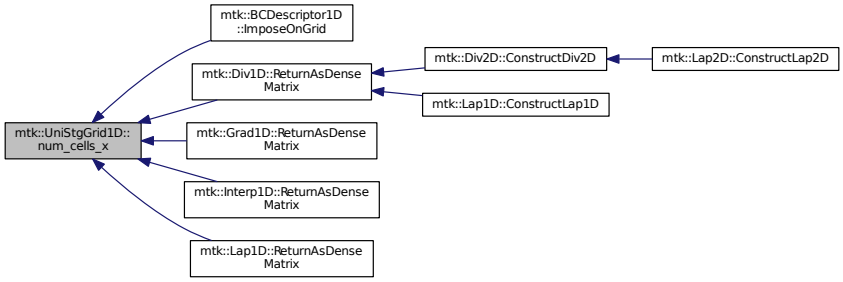
16.18.3.7 int mtk::UniStgGrid1D::num_cells_x ( ) const

Returns

Number of cells of the grid.

Definition at line 171 of file mtk_uni_stg_grid_1d.cc.

Here is the caller graph for this function:



16.18.3.8 mtk::Real mtk::UniStgGrid1D::west_bndy_x ( ) const

Returns

West boundary spatial coordinate.

Definition at line 146 of file mtk_uni_stg_grid_1d.cc.

16.18.3.9 bool mtk::UniStgGrid1D::WriteToFile (std::string *filename*, std::string *space_name*, std::string *field_name*) const

Parameters

in	*filename*	Name of the output file.
in	*space_name*	Name for the first column of the data.
in	*field_name*	Name for the second column of the data.

Returns

Success of the file writing process.

See also

http://www.gnuplot.info/

Definition at line 240 of file mtk_uni_stg_grid_1d.cc.

16.18.4 Friends And Related Function Documentation

16.18.4.1 std::ostream& operator<< (std::ostream & *stream*, mtk::UniStgGrid1D & *in*) [friend]

1. Print spatial coordinates.
2. Print scalar field.

Definition at line 68 of file mtk_uni_stg_grid_1d.cc.

16.18.5 Member Data Documentation

16.18.5.1 Real mtk::UniStgGrid1D::delta_x_ [private]

Definition at line 200 of file mtk_uni_stg_grid_1d.h.

16.18.5.2 std::vector<Real> mtk::UniStgGrid1D::discrete_domain_x_ [private]

Definition at line 194 of file mtk_uni_stg_grid_1d.h.

16.18.5.3 std::vector<Real> mtk::UniStgGrid1D::discrete_field_u_ [private]

Definition at line 195 of file mtk_uni_stg_grid_1d.h.

16.18.5.4 Real mtk::UniStgGrid1D::east_bndy_x_ [private]

Definition at line 198 of file mtk_uni_stg_grid_1d.h.

16.18.5.5 FieldNature mtk::UniStgGrid1D::nature_ [private]

Definition at line 192 of file mtk_uni_stg_grid_1d.h.

16.18.5.6 Real mtk::UniStgGrid1D::num_cells_x_ [private]

Definition at line 199 of file mtk_uni_stg_grid_1d.h.

16.18.5.7 Real mtk::UniStgGrid1D::west_bndy_x_ [private]

Definition at line 197 of file mtk_uni_stg_grid_1d.h.

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

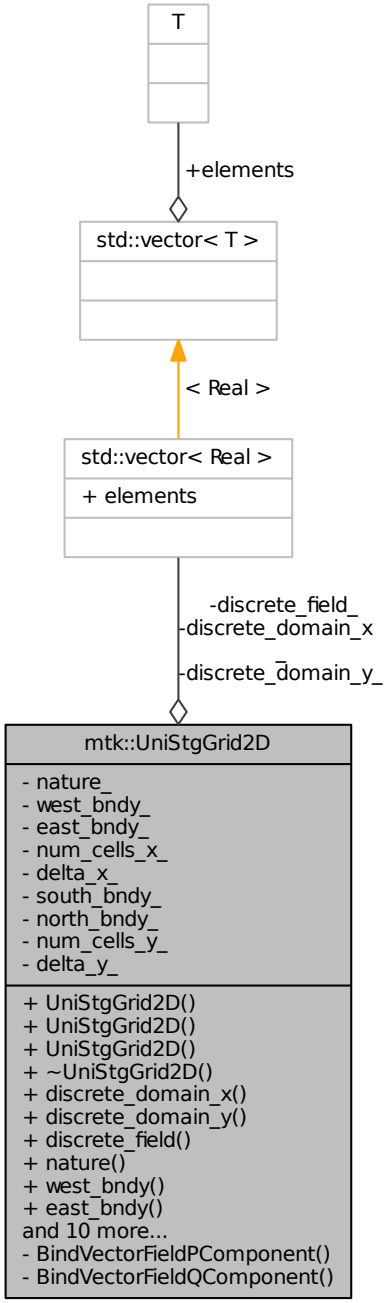
- include/mtk_uni_stg_grid_1d.h
- src/mtk_uni_stg_grid_1d.cc

16.19 mtk::UniStgGrid2D Class Reference

Uniform 2D Staggered Grid.

`#include <mtk_uni_stg_grid_2d.h>`

Collaboration diagram for mtk::UniStgGrid2D:



Public Member Functions

- UniStgGrid2D ()

Default constructor.

- UniStgGrid2D (const UniStgGrid2D &grid)

  Copy constructor.

- UniStgGrid2D (const Real &west_bndy_x, const Real &east_bndy_x, const int &num_cells_x, const Real &south_bndy_y, const Real &north_bndy_y, const int &num_cells_y, const mtk::FieldNature &nature=mtk::SCALAR)

  Construct a grid based on spatial discretization parameters.

- ~UniStgGrid2D ()

  Destructor.

- const Real * discrete_domain_x () const

  Provides access to the grid spatial data.

- const Real * discrete_domain_y () const

  Provides access to the grid spatial data.

- Real * discrete_field ()

  Provides access to the grid field data.

- FieldNature nature () const

  Physical nature of the data bound to the grid.

- Real west_bndy () const

  Provides access to west boundary spatial coordinate.

- Real east_bndy () const

  Provides access to east boundary spatial coordinate.

- int num_cells_x () const

  Provides access to the number of cells of the grid.

- Real delta_x () const

  Provides access to the computed $ x $.

- Real south_bndy () const

  Provides access to south boundary spatial coordinate.

- Real north_bndy () const

  Provides access to north boundary spatial coordinate.

- int num_cells_y () const

  Provides access to the number of cells of the grid.

- Real delta_y () const

  Provides access to the computed $ y $.

- bool Bound () const

  Have any field been bound to the grid?

- void BindScalarField (Real(*ScalarField)(Real xx, Real yy))

  Binds a given scalar field to the grid.

- void BindVectorField (Real(*VectorFieldPComponent)(Real xx, Real yy), Real(*VectorFieldQComponent)(Real xx, Real yy))

  Binds a given vector field to the grid.

- bool WriteToFile (std::string filename, std::string space_name_x, std::string space_name_y, std::string field_name) const

  Writes grid to a file compatible with Gnuplot 4.6.

Private Member Functions

- void BindVectorFieldPComponent (Real(*VectorFieldPComponent)(Real xx, Real yy))

  Binds a given component of a vector field to the grid.
- void BindVectorFieldQComponent (Real(*VectorFieldQComponent)(Real xx, Real yy))

  Binds a given component of a vector field to the grid.

Private Attributes

- std::vector< Real > discrete_domain_x_

  Array of spatial data.
- std::vector< Real > discrete_domain_y_

  Array of spatial data.
- std::vector< Real > discrete_field_

  Array of field's data.
- FieldNature nature_

  Nature of the discrete field.
- Real west_bndy_

  West boundary spatial coordinate.
- Real east_bndy_

  East boundary spatial coordinate.
- int num_cells_x_

  Number of cells discretizing the domain.
- Real delta_x_

  Computed Δx.
- Real south_bndy_

  West boundary spatial coordinate.
- Real north_bndy_

  East boundary spatial coordinate.
- int num_cells_y_

  Number of cells discretizing the domain.
- Real delta_y_

  Computed Δy.

Friends

- std::ostream & operator<< (std::ostream &stream, UniStgGrid2D &in)

  Prints the grid as a tuple of arrays.

16.19.1 Detailed Description

Uniform 2D Staggered Grid.

Definition at line 79 of file mtk_uni_stg_grid_2d.h.

16.19.2 Constructor & Destructor Documentation

16.19.2.1 mtk::UniStgGrid2D::UniStgGrid2D ( )

Definition at line 131 of file mtk_uni_stg_grid_2d.cc.

16.19.2.2 mtk::UniStgGrid2D::UniStgGrid2D (const UniStgGrid2D & *grid*)

Parameters

in	*grid*	Given grid.

Definition at line 145 of file mtk_uni_stg_grid_2d.cc.

16.19.2.3 mtk::UniStgGrid2D::UniStgGrid2D (const Real & *west_bndy_x*, const Real & *east_bndy_x*, const int & *num_cells_x*, const Real & *south_bndy_y*, const Real & *north_bndy_y*, const int & *num_cells_y*, const mtk::FieldNature & *nature* = mtk::SCALAR)

Parameters

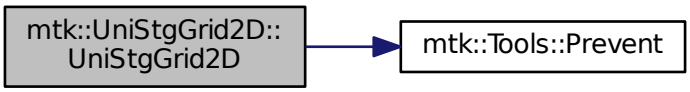
in	*west_bndy_x*	Coordinate for the west boundary.
in	*east_bndy_x*	Coordinate for the east boundary.
in	*num_cells_x*	Number of cells of the required grid.
in	*south_bndy_y*	Coordinate for the west boundary.
in	*north_bndy_y*	Coordinate for the east boundary.
in	*num_cells_y*	Number of cells of the required grid.
in	*nature*	Nature of the discrete field to hold.

See also

mtk::FieldNature

Definition at line 169 of file mtk_uni_stg_grid_2d.cc.

Here is the call graph for this function:



16.19.2.4 mtk::UniStgGrid2D::~UniStgGrid2D ( )

Definition at line 203 of file mtk_uni_stg_grid_2d.cc.

16.19.3 Member Function Documentation

16.19.3.1 void mtk::UniStgGrid2D::BindScalarField (Real(∗)(Real xx, Real yy) *ScalarField*)

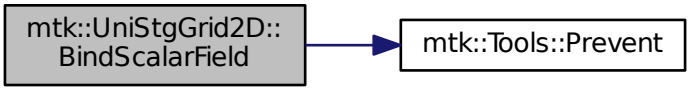
Parameters

in	*ScalarField*	Pointer to the function implementing the scalar field.

1. Create collection of spatial coordinates for x.
2. Create collection of spatial coordinates for y.
3. Create collection of field samples.

Definition at line 270 of file mtk_uni_stg_grid_2d.cc.

Here is the call graph for this function:



16.19.3.2 void mtk::UniStgGrid2D::BindVectorField (Real(∗)(Real xx, Real yy) *VectorFieldPComponent,* Real(∗)(Real xx, Real yy) *VectorFieldQComponent*)

We assume the field to be of the form:

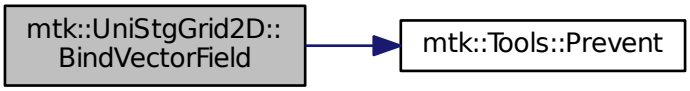
$$\mathbf{v}(x) = p(x,y)\hat{\mathbf{i}} + q(x,y)\hat{\mathbf{j}}$$

Parameters

in	*VectorFieldP↩ Component*	Pointer to the function implementing the $ p $ component of the vector field.
in	*VectorFieldP↩ Component*	Pointer to the function implementing the $ q $ component of the vector field.

Definition at line 417 of file mtk_uni_stg_grid_2d.cc.

Here is the call graph for this function:



16.19.3.3 void mtk::UniStgGrid2D::BindVectorFieldPComponent (Real(∗)(Real xx, Real yy) *VectorFieldPComponent*) [private]

We assume the field to be of the form:

$$\mathbf{v}(x) = p(x,y)\hat{\mathbf{i}} + q(x,y)\hat{\mathbf{j}}$$

Parameters

in	*BindVectorField↩ PComponent*	Pointer to the function implementing the $ p $ component of the vector field.

1. Create collection of spatial coordinates for x.
2. Create collection of spatial coordinates for y.
3. Allocate space for discrete vector field and bind $ p $ component.

Definition at line 324 of file mtk_uni_stg_grid_2d.cc.

16.19.3.4 void mtk::UniStgGrid2D::BindVectorFieldQComponent (Real(∗)(Real xx, Real yy) *VectorFieldQComponent*) [private]

We assume the field to be of the form:

$$\mathbf{v}(x) = p(x,y)\hat{\mathbf{i}} + q(x,y)\hat{\mathbf{j}}$$

Parameters

in	*BindVectorField↩ QComponent*	Pointer to the function implementing the $ q $ component of the vector field.

1. Bind $ q $ component, since $ p $ component has already been bound.

Definition at line 389 of file mtk_uni_stg_grid_2d.cc.

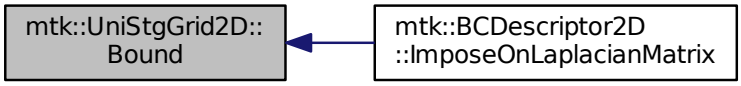
16.19.3.5 bool mtk::UniStgGrid2D::Bound ( ) const

Returns

True is a field has been bound.

Definition at line 255 of file mtk_uni_stg_grid_2d.cc.

Here is the caller graph for this function:



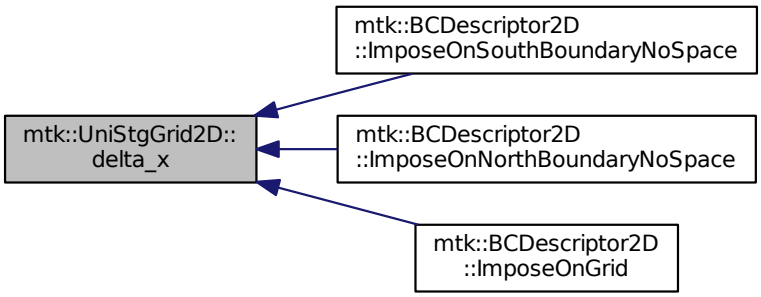
16.19.3.6 mtk::Real mtk::UniStgGrid2D::delta_x ( ) const

Returns

Computed $ x $.

Definition at line 225 of file mtk_uni_stg_grid_2d.cc.

Here is the caller graph for this function:



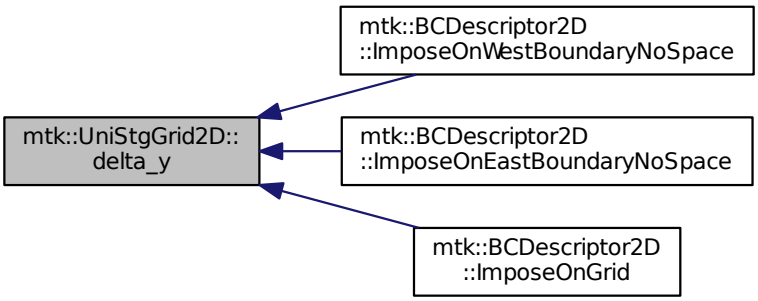
16.19.3.7 mtk::Real mtk::UniStgGrid2D::delta_y ( ) const

Returns

Computed $ y $.

Definition at line 250 of file mtk_uni_stg_grid_2d.cc.

Here is the caller graph for this function:



16.19.3.8 const mtk::Real * mtk::UniStgGrid2D::discrete_domain_x ( ) const

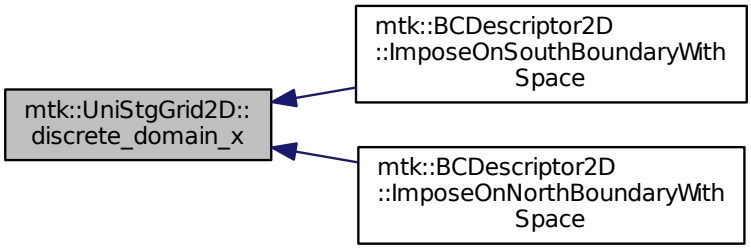
Returns

Pointer to the spatial data.

Todo Review const-correctness of the pointer we return.

Definition at line 230 of file mtk_uni_stg_grid_2d.cc.

Here is the caller graph for this function:



16.19.3.9 const mtk::Real * mtk::UniStgGrid2D::discrete_domain_y ( ) const

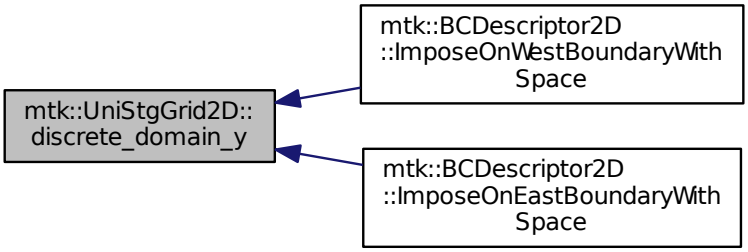
Returns

Pointer to the spatial data.

Todo Review const-correctness of the pointer we return.

Definition at line 260 of file mtk_uni_stg_grid_2d.cc.

Here is the caller graph for this function:



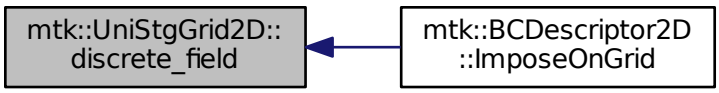
16.19.3.10 mtk::Real * mtk::UniStgGrid2D::discrete_field ( )

Returns

Pointer to the field data.

Definition at line 265 of file mtk_uni_stg_grid_2d.cc.

Here is the caller graph for this function:



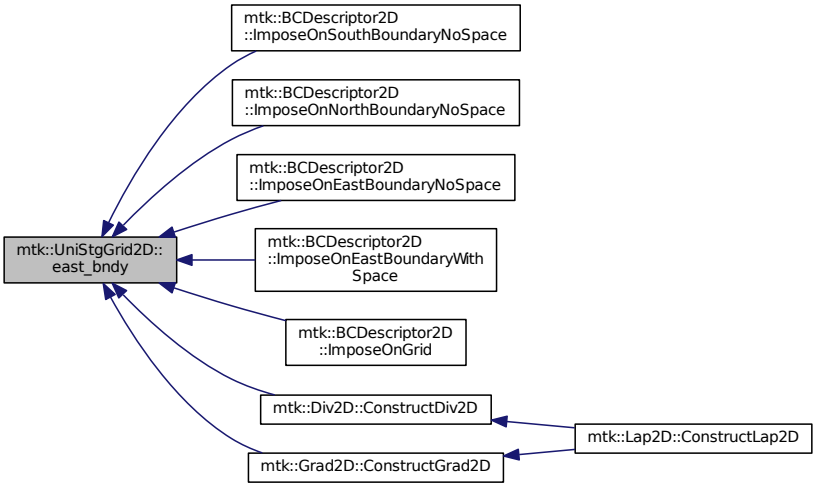
16.19.3.11 mtk::Real mtk::UniStgGrid2D::east_bndy ( ) const

Returns

East boundary spatial coordinate.

Definition at line 215 of file mtk_uni_stg_grid_2d.cc.

Here is the caller graph for this function:



16.19.3.12 mtk::FieldNature mtk::UniStgGrid2D::nature ( ) const

Returns

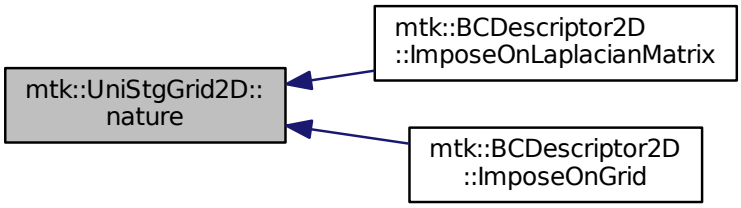
Value of an enumeration.

See also

mtk::FieldNature

Definition at line 205 of file mtk_uni_stg_grid_2d.cc.

Here is the caller graph for this function:



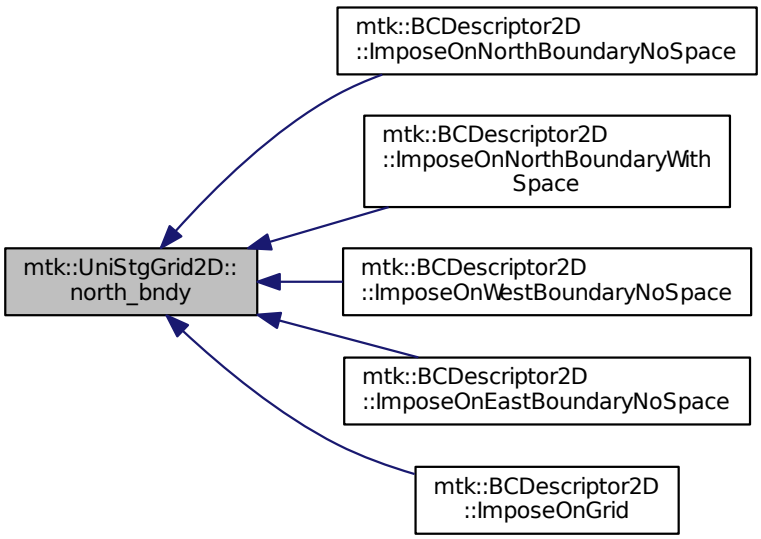
16.19.3.13 mtk::Real mtk::UniStgGrid2D::north_bndy ( ) const

Returns

North boundary spatial coordinate.

Definition at line 240 of file mtk_uni_stg_grid_2d.cc.

Here is the caller graph for this function:



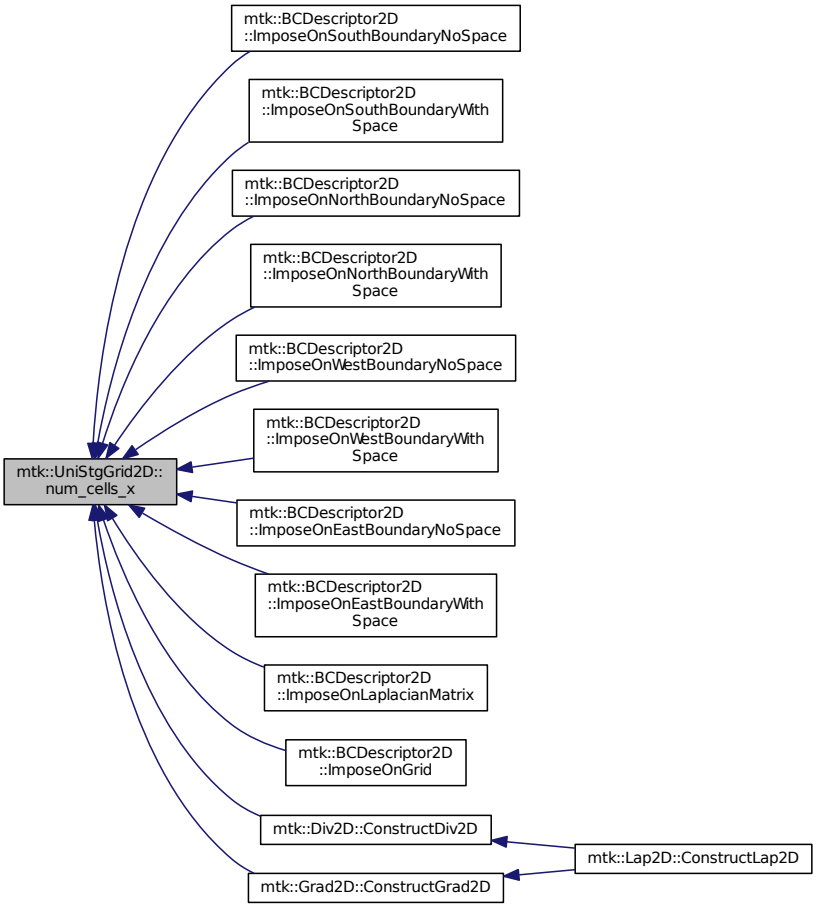
16.19.3.14 int mtk::UniStgGrid2D::num_cells_x ( ) const

Returns

Number of cells of the grid.

Definition at line 220 of file mtk_uni_stg_grid_2d.cc.

Here is the caller graph for this function:



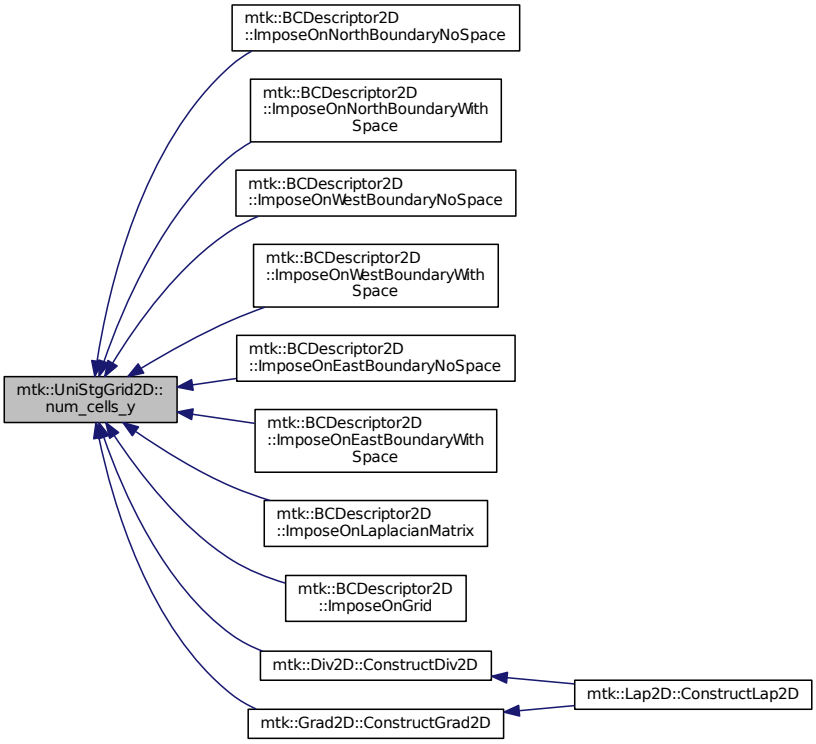
16.19.3.15 int mtk::UniStgGrid2D::num_cells_y ( ) const

Returns

Number of cells of the grid.

Definition at line 245 of file mtk_uni_stg_grid_2d.cc.

Here is the caller graph for this function:



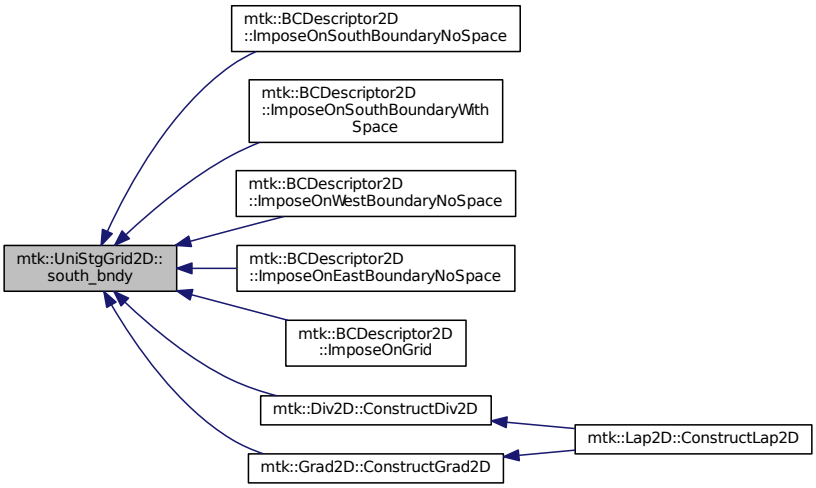
16.19.3.16 mtk::Real mtk::UniStgGrid2D::south_bndy ( ) const

Returns

South boundary spatial coordinate.

Definition at line 235 of file mtk_uni_stg_grid_2d.cc.

Here is the caller graph for this function:



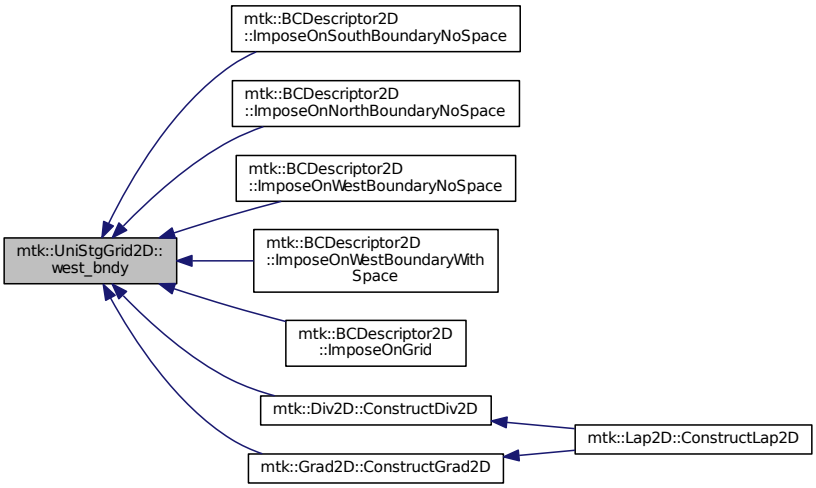
16.19.3.17 mtk::Real mtk::UniStgGrid2D::west_bndy ( ) const

Returns

West boundary spatial coordinate.

Definition at line 210 of file mtk_uni_stg_grid_2d.cc.

Here is the caller graph for this function:



16.19.3.18 bool mtk::UniStgGrid2D::WriteToFile (std::string *filename*, std::string *space_name_x*, std::string *space_name_y*, std::string *field_name*) const

Parameters

in	*filename*	Name of the output file.
in	*space_name_x*	Name for the first column of the (spatial) data.
in	*space_name_y*	Name for the second column of the (spatial) data.
in	*field_name*	Name for the second column of the (physical field) data.

Returns

Success of the file writing process.

See also

http://www.gnuplot.info/

Write the values of the p component, with a null q component.

Write the values of the q component, with a null p component.

Definition at line 429 of file mtk_uni_stg_grid_2d.cc.

16.19.4 Friends And Related Function Documentation

16.19.4.1 std::ostream& operator<< (std::ostream & *stream*, mtk::UniStgGrid2D & *in*) [friend]

1. Print spatial coordinates.

2. Print scalar field.

Definition at line 67 of file mtk_uni_stg_grid_2d.cc.

16.19.5 Member Data Documentation

16.19.5.1 Real mtk::UniStgGrid2D::delta_x_ [private]

Definition at line 296 of file mtk_uni_stg_grid_2d.h.

16.19.5.2 Real mtk::UniStgGrid2D::delta_y_ [private]

Definition at line 301 of file mtk_uni_stg_grid_2d.h.

16.19.5.3 std::vector<Real> mtk::UniStgGrid2D::discrete_domain_x_ [private]

Definition at line 287 of file mtk_uni_stg_grid_2d.h.

16.19.5.4 std::vector<Real> mtk::UniStgGrid2D::discrete_domain_y_ [private]

Definition at line 288 of file mtk_uni_stg_grid_2d.h.

16.19.5.5 std::vector<Real> mtk::UniStgGrid2D::discrete_field_ [private]

Definition at line 289 of file mtk_uni_stg_grid_2d.h.

16.19.5.6 Real mtk::UniStgGrid2D::east_bndy_ [private]

Definition at line 294 of file mtk_uni_stg_grid_2d.h.

16.19.5.7 FieldNature mtk::UniStgGrid2D::nature_ [private]

Definition at line 291 of file mtk_uni_stg_grid_2d.h.

16.19.5.8 Real mtk::UniStgGrid2D::north_bndy_ [private]

Definition at line 299 of file mtk_uni_stg_grid_2d.h.

16.19.5.9 int mtk::UniStgGrid2D::num_cells_x_ [private]

Definition at line 295 of file mtk_uni_stg_grid_2d.h.

16.19.5.10 int mtk::UniStgGrid2D::num_cells_y_ [private]

Definition at line 300 of file mtk_uni_stg_grid_2d.h.

16.19.5.11 Real mtk::UniStgGrid2D::south_bndy_ [private]

Definition at line 298 of file mtk_uni_stg_grid_2d.h.

16.19.5.12 Real mtk::UniStgGrid2D::west_bndy_ [private]

Definition at line 293 of file mtk_uni_stg_grid_2d.h.

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

- include/mtk_uni_stg_grid_2d.h
- src/mtk_uni_stg_grid_2d.cc

Chapter 17

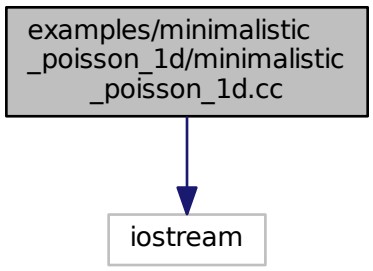
File Documentation

17.1 examples/minimalistic_poisson_1d/minimalistic_poisson_1d.cc File Reference

Poisson Equation on a 1D Uniform Staggered Grid with Robin BCs.

```
#include <iostream>
```

Include dependency graph for minimalistic_poisson_1d.cc:



Functions

- int main ()

17.1.1 Detailed Description

We solve:

$$\nabla^2 p(x) = -s(x),$$

for $x \in \Omega = [a,b] = [0,1]$.

The source term function is defined as

$$s(x) = \frac{\lambda^2 \exp(\lambda x)}{\exp(\lambda) - 1}$$

where $\lambda = -1$ is a parameter.

We consider Robin's boundary conditions of the form:

$$\alpha p(a) - \beta p'(a) = \omega,$$

$$\alpha p(b) + \beta p'(b) = \varepsilon.$$

The analytical solution for this problem is given by

$$p(x) = \frac{\exp(\lambda x) - 1}{\exp(\lambda) - 1}.$$

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu
: Raul Vargas–Navarro - vargasna at rohan dot sdsu dot edu

Definition in file minimalistic_poisson_1d.cc.

17.1.2 Function Documentation

17.1.2.1 int main ( )

Definition at line 167 of file minimalistic_poisson_1d.cc.

17.2 minimalistic_poisson_1d.cc

```
00001
00042 /*
00043 Copyright (C) 2015, Computational Science Research Center, San Diego State
00044 University. All rights reserved.
00045
00046 Redistribution and use in source and binary forms, with or without modification,
00047 are permitted provided that the following conditions are met:
00048
00049 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00050 and a copy of the modified files should be reported once modifications are
00051 completed, unless these modifications are made through the project's GitHub
00052 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00053 should be developed and included in any deliverable.
00054
00055 2. Redistributions of source code must be done through direct
00056 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00057
00058 3. Redistributions in binary form must reproduce the above copyright notice,
00059 this list of conditions and the following disclaimer in the documentation and/or
00060 other materials provided with the distribution.
00061
00062 4. Usage of the binary form on proprietary applications shall require explicit
00063 prior written permission from the the copyright holders, and due credit should
00064 be given to the copyright holders.
```

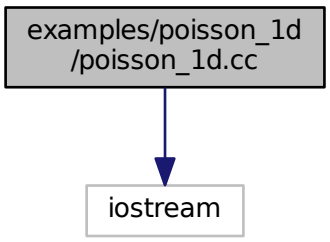
```
00065
00066 5. Neither the name of the copyright holder nor the names of its contributors
00067 may be used to endorse or promote products derived from this software without
00068 specific prior written permission.
00069
00070 The copyright holders provide no reassurances that the source code provided does
00071 not infringe any patent, copyright, or any other intellectual property rights of
00072 third parties. The copyright holders disclaim any liability to any recipient for
00073 claims brought against recipient by any third party for infringement of that
00074 parties intellectual property rights.
00075
00076 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00077 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00078 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00079 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00080 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00081 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00082 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00083 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00084 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00085 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00086 */
00087
00088 #if __cplusplus == 201103L
00089
00090 #include <iostream>
00091 #include <fstream>
00092 #include <cmath>
00093 #include <vector>
00094
00095 #include "mtk.h"
00096
00097 mtk::Real Source(mtk::Real xx) {
00098   mtk::Real lambda = -1.0;
00099   return lambda*lambda*exp(lambda*xx)/(exp(lambda) - 1.0);
00100 }
00101
00102 mtk::Real KnownSolution(mtk::Real xx) {
00103   mtk::Real lambda = -1.0;
00104   return (exp(lambda*xx) - 1.0)/(exp(lambda) - 1.0);
00105 }
00106
00107 int main () {
00108
00109   mtk::Real west_bndy_x = 0.0;
00110   mtk::Real east_bndy_x = 1.0;
00111   mtk::Real relative_norm_2_error{};
00112   int num_cells_x = 5;
00113   mtk::Grad1D grad;
00114   mtk::Lap1D lap;
00115   std::vector<mtk::Real> west_coeffs;
00116   std::vector<mtk::Real> east_coeffs;
00117   mtk::UniStgGrid1D source(west_bndy_x, east_bndy_x, num_cells_x);
00118   mtk::UniStgGrid1D comp_sol(west_bndy_x, east_bndy_x, num_cells_x);
00119   mtk::UniStgGrid1D known_sol(west_bndy_x, east_bndy_x, num_cells_x);
00120
00121   if (!lap.ConstructLap1D()) {
00122     std::cerr << "Mimetic lap could not be built." << std::endl;
00123     return EXIT_FAILURE;
00124   }
00125   mtk::DenseMatrix lapm(lap.ReturnAsDenseMatrix(comp_sol));
00126   if (!grad.ConstructGrad1D()) {
00127     std::cerr << "Mimetic grad could not be built." << std::endl;
00128     return EXIT_FAILURE;
00129   }
00130   mtk::DenseMatrix gradm(grad.ReturnAsDenseMatrix(comp_sol));
00131
00132   source.BindScalarField(Source);
00133
00134   for (auto ii = 0; ii < grad.num_bndy_coeffs(); ++ii) {
00135     west_coeffs.push_back(-((exp(-1.0) - 1.0)/-1.0)*gradm.GetValue(0, ii));
00136   }
00137   for (auto ii = 0; ii < grad.num_bndy_coeffs(); ++ii) {
00138     east_coeffs.push_back(
00139       ((exp(-1.0) - 1.0)/-1.0)*gradm.GetValue(gradm.num_rows() - 1,
00140                                               gradm.num_cols() - 1 - ii));
00141   }
00142   west_coeffs[0] += -exp(-1.0);
00143   east_coeffs[0] += -exp(-1.0);
00144   mtk::BCDescriptor1D::ImposeOnLaplacianMatrix(lapm,
      west_coeffs, east_coeffs);
```

```
00145   mtk::BCDescriptor1D::ImposeOnGrid(source, -1.0, 0.0);
00146
00147   int info{mtk::LAPACKAdapter::SolveDenseSystem(lapm, source)};
00148   if (info != 0) {
00149     std::cerr << "Something wrong solving system! info = " << info << std::endl;
00150     return EXIT_FAILURE;
00151   }
00152
00153   source.WriteToFile("minimalistic_poisson_1d_comp_sol.dat", "x", "~u(x)");
00154   known_sol.BindScalarField(KnownSolution);
00155   relative_norm_2_error =
00156     mtk::BLASAdapter::RelNorm2Error(source.discrete_field_u(),
00157                                     known_sol.discrete_field_u(),
00158                                     known_sol.num_cells_x());
00159   std::cout << "relative_norm_2_error = ";
00160   std::cout << relative_norm_2_error << std::endl;
00161 }
00162
00163 #else
00164 #include <iostream>
00165 using std::cout;
00166 using std::endl;
00167 int main () {
00168   cout << "This code HAS to be compiled with support for C++11." << endl;
00169   cout << "Exiting..." << endl;
00170   return EXIT_SUCCESS;
00171 }
00172 #endif
```

17.3 examples/poisson_1d/poisson_1d.cc File Reference

Poisson Equation on a 1D Uniform Staggered Grid with Robin BCs.

`#include <iostream>`
Include dependency graph for poisson_1d.cc:



Functions

- int main ()

17.3.1 Detailed Description

We solve:

$$\nabla^2 p(x) = -s(x),$$

for $x \in \Omega = [a,b] = [0,1]$.

The source term function is defined as

$$s(x) = \frac{\lambda^2 \exp(\lambda x)}{\exp(\lambda) - 1}$$

where $\lambda = -1$ is a parameter.

We consider Robin's boundary conditions of the form:

$$\alpha p(a) - \beta p'(a) = \omega,$$

$$\alpha p(b) + \beta p'(b) = \varepsilon.$$

The analytical solution for this problem is given by

$$p(x) = \frac{\exp(\lambda x) - 1}{\exp(\lambda) - 1}.$$

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu
: Raul Vargas–Navarro - vargasna at rohan dot sdsu dot edu

Definition in file poisson_1d.cc.

17.3.2 Function Documentation

17.3.2.1 int main ( )

Definition at line 261 of file poisson_1d.cc.

17.4 poisson_1d.cc

```
00001
00042 /*
00043 Copyright (C) 2015, Computational Science Research Center, San Diego State
00044 University. All rights reserved.
00045
00046 Redistribution and use in source and binary forms, with or without modification,
00047 are permitted provided that the following conditions are met:
00048
00049 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00050 and a copy of the modified files should be reported once modifications are
00051 completed, unless these modifications are made through the project's GitHub
00052 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00053 should be developed and included in any deliverable.
00054
00055 2. Redistributions of source code must be done through direct
00056 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00057
00058 3. Redistributions in binary form must reproduce the above copyright notice,
00059 this list of conditions and the following disclaimer in the documentation and/or
00060 other materials provided with the distribution.
00061
00062 4. Usage of the binary form on proprietary applications shall require explicit
```

```
00063 prior written permission from the the copyright holders, and due credit should
00064 be given to the copyright holders.
00065
00066 5. Neither the name of the copyright holder nor the names of its contributors
00067 may be used to endorse or promote products derived from this software without
00068 specific prior written permission.
00069
00070 The copyright holders provide no reassurances that the source code provided does
00071 not infringe any patent, copyright, or any other intellectual property rights of
00072 third parties. The copyright holders disclaim any liability to any recipient for
00073 claims brought against recipient by any third party for infringement of that
00074 parties intellectual property rights.
00075
00076 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00077 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00078 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00079 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00080 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00081 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00082 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00083 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00084 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00085 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00086 */
00087
00088 #if __cplusplus == 201103L
00089
00090 #include <iostream>
00091 #include <fstream>
00092 #include <cmath>
00093
00094 #include <vector>
00095
00096 #include "mtk.h"
00097
00098 mtk::Real Source(mtk::Real xx) {
00099
00100   mtk::Real lambda = -1.0;
00101
00102   return lambda*lambda*exp(lambda*xx)/(exp(lambda) - 1.0);
00103 }
00104
00105 mtk::Real KnownSolution(mtk::Real xx) {
00106
00107   mtk::Real lambda = -1.0;
00108
00109   return (exp(lambda*xx) - 1.0)/(exp(lambda) - 1.0);
00110 }
00111
00112 int main () {
00113
00114   std::cout << "Example: Poisson Equation on a 1D Uniform Staggered Grid ";
00115   std::cout << "with Robin BCs." << std::endl;
00116
00118
00119   mtk::Real lambda = -1.0;
00120   mtk::Real alpha = -exp(lambda);
00121   mtk::Real beta = (exp(lambda) - 1.0)/lambda;
00122   mtk::Real omega = -1.0;
00123   mtk::Real epsilon = 0.0;
00124
00126
00127   mtk::Real west_bndy_x = 0.0;
00128   mtk::Real east_bndy_x = 1.0;
00129   int num_cells_x = 5;
00130
00131   mtk::UniStgGrid1D comp_sol(west_bndy_x, east_bndy_x, num_cells_x);
00132
00134
00135   int order_of_accuracy{2};  // Desired order of accuracy for approximation.
00136
00137   mtk::Grad1D grad;  // Mimetic gradient operator.
00138
00139   mtk::Lap1D lap;  // Mimetic Laplacian operator.
00140
00141   if (!lap.ConstructLap1D(order_of_accuracy)) {
00142     std::cerr << "Mimetic lap could not be built." << std::endl;
00143     return EXIT_FAILURE;
00144   }
00145
00146   mtk::DenseMatrix lapm(lap.ReturnAsDenseMatrix(comp_sol));
```

```
00147
00148   std::cout << "Mimetic Laplacian operator: " << std::endl;
00149   std::cout << lapm << std::endl;
00150
00151   if (!grad.ConstructGrad1D(order_of_accuracy)) {
00152     std::cerr << "Mimetic grad could not be built." << std::endl;
00153     return EXIT_FAILURE;
00154   }
00155
00156   mtk::DenseMatrix gradm(grad.ReturnAsDenseMatrix(comp_sol));
00157
00158   std::cout << "Mimetic gradient operator: " << std::endl;
00159   std::cout << gradm << std::endl;
00160
00162
00163   mtk::UniStgGrid1D source(west_bndy_x, east_bndy_x, num_cells_x);
00164
00165   source.BindScalarField(Source);
00166
00167   std::cout << source << std::endl;
00168
00170
00171   // Since we need to approximate the first derivative times beta, we must use
00172   // the approximation of the gradient at the boundary. We could extract them
00173   // from the gradient operator as packed in the grad object. BUT, since we have
00174   // generated at matrix containing this operator, we can extract these from the
00175   // matrix.
00176
00177   // Array containing the coefficients for the west boundary condition.
00178   std::vector<mtk::Real> west_coeffs;
00179
00180   for (auto ii = 0; ii < grad.num_bndy_coeffs(); ++ii) {
00181     west_coeffs.push_back(-beta*gradm.GetValue(0, ii));
00182   }
00183
00184   // Array containing the coefficients for the east boundary condition.
00185   std::vector<mtk::Real> east_coeffs;
00186
00187   for (auto ii = 0; ii < grad.num_bndy_coeffs(); ++ii) {
00188     east_coeffs.push_back(beta*gradm.GetValue(gradm.num_rows() - 1,
00189                                               gradm.num_cols() - 1 - ii));
00190   }
00191
00192   // To impose the Dirichlet condition, we simple add its coefficient to the
00193   // first entry of the west, and the last entry of the east array.
00194
00195   west_coeffs[0] += alpha;
00196
00197   east_coeffs[0] += alpha;
00198
00199   // Now that we have the coefficients that should be in the operator, we create
00200   // a boundary condition descriptor object, which will encapsulate the
00201   // complexity of assigning them in the matrix, to complete the construction of
00202   // the mimetic operator.
00203
00204   mtk::BCDescriptor1D::ImposeOnLaplacianMatrix(lapm,
    west_coeffs, east_coeffs);
00205
00206   std::cout << "Mimetic Laplacian with Robin conditions:" << std::endl;
00207   std::cout << lapm << std::endl;
00208
00209   mtk::BCDescriptor1D::ImposeOnGrid(source, omega, epsilon);
00210
00211   std::cout << "Source term with imposed BCs:" << std::endl;
00212   std::cout << source << std::endl;
00213
00214   source.WriteToFile("poisson_1d_source.dat", "x", "s(x)");
00215
00217
00218   int info{mtk::LAPACKAdapter::SolveDenseSystem(lapm, source)};
00219
00220   if (!info) {
00221     std::cout << "System solved! Problem solved!" << std::endl;
00222     std::cout << std::endl;
00223   }
00224   else {
00225     std::cerr << "Something wrong solving system! info = " << info << std::endl;
00226     std::cerr << "Exiting..." << std::endl;
00227     return EXIT_FAILURE;
00228   }
00229
```

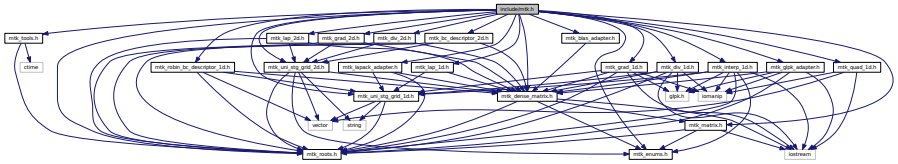
```
00230   std::cout << "Computed solution:" << std::endl;
00231   std::cout << source << std::endl;
00232
00233   source.WriteToFile("poisson_1d_comp_sol.dat", "x", "~u(x)");
00234
00236
00237   mtk::UniStgGrid1D known_sol(west_bndy_x, east_bndy_x, num_cells_x);
00238
00239   known_sol.BindScalarField(KnownSolution);
00240
00241   std::cout << "known_sol =" << std::endl;
00242   std::cout << known_sol << std::endl;
00243
00244   known_sol.WriteToFile("poisson_1d_known_sol.dat", "x", "u(x)");
00245
00246   mtk::Real relative_norm_2_error{};  // Relative norm 2 of the error.
00247
00248   relative_norm_2_error =
00249     mtk::BLASAdapter::RelNorm2Error(source.discrete_field_u(),
00250                                     known_sol.discrete_field_u(),
00251                                     known_sol.num_cells_x());
00252
00253   std::cout << "relative_norm_2_error = ";
00254   std::cout << relative_norm_2_error << std::endl;
00255 }
00256
00257 #else
00258 #include <iostream>
00259 using std::cout;
00260 using std::endl;
00261 int main () {
00262   cout << "This code HAS to be compiled with support for C++11." << endl;
00263   cout << "Exiting..." << endl;
00264   return EXIT_SUCCESS;
00265 }
00266 #endif
```

17.5 include/mtk.h File Reference

Includes the entire API.

```
#include "mtk_roots.h"
#include "mtk_enums.h"
#include "mtk_tools.h"
#include "mtk_matrix.h"
#include "mtk_dense_matrix.h"
#include "mtk_blas_adapter.h"
#include "mtk_lapack_adapter.h"
#include "mtk_glpk_adapter.h"
#include "mtk_uni_stg_grid_1d.h"
#include "mtk_uni_stg_grid_2d.h"
#include "mtk_grad_1d.h"
#include "mtk_div_1d.h"
#include "mtk_lap_1d.h"
#include "mtk_robin_bc_descriptor_1d.h"
#include "mtk_quad_1d.h"
#include "mtk_interp_1d.h"
#include "mtk_grad_2d.h"
#include "mtk_div_2d.h"
#include "mtk_lap_2d.h"
#include "mtk_bc_descriptor_2d.h"
```

Include dependency graph for mtk.h:



17.5.1 Detailed Description

This file contains every required header file, thus containing the entire API. In this way, client codes only have to instruct #include "mtk.h".

Warning

IT IS EXTREMELY IMPORTANT THAT THE HEADERS ARE ADDED TO THIS FILE IN A SPECIFIC ORDER; THAT IS, CONSIDERING THE DEPENDENCE BETWEEN THE CLASSES THESE CONTAIN!

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk.h.

17.6 mtk.h

```
00001
00015 /*
00016 Copyright (C) 2015, Computational Science Research Center, San Diego State
00017 University. All rights reserved.
00018
00019 Redistribution and use in source and binary forms, with or without modification,
00020 are permitted provided that the following conditions are met:
00021
00022 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00023 and a copy of the modified files should be reported once modifications are
00024 completed, unless these modifications are made through the project's GitHub
00025 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00026 should be developed and included in any deliverable.
00027
00028 2. Redistributions of source code must be done through direct
00029 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00030
00031 3. Redistributions in binary form must reproduce the above copyright notice,
00032 this list of conditions and the following disclaimer in the documentation and/or
00033 other materials provided with the distribution.
00034
00035 4. Usage of the binary form on proprietary applications shall require explicit
00036 prior written permission from the the copyright holders, and due credit should
00037 be given to the copyright holders.
00038
00039 5. Neither the name of the copyright holder nor the names of its contributors
00040 may be used to endorse or promote products derived from this software without
00041 specific prior written permission.
00042
00043 The copyright holders provide no reassurances that the source code provided does
00044 not infringe any patent, copyright, or any other intellectual property rights of
00045 third parties. The copyright holders disclaim any liability to any recipient for
00046 claims brought against recipient by any third party for infringement of that
00047 parties intellectual property rights.
00048
```

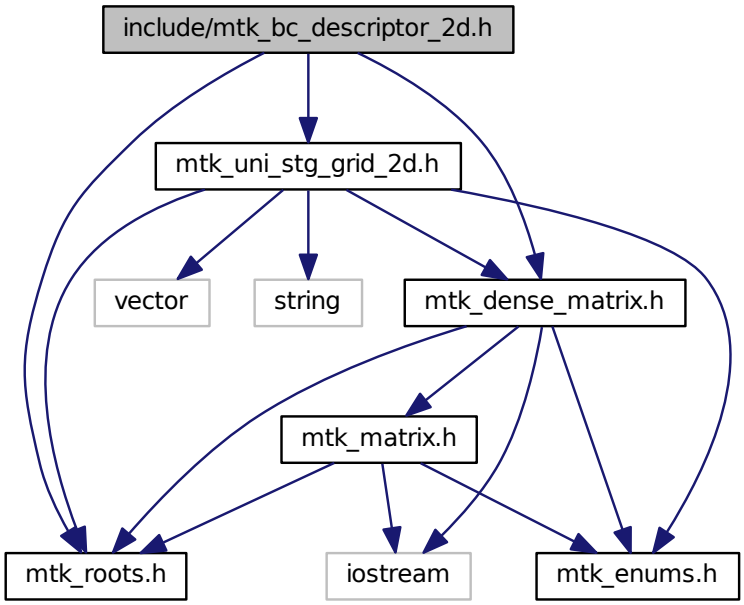
```
00049 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00050 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00051 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00052 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00053 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00054 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00055 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00056 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00057 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00058 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00059 */
00379 #ifndef MTK_INCLUDE_MTK_H_
00380 #define MTK_INCLUDE_MTK_H_
00381
00389 #include "mtk_roots.h"
00390
00398 #include "mtk_enums.h"
00399
00407 #include "mtk_tools.h"
00408
00416 #include "mtk_matrix.h"
00417 #include "mtk_dense_matrix.h"
00418
00426 #include "mtk_blas_adapter.h"
00427 #include "mtk_lapack_adapter.h"
00428 #include "mtk_glpk_adapter.h"
00429
00437 #include "mtk_uni_stg_grid_1d.h"
00438 #include "mtk_uni_stg_grid_2d.h"
00439
00447 #include "mtk_grad_1d.h"
00448 #include "mtk_div_1d.h"
00449 #include "mtk_lap_1d.h"
00450 #include "mtk_robin_bc_descriptor_1d.h"
00451 #include "mtk_quad_1d.h"
00452 #include "mtk_interp_1d.h"
00453
00454 #include "mtk_grad_2d.h"
00455 #include "mtk_div_2d.h"
00456 #include "mtk_lap_2d.h"
00457 #include "mtk_bc_descriptor_2d.h"
00458
00459 #endif // End of: MTK_INCLUDE_MTK_H_
```

17.7 include/mtk_bc_descriptor_2d.h File Reference

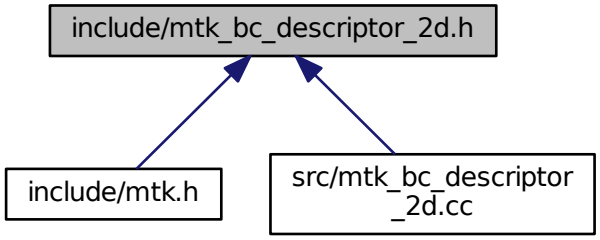
Imposes mimetic boundary conditions on the operators and on the grids.

```
#include "mtk_roots.h"
#include "mtk_dense_matrix.h"
#include "mtk_uni_stg_grid_2d.h"
```

Include dependency graph for mtk_bc_descriptor_2d.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mtk::BCDescriptor2D

  Enforces boundary conditions in either the operator or the grid.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

Typedefs

- typedef Real(∗ mtk::CoefficientFunction2D)(const Real &, const Real &)

  A function of a BC coefficient evaluated on a 2D domain.

17.7.1 Detailed Description

This class presents an interface for the user to specify mimetic boundary conditions on 2D mimetic operators and the grids they are acting on.

Def. Let $f(t,\mathbf{x})$ be any scalar field defined over a domain Ω on a given time snapshot $t \in [t_0,t_n]$. Any linear combination of f and its m derivatives fulfilling a condition $\beta(t,\mathbf{x})$ is defined as a **boundary condition**:

$$\forall t \in [t_0,t_n]\ \forall \mathbf{x} \in \partial\Omega : \sum_{i=0}^{m} c_i(\mathbf{x},t) < \hat{\mathbf{n}}, \frac{\partial^i f}{\partial \mathbf{x}^i}(\mathbf{x},t) >= \beta(\mathbf{x},t).$$

This class receives information about all possible coefficient functions, $c_i(\mathbf{x},t)$ for any subset of the boundary (south, north, west and east), and every condition $\beta(\mathbf{x},t)$ for any subset of the boundary, and takes care of assigning them to both the differentiation matrices and the grids. The highest order of differentiation is computed based on how many coefficients have been given at run-time.

Warning

For now, we only support orders as high as 1.

See also

http://mathworld.wolfram.com/NormalVector.html

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_bc_descriptor_2d.h.

17.8 mtk_bc_descriptor_2d.h

```
00001
00036 /*
00037 Copyright (C) 2015, Computational Science Research Center, San Diego State
00038 University. All rights reserved.
00039
00040 Redistribution and use in source and binary forms, with or without modification,
00041 are permitted provided that the following conditions are met:
00042
00043 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00044 and a copy of the modified files should be reported once modifications are
00045 completed, unless these modifications are made through the project's GitHub
00046 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
```

```
00047 should be developed and included in any deliverable.
00048
00049 2. Redistributions of source code must be done through direct
00050 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00051
00052 3. Redistributions in binary form must reproduce the above copyright notice,
00053 this list of conditions and the following disclaimer in the documentation and/or
00054 other materials provided with the distribution.
00055
00056 4. Usage of the binary form on proprietary applications shall require explicit
00057 prior written permission from the the copyright holders, and due credit should
00058 be given to the copyright holders.
00059
00060 5. Neither the name of the copyright holder nor the names of its contributors
00061 may be used to endorse or promote products derived from this software without
00062 specific prior written permission.
00063
00064 The copyright holders provide no reassurances that the source code provided does
00065 not infringe any patent, copyright, or any other intellectual property rights of
00066 third parties. The copyright holders disclaim any liability to any recipient for
00067 claims brought against recipient by any third party for infringement of that
00068 parties intellectual property rights.
00069
00070 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00071 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00072 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00073 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00074 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00075 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00076 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00077 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00078 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00079 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00080 */
00081
00082 #ifndef MTK_INCLUDE_BC_DESCRIPTOR_2D_H_
00083 #define MTK_INCLUDE_BC_DESCRIPTOR_2D_H_
00084
00085 #include "mtk_roots.h"
00086 #include "mtk_dense_matrix.h"
00087 #include "mtk_uni_stg_grid_2d.h"
00088
00089 namespace mtk{
00090
00098 typedef Real (*CoefficientFunction2D)(const Real &, const Real &);
00099
00130 class BCDescriptor2D {
00131  public:
00133   BCDescriptor2D();
00134
00140   BCDescriptor2D(const BCDescriptor2D &desc);
00141
00143   ~BCDescriptor2D() noexcept;
00144
00150   int highest_order_diff_west() const noexcept;
00151
00157   int highest_order_diff_east() const noexcept;
00158
00164   int highest_order_diff_south() const noexcept;
00165
00171   int highest_order_diff_north() const noexcept;
00172
00178   void PushBackWestCoeff(CoefficientFunction2D cw);
00179
00185   void PushBackEastCoeff(CoefficientFunction2D ce);
00186
00192   void PushBackSouthCoeff(CoefficientFunction2D cs);
00193
00199   void PushBackNorthCoeff(CoefficientFunction2D cn);
00200
00206   void set_west_condition(Real (*west_condition)(Real xx, Real yy)) noexcept;
00207
00213   void set_east_condition(Real (*east_condition)(Real xx, Real yy)) noexcept;
00214
00220   void set_south_condition(Real (*south_condition)(Real xx, Real yy)) noexcept;
00221
00227   void set_north_condition(Real (*north_condition)(Real xx, Real yy)) noexcept;
00228
00236   void ImposeOnLaplacianMatrix(const UniStgGrid2D &grid,
00237                                DenseMatrix &matrix,
00238                                const int &order_accuracy = 2) const;
```

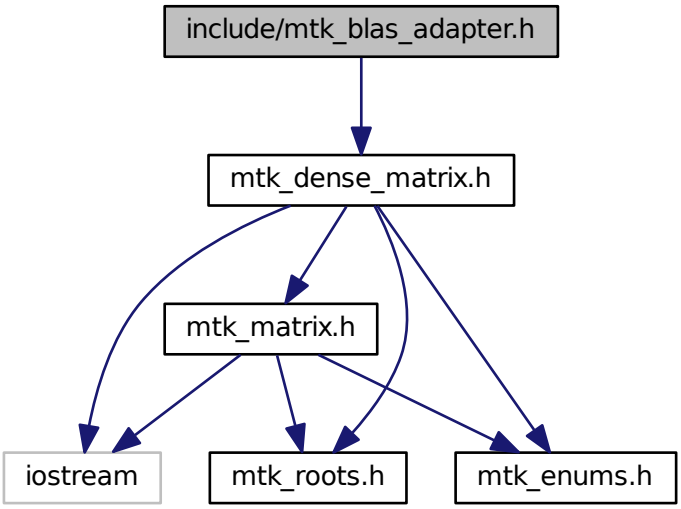
```
00239
00245   void ImposeOnGrid(UniStgGrid2D &grid) const;
00246
00247 private:
00255   void ImposeOnSouthBoundaryNoSpace(const
      mtk::UniStgGrid2D &grid,
00256                                     mtk::DenseMatrix &matrix,
00257                                     const int &order_accuracy) const;
00265   void ImposeOnNorthBoundaryNoSpace(const
      mtk::UniStgGrid2D &grid,
00266                                     mtk::DenseMatrix &matrix,
00267                                     const int &order_accuracy) const;
00275   void ImposeOnWestBoundaryNoSpace(const
      mtk::UniStgGrid2D &grid,
00276                                    mtk::DenseMatrix &matrix,
00277                                    const int &order_accuracy) const;
00285   void ImposeOnEastBoundaryNoSpace(const
      mtk::UniStgGrid2D &grid,
00286                                    mtk::DenseMatrix &matrix,
00287                                    const int &order_accuracy) const;
00295   void ImposeOnSouthBoundaryWithSpace(const
      mtk::UniStgGrid2D &grid,
00296                                       mtk::DenseMatrix &matrix,
00297                                       const int &order_accuracy) const;
00305   void ImposeOnNorthBoundaryWithSpace(const
      mtk::UniStgGrid2D &grid,
00306                                       mtk::DenseMatrix &matrix,
00307                                       const int &order_accuracy) const;
00315   void ImposeOnWestBoundaryWithSpace(const
      mtk::UniStgGrid2D &grid,
00316                                      mtk::DenseMatrix &matrix,
00317                                      const int &order_accuracy) const;
00325   void ImposeOnEastBoundaryWithSpace(const
      mtk::UniStgGrid2D &grid,
00326                                      mtk::DenseMatrix &matrix,
00327                                      const int &order_accuracy) const;
00328
00329   int highest_order_diff_west_;
00330   int highest_order_diff_east_;
00331   int highest_order_diff_south_;
00332   int highest_order_diff_north_;
00333
00334   std::vector<CoefficientFunction2D> west_coefficients_;
00335   std::vector<CoefficientFunction2D> east_coefficients_;
00336   std::vector<CoefficientFunction2D> south_coefficients_;
00337   std::vector<CoefficientFunction2D> north_coefficients_;
00338
00339   Real (*west_condition_)(Real xx, Real yy);
00340   Real (*east_condition_)(Real xx, Real yy);
00341   Real (*south_condition_)(Real xx, Real yy);
00342   Real (*north_condition_)(Real xx, Real yy);
00343 };
00344 }
00345 #endif  // End of: MTK_INCLUDE_BC_DESCRIPTOR_2D_H_
```

17.9 include/mtk_blas_adapter.h File Reference

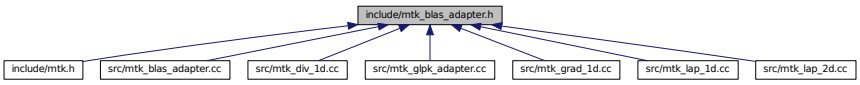
Adapter class for the BLAS API.

```
#include "mtk_dense_matrix.h"
```

Include dependency graph for mtk_blas_adapter.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mtk::BLASAdapter

  Adapter class for the BLAS API.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

17.9.1 Detailed Description

This class contains a collection of static classes, that posses direct access to the underlying structure of the matrices, thus allowing programmers to exploit some of the numerical methods implemented in the BLAS.

The **BLAS (Basic Linear Algebra Subprograms)** are routines that provide standard building blocks for performing basic vector and matrix operations. The Level 1 BLAS perform scalar, vector and vector-vector operations, the Level 2 BLAS perform matrix-vector operations, and the Level 3 BLAS perform matrix-matrix operations.

The BLAS can be installed from links given in the See Also section of this page.

See also

http://www.netlib.org/blas/
https://software.intel.com/en-us/non-commercial-software-development

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_blas_adapter.h.

17.10 mtk_blas_adapter.h

```
00001
00024 /*
00025 Copyright (C) 2015, Computational Science Research Center, San Diego State
00026 University. All rights reserved.
00027
00028 Redistribution and use in source and binary forms, with or without modification,
00029 are permitted provided that the following conditions are met:
00030
00031 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00032 and a copy of the modified files should be reported once modifications are
00033 completed, unless these modifications are made through the project's GitHub
00034 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00035 should be developed and included in any deliverable.
00036
00037 2. Redistributions of source code must be done through direct
00038 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00039
00040 3. Redistributions in binary form must reproduce the above copyright notice,
00041 this list of conditions and the following disclaimer in the documentation and/or
00042 other materials provided with the distribution.
00043
00044 4. Usage of the binary form on proprietary applications shall require explicit
00045 prior written permission from the the copyright holders, and due credit should
00046 be given to the copyright holders.
00047
00048 5. Neither the name of the copyright holder nor the names of its contributors
00049 may be used to endorse or promote products derived from this software without
00050 specific prior written permission.
00051
00052 The copyright holders provide no reassurances that the source code provided does
00053 not infringe any patent, copyright, or any other intellectual property rights of
00054 third parties. The copyright holders disclaim any liability to any recipient for
00055 claims brought against recipient by any third party for infringement of that
00056 parties intellectual property rights.
00057
00058 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00059 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00060 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00061 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00062 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00063 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00064 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00065 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00066 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00067 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00068 */
00069
00070 #ifndef MTK_INCLUDE_BLAS_ADAPTER_H_
00071 #define MTK_INCLUDE_BLAS_ADAPTER_H_
00072
00073 #include "mtk_dense_matrix.h"
```

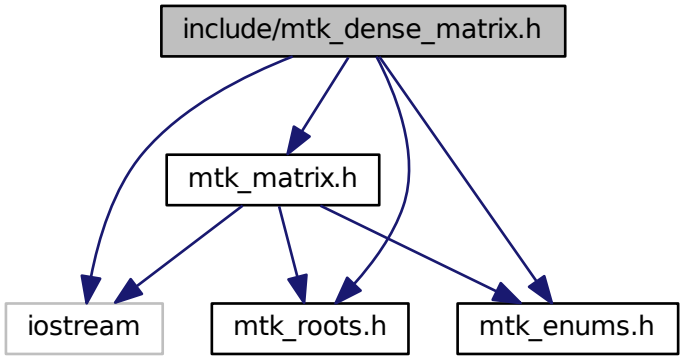
```
00074
00075 namespace mtk {
00076
00096 class BLASAdapter {
00097  public:
00106   static Real RealNRM2(Real *in, int &in_length);
00107
00124   static void RealAXPY(Real alpha, Real *xx, Real *yy, int &in_length);
00125
00140   static Real RelNorm2Error(Real *computed, Real *known, int length);
00141
00159   static void RealDenseMV(Real &alpha,
00160                           DenseMatrix &aa,
00161                           Real *xx,
00162                           Real &beta,
00163                           Real *yy);
00164
00179   static DenseMatrix RealDenseMM(DenseMatrix &aa,
      DenseMatrix &bb);
00180 };
00181 }
00182 #endif  // End of: MTK_INCLUDE_BLAS_ADAPTER_H_
```

17.11 include/mtk_dense_matrix.h File Reference

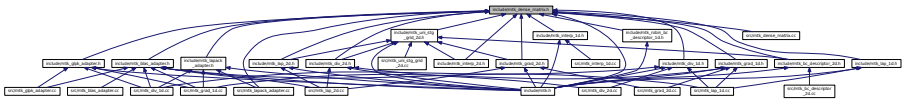
Defines a common dense matrix, using a 1D array.

```
#include <iostream>
#include "mtk_roots.h"
#include "mtk_enums.h"
#include "mtk_matrix.h"
```

Include dependency graph for mtk_dense_matrix.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mtk::DenseMatrix

  Defines a common dense matrix, using a 1D array.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

17.11.1 Detailed Description

For developing purposes, it is better to have a not-so-intrincated data structure implementing matrices. This is the purpose of this class: to be used for prototypes of new code for small test cases. In every other instance, this should be replaced by the most appropriate sparse matrix.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Note

We prefer composition to inheritance [Reedy, 2011]. The main reason for this preference is that inheritance produces a more tightly coupled design. When a class inherits from another type be it public, protected, or private inheritance the subclass gains access to all public and protected members of the base class, whereas with composition, the class is only coupled to the public members of the other class. Furthermore, if you only hold a pointer to the other object, then your interface can use a forward declaration of the class rather than #include its full definition. This results in greater compile-time insulation and improves the time it takes to compile your code.

Definition in file mtk_dense_matrix.h.

17.12 mtk_dense_matrix.h

```
00001
00023 /*
00024 Copyright (C) 2015, Computational Science Research Center, San Diego State
00025 University. All rights reserved.
00026
00027 Redistribution and use in source and binary forms, with or without modification,
00028 are permitted provided that the following conditions are met:
00029
00030 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00031 and a copy of the modified files should be reported once modifications are
00032 completed, unless these modifications are made through the project's GitHub
00033 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00034 should be developed and included in any deliverable.
00035
00036 2. Redistributions of source code must be done through direct
00037 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00038
00039 3. Redistributions in binary form must reproduce the above copyright notice,
00040 this list of conditions and the following disclaimer in the documentation and/or
00041 other materials provided with the distribution.
00042
00043 4. Usage of the binary form on proprietary applications shall require explicit
00044 prior written permission from the the copyright holders, and due credit should
00045 be given to the copyright holders.
00046
00047 5. Neither the name of the copyright holder nor the names of its contributors
```

```
00048 may be used to endorse or promote products derived from this software without
00049 specific prior written permission.
00050
00051 The copyright holders provide no reassurances that the source code provided does
00052 not infringe any patent, copyright, or any other intellectual property rights of
00053 third parties. The copyright holders disclaim any liability to any recipient for
00054 claims brought against recipient by any third party for infringement of that
00055 parties intellectual property rights.
00056
00057 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00058 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00059 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00060 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00061 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00062 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00063 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00064 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00065 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00066 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00067 */
00068
00069 #ifndef MTK_INCLUDE_DENSE_MATRIX_H_
00070 #define MTK_INCLUDE_DENSE_MATRIX_H_
00071
00072 #include <iostream>
00073
00074 #include "mtk_roots.h"
00075 #include "mtk_enums.h"
00076 #include "mtk_matrix.h"
00077
00078 namespace mtk {
00079
00092 class DenseMatrix {
00093  public:
00095   friend std::ostream& operator <<(std::ostream &stream, DenseMatrix &in);
00096
00098   DenseMatrix& operator =(const DenseMatrix &in);
00099
00101   bool operator ==(const DenseMatrix &in);
00102
00104   DenseMatrix();
00105
00111   DenseMatrix(const DenseMatrix &in);
00112
00121   DenseMatrix(const int &num_rows, const int &num_cols);
00122
00148   DenseMatrix(const int &rank, const bool &padded, const bool &transpose);
00149
00183   DenseMatrix(const Real *const gen,
00184               const int &gen_length,
00185               const int &pro_length,
00186               const bool &transpose);
00187
00189   ~DenseMatrix();
00190
00196   Matrix matrix_properties() const noexcept;
00197
00203   int num_rows() const noexcept;
00204
00210   int num_cols() const noexcept;
00211
00217   Real* data() const noexcept;
00218
00226   void SetOrdering(mtk::MatrixOrdering oo) noexcept;
00227
00236   Real GetValue(const int &row_coord, const int &col_coord) const noexcept;
00237
00245   void SetValue(const int &row_coord,
00246                 const int &col_coord,
00247                 const Real &val) noexcept;
00248
00250   void Transpose();
00251
00253   void OrderRowMajor();
00254
00256   void OrderColMajor();
00257
00268   static DenseMatrix Kron(const DenseMatrix &aa,
00269                           const DenseMatrix &bb);
00270
00280   bool WriteToFile(const std::string &filename) const;
```

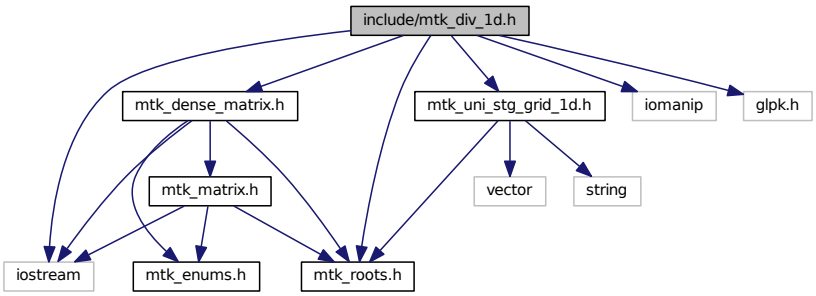
```
00281
00282  private:
00283   Matrix matrix_properties_;
00284
00285   Real *data_;
00286 };
00287 }
00288 #endif  // End of: MTK_INCLUDE_MTK_DENSE_MATRIX_H_
```

17.13 include/mtk_div_1d.h File Reference

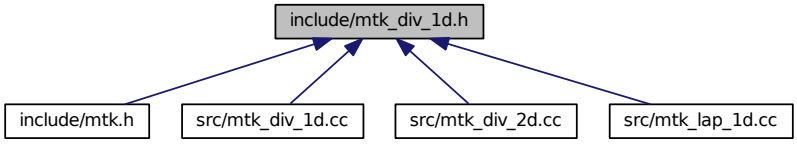
Includes the definition of the class Div1D.

```
#include <iostream>
#include <iomanip>
#include "glpk.h"
#include "mtk_roots.h"
#include "mtk_dense_matrix.h"
#include "mtk_uni_stg_grid_1d.h"
```

Include dependency graph for mtk_div_1d.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mtk::Div1D

  Implements a 1D mimetic divergence operator.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

17.13.1 Detailed Description

This class implements a 1D divergence operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm (CBSA).

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_div_1d.h.

17.14 mtk_div_1d.h

```
00001
00011 /*
00012 Copyright (C) 2015, Computational Science Research Center, San Diego State
00013 University. All rights reserved.
00014
00015 Redistribution and use in source and binary forms, with or without modification,
00016 are permitted provided that the following conditions are met:
00017
00018 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00019 and a copy of the modified files should be reported once modifications are
00020 completed, unless these modifications are made through the project's GitHub
00021 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00022 should be developed and included in any deliverable.
00023
00024 2. Redistributions of source code must be done through direct
00025 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00026
00027 3. Redistributions in binary form must reproduce the above copyright notice,
00028 this list of conditions and the following disclaimer in the documentation and/or
00029 other materials provided with the distribution.
00030
00031 4. Usage of the binary form on proprietary applications shall require explicit
00032 prior written permission from the the copyright holders, and due credit should
00033 be given to the copyright holders.
00034
00035 5. Neither the name of the copyright holder nor the names of its contributors
00036 may be used to endorse or promote products derived from this software without
00037 specific prior written permission.
00038
00039 The copyright holders provide no reassurances that the source code provided does
00040 not infringe any patent, copyright, or any other intellectual property rights of
00041 third parties. The copyright holders disclaim any liability to any recipient for
00042 claims brought against recipient by any third party for infringement of that
00043 parties intellectual property rights.
00044
00045 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00046 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00047 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00048 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00049 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00050 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00051 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00052 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00053 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00054 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00055 */
00056
00057 #ifndef MTK_INCLUDE_DIV_1D_H_
00058 #define MTK_INCLUDE_DIV_1D_H_
00059
00060 #include <iostream>
```

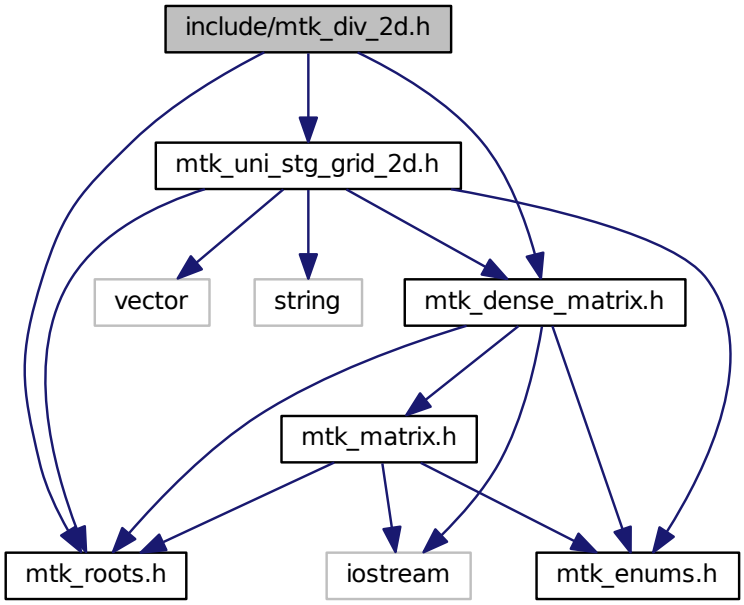
```
00061 #include <iomanip>
00062
00063 #include "glpk.h"
00064
00065 #include "mtk_roots.h"
00066 #include "mtk_dense_matrix.h"
00067 #include "mtk_uni_stg_grid_1d.h"
00068
00069 namespace mtk {
00070
00081 class Div1D {
00082  public:
00084   friend std::ostream& operator <<(std::ostream& stream, Div1D &in);
00085
00087   Div1D();
00088
00094   Div1D(const Div1D &div);
00095
00097   ~Div1D();
00098
00104   bool ConstructDiv1D(int order_accuracy = kDefaultOrderAccuracy,
00105                       Real mimetic_threshold = kDefaultMimeticThreshold);
00106
00112   int num_bndy_coeffs() const;
00113
00119   Real *coeffs_interior() const;
00120
00126   Real *weights_crs(void) const;
00127
00133   Real *weights_cbs(void) const;
00134
00140   DenseMatrix mim_bndy() const;
00141
00147   DenseMatrix ReturnAsDenseMatrix(const
      UniStgGrid1D &grid) const;
00148
00149  private:
00155   bool ComputeStencilInteriorGrid(void);
00156
00163   bool ComputeRationalBasisNullSpace(void);
00164
00170   bool ComputePreliminaryApproximations(void);
00171
00177   bool ComputeWeights(void);
00178
00184   bool ComputeStencilBoundaryGrid(void);
00185
00191   bool AssembleOperator(void);
00192
00193   int order_accuracy_;
00194   int dim_null_;
00195   int num_bndy_coeffs_;
00196   int divergence_length_;
00197   int minrow_;
00198   int row_;
00199
00200   DenseMatrix rat_basis_null_space_;
00201
00202   Real *coeffs_interior_;
00203   Real *prem_apps_;
00204   Real *weights_crs_;
00205   Real *weights_cbs_;
00206   Real *mim_bndy_;
00207   Real *divergence_;
00208
00209   Real mimetic_threshold_;
00210 };
00211 }
00212 #endif  // End of: MTK_INCLUDE_DIV_1D_H_
```

17.15 include/mtk_div_2d.h File Reference

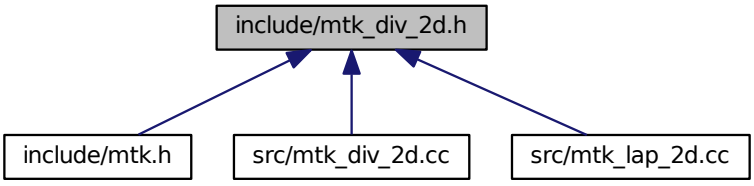
Includes the definition of the class Div2D.

```
#include "mtk_roots.h"
#include "mtk_dense_matrix.h"
#include "mtk_uni_stg_grid_2d.h"
```

Include dependency graph for mtk_div_2d.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mtk::Div2D

  Implements a 2D mimetic divergence operator.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

17.15.1 Detailed Description

This class implements a 2D divergence operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm (CBSA).

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_div_2d.h.

17.16 mtk_div_2d.h

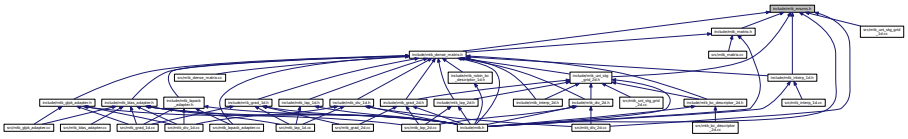
```
00001
00011 /*
00012 Copyright (C) 2015, Computational Science Research Center, San Diego State
00013 University. All rights reserved.
00014
00015 Redistribution and use in source and binary forms, with or without modification,
00016 are permitted provided that the following conditions are met:
00017
00018 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00019 and a copy of the modified files should be reported once modifications are
00020 completed, unless these modifications are made through the project's GitHub
00021 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00022 should be developed and included in any deliverable.
00023
00024 2. Redistributions of source code must be done through direct
00025 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00026
00027 3. Redistributions in binary form must reproduce the above copyright notice,
00028 this list of conditions and the following disclaimer in the documentation and/or
00029 other materials provided with the distribution.
00030
00031 4. Usage of the binary form on proprietary applications shall require explicit
00032 prior written permission from the the copyright holders, and due credit should
00033 be given to the copyright holders.
00034
00035 5. Neither the name of the copyright holder nor the names of its contributors
00036 may be used to endorse or promote products derived from this software without
00037 specific prior written permission.
00038
00039 The copyright holders provide no reassurances that the source code provided does
00040 not infringe any patent, copyright, or any other intellectual property rights of
00041 third parties. The copyright holders disclaim any liability to any recipient for
00042 claims brought against recipient by any third party for infringement of that
00043 parties intellectual property rights.
00044
00045 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00046 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00047 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00048 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00049 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00050 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00051 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00052 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00053 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00054 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00055 */
00056
00057 #ifndef MTK_INCLUDE_MTK_DIV_2D_H_
00058 #define MTK_INCLUDE_MTK_DIV_2D_H_
00059
00060 #include "mtk_roots.h"
```

```
00061 #include "mtk_dense_matrix.h"
00062 #include "mtk_uni_stg_grid_2d.h"
00063
00064 namespace mtk{
00065
00076 class Div2D {
00077  public:
00079   Div2D();
00080
00086   Div2D(const Div2D &div);
00087
00089   ~Div2D();
00090
00096   bool ConstructDiv2D(const UniStgGrid2D &grid,
00097                       int order_accuracy = kDefaultOrderAccuracy,
00098                       Real mimetic_threshold = kDefaultMimeticThreshold);
00099
00105   DenseMatrix ReturnAsDenseMatrix() const;
00106
00107  private:
00108   DenseMatrix divergence_;
00109
00110   int order_accuracy_;
00111
00112   Real mimetic_threshold_;
00113 };
00114 }
00115 #endif  // End of: MTK_INCLUDE_MTK_DIV_2D_H_
```

17.17 include/mtk_enums.h File Reference

Considered enumeration types in the MTK.

This graph shows which files directly or indirectly include this file:



Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

Enumerations

- enum mtk::MatrixStorage { mtk::DENSE, mtk::BANDED, mtk::CRS }

  Considered matrix storage schemes to implement sparse matrices.
- enum mtk::MatrixOrdering { mtk::ROW_MAJOR, mtk::COL_MAJOR }

  Considered matrix ordering (for Fortran purposes).
- enum mtk::FieldNature { mtk::SCALAR, mtk::VECTOR }

  Nature of the field discretized in a given grid.
- enum mtk::DirInterp { mtk::SCALAR_TO_VECTOR, mtk::VECTOR_TO_SCALAR }

  Interpolation operator.

17.17.1 Detailed Description

Enumeration types are used throughout the MTK to differentiate instances of derived classes, as well as for mnemonic purposes. In this file, the enumeration types are listed alphabetically.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_enums.h.

17.18 mtk_enums.h

```
00001
00012 /*
00013 Copyright (C) 2015, Computational Science Research Center, San Diego State
00014 University. All rights reserved.
00015
00016 Redistribution and use in source and binary forms, with or without modification,
00017 are permitted provided that the following conditions are met:
00018
00019 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00020 and a copy of the modified files should be reported once modifications are
00021 completed, unless these modifications are made through the project's GitHub
00022 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00023 should be developed and included in any deliverable.
00024
00025 2. Redistributions of source code must be done through direct
00026 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00027
00028 3. Redistributions in binary form must reproduce the above copyright notice,
00029 this list of conditions and the following disclaimer in the documentation and/or
00030 other materials provided with the distribution.
00031
00032 4. Usage of the binary form on proprietary applications shall require explicit
00033 prior written permission from the the copyright holders, and due credit should
00034 be given to the copyright holders.
00035
00036 5. Neither the name of the copyright holder nor the names of its contributors
00037 may be used to endorse or promote products derived from this software without
00038 specific prior written permission.
00039
00040 The copyright holders provide no reassurances that the source code provided does
00041 not infringe any patent, copyright, or any other intellectual property rights of
00042 third parties. The copyright holders disclaim any liability to any recipient for
00043 claims brought against recipient by any third party for infringement of that
00044 parties intellectual property rights.
00045
00046 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00047 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00048 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00049 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00050 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00051 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00052 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00053 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00054 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00055 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00056 */
00057
00058 #ifndef MTK_INCLUDE_ENUMS_H_
00059 #define MTK_INCLUDE_ENUMS_H_
00060
00061 namespace mtk {
00062
00077 enum MatrixStorage {
00078   DENSE,
00079   BANDED,
00080   CRS
00081 };
00082
00095 enum MatrixOrdering {
00096   ROW_MAJOR,
```

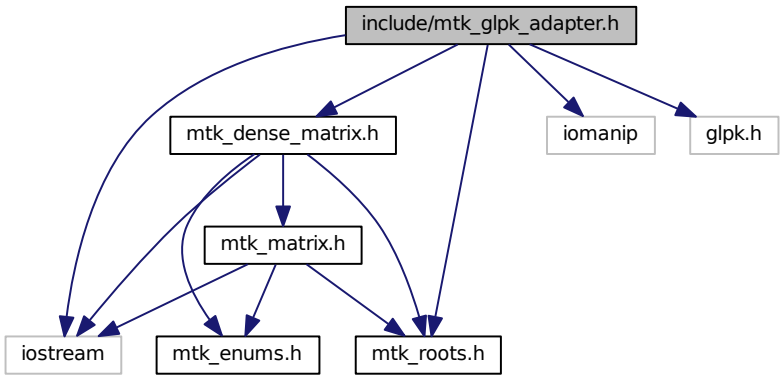
```
00097   COL_MAJOR
00098 };
00099
00113 enum FieldNature {
00114   SCALAR,
00115   VECTOR
00116 };
00117
00127 enum DirInterp {
00128   SCALAR_TO_VECTOR,
00129   VECTOR_TO_SCALAR
00130 };
00131 }
00132 #endif  // End of: MTK_INCLUDE_ENUMS_H_
```

17.19 include/mtk_glpk_adapter.h File Reference

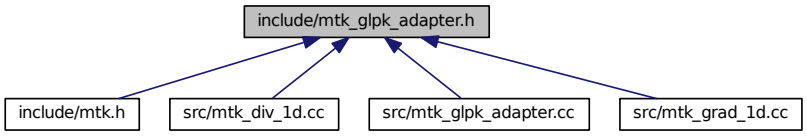
Adapter class for the GLPK API.

```
#include <iostream>
#include <iomanip>
#include "glpk.h"
#include "mtk_roots.h"
#include "mtk_dense_matrix.h"
```

Include dependency graph for mtk_glpk_adapter.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mtk::GLPKAdapter

  Adapter class for the GLPK API.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

17.19.1 Detailed Description

This class contains a collection of static classes, that posses direct access to the underlying structure of the matrices, thus allowing programmers to exploit some of the numerical methods implemented in the GLPK.

The **GLPK (GNU Linear Programming Kit)** package is intended for solving large-scale linear programming (LP), mixed integer programming (MIP), and other related problems. It is a set of routines written in ANSI C and organized in the form of a callable library.

See also

http://www.gnu.org/software/glpk/

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_glpk_adapter.h.

17.20 mtk_glpk_adapter.h

```
00001
00019 /*
00020 Copyright (C) 2015, Computational Science Research Center, San Diego State
00021 University. All rights reserved.
00022
00023 Redistribution and use in source and binary forms, with or without modification,
00024 are permitted provided that the following conditions are met:
00025
00026 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00027 and a copy of the modified files should be reported once modifications are
00028 completed, unless these modifications are made through the project's GitHub
00029 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00030 should be developed and included in any deliverable.
00031
00032 2. Redistributions of source code must be done through direct
00033 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00034
00035 3. Redistributions in binary form must reproduce the above copyright notice,
00036 this list of conditions and the following disclaimer in the documentation and/or
00037 other materials provided with the distribution.
00038
00039 4. Usage of the binary form on proprietary applications shall require explicit
00040 prior written permission from the the copyright holders, and due credit should
00041 be given to the copyright holders.
00042
00043 5. Neither the name of the copyright holder nor the names of its contributors
00044 may be used to endorse or promote products derived from this software without
00045 specific prior written permission.
00046
00047 The copyright holders provide no reassurances that the source code provided does
```

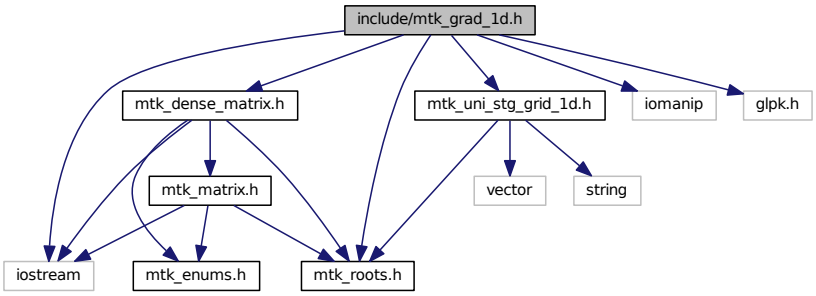
```
00048 not infringe any patent, copyright, or any other intellectual property rights of
00049 third parties. The copyright holders disclaim any liability to any recipient for
00050 claims brought against recipient by any third party for infringement of that
00051 parties intellectual property rights.
00052
00053 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00054 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00055 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00056 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00057 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00058 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00059 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00060 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00061 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00062 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00063 */
00064
00065 #ifndef MTK_INCLUDE_GLPK_ADAPTER_H_
00066 #define MTK_INCLUDE_GLPK_ADAPTER_H_
00067
00068 #include <iostream>
00069 #include <iomanip>
00070
00071 #include "glpk.h"
00072
00073 #include "mtk_roots.h"
00074 #include "mtk_dense_matrix.h"
00075
00076 namespace mtk {
00077
00101 class GLPKAdapter {
00102  public:
00123   static mtk::Real SolveSimplexAndCompare(
      mtk::Real *A,
00124                                           int nrows,
00125                                           int ncols,
00126                                           int kk,
00127                                           mtk::Real *hh,
00128                                           mtk::Real *qq,
00129                                           int robjective,
00130                                           mtk::Real mimetic_tol,
00131                                           int copy);
00132 };
00133 }
00134 #endif  // End of: MTK_INCLUDE_MTK_GLPK_ADAPTER_H_
```

17.21 include/mtk_grad_1d.h File Reference

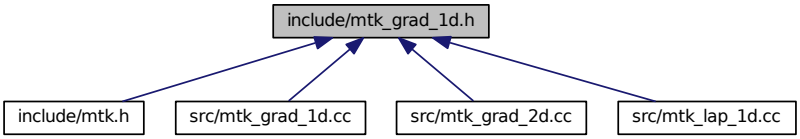
Includes the definition of the class Grad1D.

```
#include <iostream>
#include <iomanip>
#include "glpk.h"
#include "mtk_roots.h"
#include "mtk_dense_matrix.h"
#include "mtk_uni_stg_grid_1d.h"
```

Include dependency graph for mtk_grad_1d.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mtk::Grad1D

  Implements a 1D mimetic gradient operator.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

17.21.1 Detailed Description

This class implements a 1D gradient operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm (C↩ BSA).

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_grad_1d.h.

17.22 mtk_grad_1d.h

```
00001
00011 /*
00012 Copyright (C) 2015, Computational Science Research Center, San Diego State
00013 University. All rights reserved.
00014
00015 Redistribution and use in source and binary forms, with or without modification,
00016 are permitted provided that the following conditions are met:
00017
00018 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00019 and a copy of the modified files should be reported once modifications are
00020 completed, unless these modifications are made through the project's GitHub
00021 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00022 should be developed and included in any deliverable.
00023
00024 2. Redistributions of source code must be done through direct
00025 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00026
00027 3. Redistributions in binary form must reproduce the above copyright notice,
00028 this list of conditions and the following disclaimer in the documentation and/or
00029 other materials provided with the distribution.
00030
00031 4. Usage of the binary form on proprietary applications shall require explicit
00032 prior written permission from the the copyright holders, and due credit should
00033 be given to the copyright holders.
00034
00035 5. Neither the name of the copyright holder nor the names of its contributors
00036 may be used to endorse or promote products derived from this software without
00037 specific prior written permission.
00038
00039 The copyright holders provide no reassurances that the source code provided does
00040 not infringe any patent, copyright, or any other intellectual property rights of
00041 third parties. The copyright holders disclaim any liability to any recipient for
00042 claims brought against recipient by any third party for infringement of that
00043 parties intellectual property rights.
00044
00045 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00046 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00047 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00048 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00049 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00050 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00051 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00052 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00053 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00054 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00055 */
00056
00057 #ifndef MTK_INCLUDE_GRAD_1D_H_
00058 #define MTK_INCLUDE_GRAD_1D_H_
00059
00060 #include <iostream>
00061 #include <iomanip>
00062
00063 #include "glpk.h"
00064
00065 #include "mtk_roots.h"
00066 #include "mtk_dense_matrix.h"
00067 #include "mtk_uni_stg_grid_1d.h"
00068
00069 namespace mtk {
00070
00081 class Grad1D {
00082  public:
00084   friend std::ostream& operator <<(std::ostream& stream, Grad1D &in);
00085
00087   Grad1D();
00088
00094   Grad1D(const Grad1D &grad);
00095
00097   ~Grad1D();
00098
00104   bool ConstructGrad1D(int order_accuracy = kDefaultOrderAccuracy,
00105                        Real mimetic_threshold = kDefaultMimeticThreshold);
00106
00112   int num_bndy_coeffs() const;
00113
00119   Real *coeffs_interior() const;
00120
```

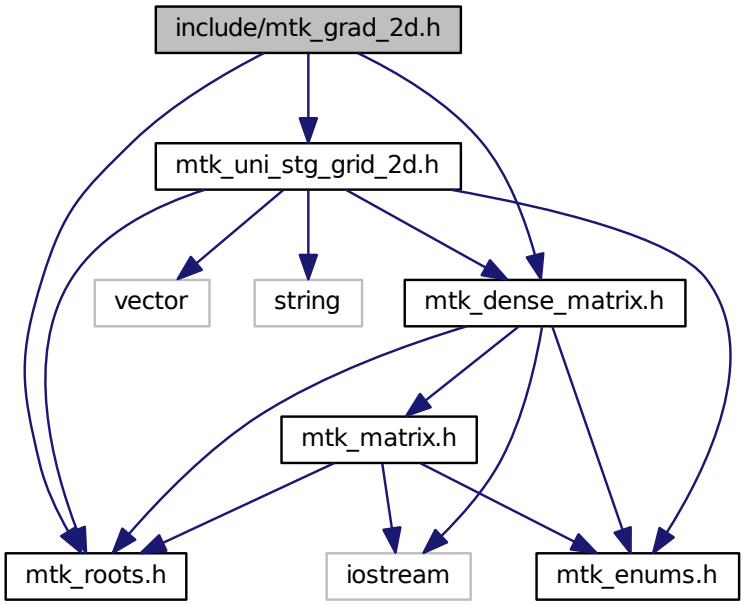
```
00126   Real *weights_crs(void) const;
00127
00133   Real *weights_cbs(void) const;
00134
00140   DenseMatrix mim_bndy() const;
00141
00147   DenseMatrix ReturnAsDenseMatrix(Real west,
      Real east, int num_cells_x) const;
00148
00154   DenseMatrix ReturnAsDenseMatrix(const
      UniStgGrid1D &grid) const;
00155
00161   DenseMatrix ReturnAsDimensionlessDenseMatrix(int num_cells_x)
      const;
00162
00163  private:
00169   bool ComputeStencilInteriorGrid(void);
00170
00177   bool ComputeRationalBasisNullSpace(void);
00178
00184   bool ComputePreliminaryApproximations(void);
00185
00191   bool ComputeWeights(void);
00192
00198   bool ComputeStencilBoundaryGrid(void);
00199
00205   bool AssembleOperator(void);
00206
00207   int order_accuracy_;
00208   int dim_null_;
00209   int num_bndy_approxs_;
00210   int num_bndy_coeffs_;
00211   int gradient_length_;
00212   int minrow_;
00213   int row_;
00214
00215   DenseMatrix rat_basis_null_space_;
00216
00217   Real *coeffs_interior_;
00218   Real *prem_apps_;
00219   Real *weights_crs_;
00220   Real *weights_cbs_;
00221   Real *mim_bndy_;
00222   Real *gradient_;
00223
00224   Real mimetic_threshold_;
00225 };
00226 }
00227 #endif  // End of: MTK_INCLUDE_GRAD_1D_H_
```

17.23 include/mtk_grad_2d.h File Reference

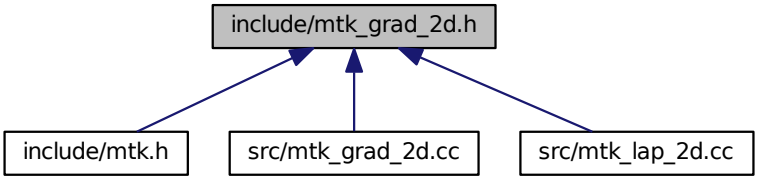
Includes the definition of the class Grad2D.

```
#include "mtk_roots.h"
#include "mtk_dense_matrix.h"
#include "mtk_uni_stg_grid_2d.h"
```

Include dependency graph for mtk_grad_2d.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mtk::Grad2D

  Implements a 2D mimetic gradient operator.

Namespaces

- mtk

Mimetic Methods Toolkit namespace.

17.23.1 Detailed Description

This class implements a 2D gradient operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm (C↩ BSA).

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_grad_2d.h.

17.24 mtk_grad_2d.h

```
00001
00011 /*
00012 Copyright (C) 2015, Computational Science Research Center, San Diego State
00013 University. All rights reserved.
00014
00015 Redistribution and use in source and binary forms, with or without modification,
00016 are permitted provided that the following conditions are met:
00017
00018 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00019 and a copy of the modified files should be reported once modifications are
00020 completed, unless these modifications are made through the project's GitHub
00021 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00022 should be developed and included in any deliverable.
00023
00024 2. Redistributions of source code must be done through direct
00025 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00026
00027 3. Redistributions in binary form must reproduce the above copyright notice,
00028 this list of conditions and the following disclaimer in the documentation and/or
00029 other materials provided with the distribution.
00030
00031 4. Usage of the binary form on proprietary applications shall require explicit
00032 prior written permission from the the copyright holders, and due credit should
00033 be given to the copyright holders.
00034
00035 5. Neither the name of the copyright holder nor the names of its contributors
00036 may be used to endorse or promote products derived from this software without
00037 specific prior written permission.
00038
00039 The copyright holders provide no reassurances that the source code provided does
00040 not infringe any patent, copyright, or any other intellectual property rights of
00041 third parties. The copyright holders disclaim any liability to any recipient for
00042 claims brought against recipient by any third party for infringement of that
00043 parties intellectual property rights.
00044
00045 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00046 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00047 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00048 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00049 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00050 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00051 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00052 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00053 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00054 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00055 */
00056
00057 #ifndef MTK_INCLUDE_MTK_GRAD_2D_H_
00058 #define MTK_INCLUDE_MTK_GRAD_2D_H_
00059
00060 #include "mtk_roots.h"
00061 #include "mtk_dense_matrix.h"
00062 #include "mtk_uni_stg_grid_2d.h"
00063
00064 namespace mtk{
```

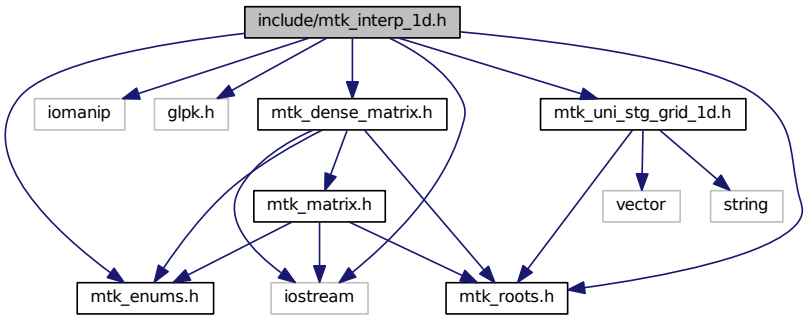
```
00065
00076 class Grad2D {
00077  public:
00079   Grad2D();
00080
00086   Grad2D(const Grad2D &grad);
00087
00089   ~Grad2D();
00090
00096   bool ConstructGrad2D(const UniStgGrid2D &grid,
00097                        int order_accuracy = kDefaultOrderAccuracy,
00098                        Real mimetic_threshold = kDefaultMimeticThreshold);
00099
00105   DenseMatrix ReturnAsDenseMatrix() const;
00106
00107  private:
00108   DenseMatrix gradient_;
00109
00110   int order_accuracy_;
00111
00112   Real mimetic_threshold_;
00113 };
00114 }
00115 #endif  // End of: MTK_INCLUDE_MTK_GRAD_2D_H_
```

17.25 include/mtk_interp_1d.h File Reference

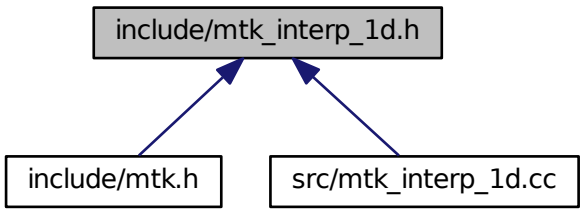
Includes the definition of the class Interp1D.

```
#include <iostream>
#include <iomanip>
#include "glpk.h"
#include "mtk_roots.h"
#include "mtk_enums.h"
#include "mtk_dense_matrix.h"
#include "mtk_uni_stg_grid_1d.h"
```

Include dependency graph for mtk_interp_1d.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mtk::Interp1D

  Implements a 1D interpolation operator.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

17.25.1 Detailed Description

This class implements a 1D interpolation operator.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu
: Johnny Corbino - jcorbino at mail dot sdsu dot edu

Definition in file mtk_interp_1d.h.

17.26 mtk_interp_1d.h

```
00001
00012 /*
00013 Copyright (C) 2015, Computational Science Research Center, San Diego State
00014 University. All rights reserved.
00015
00016 Redistribution and use in source and binary forms, with or without modification,
00017 are permitted provided that the following conditions are met:
00018
00019 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00020 and a copy of the modified files should be reported once modifications are
00021 completed, unless these modifications are made through the project's GitHub
00022 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00023 should be developed and included in any deliverable.
00024
```

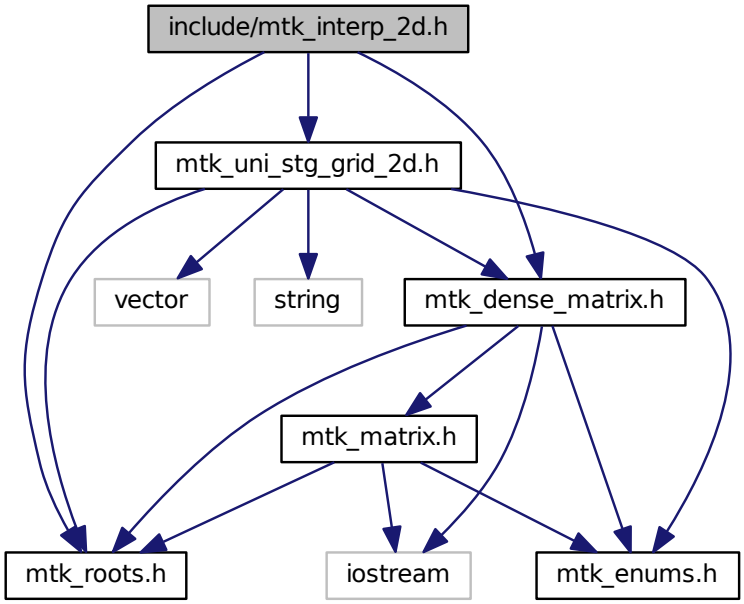
```
00025 2. Redistributions of source code must be done through direct
00026 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00027
00028 3. Redistributions in binary form must reproduce the above copyright notice,
00029 this list of conditions and the following disclaimer in the documentation and/or
00030 other materials provided with the distribution.
00031
00032 4. Usage of the binary form on proprietary applications shall require explicit
00033 prior written permission from the the copyright holders, and due credit should
00034 be given to the copyright holders.
00035
00036 5. Neither the name of the copyright holder nor the names of its contributors
00037 may be used to endorse or promote products derived from this software without
00038 specific prior written permission.
00039
00040 The copyright holders provide no reassurances that the source code provided does
00041 not infringe any patent, copyright, or any other intellectual property rights of
00042 third parties. The copyright holders disclaim any liability to any recipient for
00043 claims brought against recipient by any third party for infringement of that
00044 parties intellectual property rights.
00045
00046 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00047 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00048 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00049 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00050 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00051 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00052 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00053 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00054 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00055 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00056 */
00057
00058 #ifndef MTK_INCLUDE_INTERP_1D_H_
00059 #define MTK_INCLUDE_INTERP_1D_H_
00060
00061 #include <iostream>
00062 #include <iomanip>
00063
00064 #include "glpk.h"
00065
00066 #include "mtk_roots.h"
00067 #include "mtk_enums.h"
00068 #include "mtk_dense_matrix.h"
00069 #include "mtk_uni_stg_grid_1d.h"
00070
00071 namespace mtk {
00072
00082 class Interp1D {
00083  public:
00085   friend std::ostream& operator <<(std::ostream& stream, Interp1D &in);
00086
00088   Interp1D();
00089
00095   Interp1D(const Interp1D &interp);
00096
00098   ~Interp1D();
00099
00105   bool ConstructInterp1D(int order_accuracy =
      kDefaultOrderAccuracy,
00106                          mtk::DirInterp dir = SCALAR_TO_VECTOR);
00107
00113   Real *coeffs_interior() const;
00114
00120   DenseMatrix ReturnAsDenseMatrix(const
      UniStgGrid1D &grid) const;
00121
00122  private:
00123   DirInterp dir_interp_;
00124
00125   int order_accuracy_;
00126
00127   Real *coeffs_interior_;
00128 };
00129 }
00130 #endif  // End of: MTK_INCLUDE_INTERP_1D_H_
```

17.27 include/mtk_interp_2d.h File Reference

Includes the definition of the class Interp2D.

```
#include "mtk_roots.h"
#include "mtk_dense_matrix.h"
#include "mtk_uni_stg_grid_2d.h"
```

Include dependency graph for mtk_interp_2d.h:



Classes

- class mtk::Interp2D

  Implements a 2D interpolation operator.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

17.27.1 Detailed Description

This class implements a 2D interpolation operator.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu
: Johnny Corbino - jcorbino at mail dot sdsu dot edu

Definition in file mtk_interp_2d.h.

17.28 mtk_interp_2d.h

```
00001
00012 /*
00013 Copyright (C) 2015, Computational Science Research Center, San Diego State
00014 University. All rights reserved.
00015
00016 Redistribution and use in source and binary forms, with or without modification,
00017 are permitted provided that the following conditions are met:
00018
00019 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00020 and a copy of the modified files should be reported once modifications are
00021 completed, unless these modifications are made through the project's GitHub
00022 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00023 should be developed and included in any deliverable.
00024
00025 2. Redistributions of source code must be done through direct
00026 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00027
00028 3. Redistributions in binary form must reproduce the above copyright notice,
00029 this list of conditions and the following disclaimer in the documentation and/or
00030 other materials provided with the distribution.
00031
00032 4. Usage of the binary form on proprietary applications shall require explicit
00033 prior written permission from the the copyright holders, and due credit should
00034 be given to the copyright holders.
00035
00036 5. Neither the name of the copyright holder nor the names of its contributors
00037 may be used to endorse or promote products derived from this software without
00038 specific prior written permission.
00039
00040 The copyright holders provide no reassurances that the source code provided does
00041 not infringe any patent, copyright, or any other intellectual property rights of
00042 third parties. The copyright holders disclaim any liability to any recipient for
00043 claims brought against recipient by any third party for infringement of that
00044 parties intellectual property rights.
00045
00046 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00047 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00048 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00049 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00050 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00051 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00052 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00053 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00054 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00055 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00056 */
00057
00058 #ifndef MTK_INCLUDE_MTK_INTERP_2D_H_
00059 #define MTK_INCLUDE_MTK_INTERP_2D_H_
00060
00061 #include "mtk_roots.h"
00062 #include "mtk_dense_matrix.h"
00063 #include "mtk_uni_stg_grid_2d.h"
00064
00065 namespace mtk{
00066
00076 class Interp2D {
00077  public:
00079   Interp2D();
00080
00086   Interp2D(const Interp2D &interp);
00087
00089   ~Interp2D();
00090
00096   DenseMatrix ConstructInterp2D(const UniStgGrid2D &grid,
00097                                 int order_accuracy = kDefaultOrderAccuracy,
```

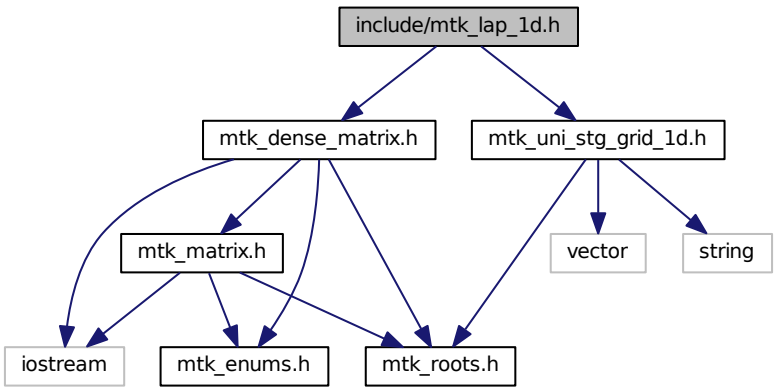
```
00098                             Real mimetic_threshold =
      kDefaultMimeticThreshold);
00099
00105   DenseMatrix ReturnAsDenseMatrix();
00106
00107  private:
00108   DenseMatrix interpolator_;
00109
00110   int order_accuracy_;
00111
00112   Real mimetic_threshold_;
00113 };
00114 }
00115 #endif  // End of: MTK_INCLUDE_MTK_INTERP_2D_H_
```

17.29 include/mtk_lap_1d.h File Reference

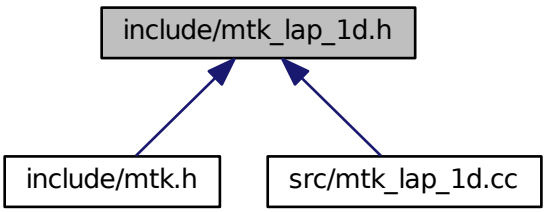
Includes the definition of the class Lap1D.

```
#include "mtk_dense_matrix.h"
#include "mtk_uni_stg_grid_1d.h"
```

Include dependency graph for mtk_lap_1d.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mtk::Lap1D

  Implements a 1D mimetic Laplacian operator.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

17.29.1 Detailed Description

This class implements a 1D Laplacian operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm (CBSA).

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_lap_1d.h.

17.30 mtk_lap_1d.h

```
00001
00011 /*
00012 Copyright (C) 2015, Computational Science Research Center, San Diego State
00013 University. All rights reserved.
00014
00015 Redistribution and use in source and binary forms, with or without modification,
00016 are permitted provided that the following conditions are met:
00017
00018 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00019 and a copy of the modified files should be reported once modifications are
00020 completed, unless these modifications are made through the project's GitHub
00021 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00022 should be developed and included in any deliverable.
00023
```

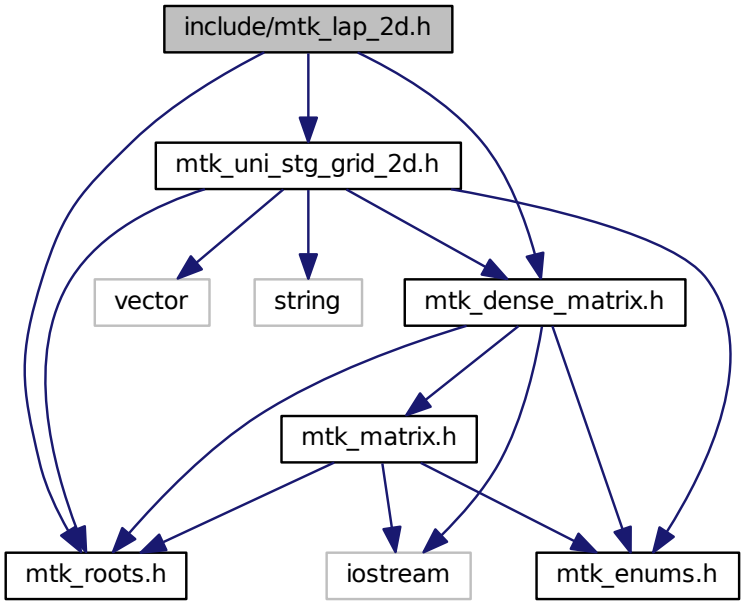
```
00024 2. Redistributions of source code must be done through direct
00025 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00026
00027 3. Redistributions in binary form must reproduce the above copyright notice,
00028 this list of conditions and the following disclaimer in the documentation and/or
00029 other materials provided with the distribution.
00030
00031 4. Usage of the binary form on proprietary applications shall require explicit
00032 prior written permission from the the copyright holders, and due credit should
00033 be given to the copyright holders.
00034
00035 5. Neither the name of the copyright holder nor the names of its contributors
00036 may be used to endorse or promote products derived from this software without
00037 specific prior written permission.
00038
00039 The copyright holders provide no reassurances that the source code provided does
00040 not infringe any patent, copyright, or any other intellectual property rights of
00041 third parties. The copyright holders disclaim any liability to any recipient for
00042 claims brought against recipient by any third party for infringement of that
00043 parties intellectual property rights.
00044
00045 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00046 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00047 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00048 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00049 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00050 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00051 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00052 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00053 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00054 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00055 */
00056
00057 #ifndef MTK_INCLUDE_LAP_1D_H_
00058 #define MTK_INCLUDE_LAP_1D_H_
00059
00060 #include "mtk_dense_matrix.h"
00061
00062 #include "mtk_uni_stg_grid_1d.h"
00063
00064 namespace mtk {
00065
00076 class Lap1D {
00077  public:
00079   friend std::ostream& operator <<(std::ostream& stream, Lap1D &in);
00080
00082   Lap1D();
00083
00089   Lap1D(const Lap1D &lap);
00090
00092   ~Lap1D();
00093
00099   bool ConstructLap1D(int order_accuracy = kDefaultOrderAccuracy,
00100                       Real mimetic_threshold = kDefaultMimeticThreshold);
00101
00107   DenseMatrix ReturnAsDenseMatrix(const
      UniStgGrid1D &grid) const;
00108
00114   const mtk::Real* data(const UniStgGrid1D &grid) const;
00115
00116  private:
00117   int order_accuracy_;
00118   int laplacian_length_;
00119
00120   Real *laplacian_;
00121
00122   Real mimetic_threshold_;
00123 };
00124 }
00125 #endif  // End of: MTK_INCLUDE_LAP_1D_H_
```

17.31 include/mtk_lap_2d.h File Reference

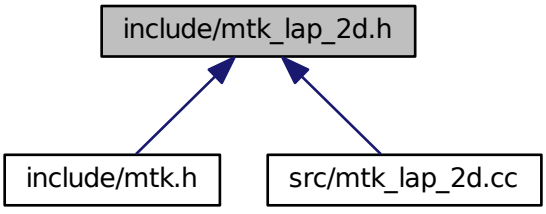
Includes the implementation of the class Lap2D.

```
#include "mtk_roots.h"
#include "mtk_dense_matrix.h"
#include "mtk_uni_stg_grid_2d.h"
```

Include dependency graph for mtk_lap_2d.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mtk::Lap2D

  Implements a 2D mimetic Laplacian operator.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

17.31.1 Detailed Description

This class implements a 2D Laplacian operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm (CBSA).

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_lap_2d.h.

17.32 mtk_lap_2d.h

```
00001
00011 /*
00012 Copyright (C) 2015, Computational Science Research Center, San Diego State
00013 University. All rights reserved.
00014
00015 Redistribution and use in source and binary forms, with or without modification,
00016 are permitted provided that the following conditions are met:
00017
00018 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00019 and a copy of the modified files should be reported once modifications are
00020 completed, unless these modifications are made through the project's GitHub
00021 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00022 should be developed and included in any deliverable.
00023
00024 2. Redistributions of source code must be done through direct
00025 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00026
00027 3. Redistributions in binary form must reproduce the above copyright notice,
00028 this list of conditions and the following disclaimer in the documentation and/or
00029 other materials provided with the distribution.
00030
00031 4. Usage of the binary form on proprietary applications shall require explicit
00032 prior written permission from the the copyright holders, and due credit should
00033 be given to the copyright holders.
00034
00035 5. Neither the name of the copyright holder nor the names of its contributors
00036 may be used to endorse or promote products derived from this software without
00037 specific prior written permission.
00038
00039 The copyright holders provide no reassurances that the source code provided does
00040 not infringe any patent, copyright, or any other intellectual property rights of
00041 third parties. The copyright holders disclaim any liability to any recipient for
00042 claims brought against recipient by any third party for infringement of that
00043 parties intellectual property rights.
00044
00045 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00046 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00047 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00048 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00049 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00050 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00051 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00052 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00053 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00054 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00055 */
00056
00057 #ifndef MTK_INCLUDE_MTK_LAP_2D_H_
00058 #define MTK_INCLUDE_MTK_LAP_2D_H_
00059
00060 #include "mtk_roots.h"
```

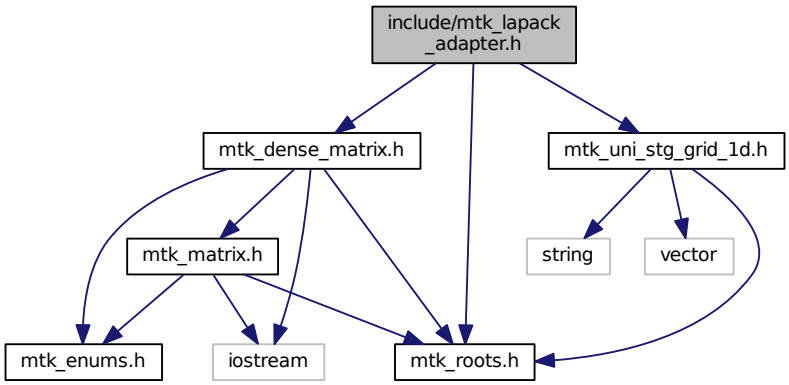
```
00061 #include "mtk_dense_matrix.h"
00062 #include "mtk_uni_stg_grid_2d.h"
00063
00064 namespace mtk{
00065
00076 class Lap2D {
00077  public:
00079   Lap2D();
00080
00086   Lap2D(const Lap2D &lap);
00087
00089   ~Lap2D();
00090
00096   bool ConstructLap2D(const UniStgGrid2D &grid,
00097                       int order_accuracy = kDefaultOrderAccuracy,
00098                       Real mimetic_threshold = kDefaultMimeticThreshold);
00099
00105   DenseMatrix ReturnAsDenseMatrix() const;
00106
00112   Real *data() const;
00113
00114  private:
00115   DenseMatrix laplacian_;
00116
00117   int order_accuracy_;
00118
00119   Real mimetic_threshold_;
00120 };
00121 }
00122 #endif  // End of: MTK_INCLUDE_MTK_LAP_2D_H_
```

17.33 include/mtk_lapack_adapter.h File Reference

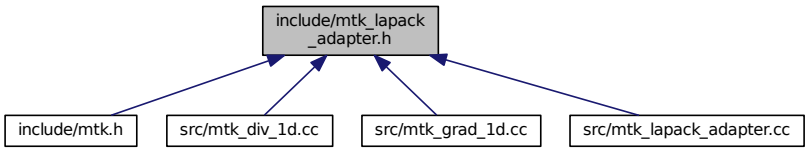
Adapter class for the LAPACK API.

```
#include "mtk_roots.h"
#include "mtk_dense_matrix.h"
#include "mtk_uni_stg_grid_1d.h"
```

Include dependency graph for mtk_lapack_adapter.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mtk::LAPACKAdapter

  Adapter class for the LAPACK API.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

17.33.1 Detailed Description

This class contains a collection of static classes, that posses direct access to the underlying structure of the matrices, thus allowing programmers to exploit some of the numerical methods implemented in the LAPACK.

The **LAPACK (Linear Algebra PACKage)** is written in Fortran 90 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems.

See also

http://www.netlib.org/lapack/

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_lapack_adapter.h.

17.34 mtk_lapack_adapter.h

```
00001
00019 /*
00020 Copyright (C) 2015, Computational Science Research Center, San Diego State
00021 University. All rights reserved.
00022
00023 Redistribution and use in source and binary forms, with or without modification,
00024 are permitted provided that the following conditions are met:
00025
00026 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00027 and a copy of the modified files should be reported once modifications are
```

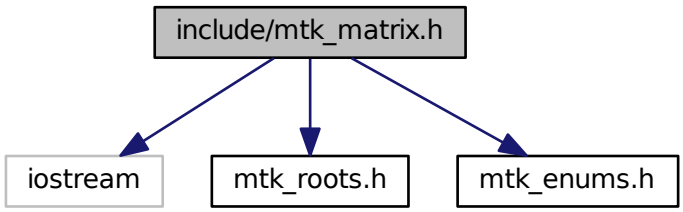
```
00028 completed, unless these modifications are made through the project's GitHub
00029 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00030 should be developed and included in any deliverable.
00031
00032 2. Redistributions of source code must be done through direct
00033 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00034
00035 3. Redistributions in binary form must reproduce the above copyright notice,
00036 this list of conditions and the following disclaimer in the documentation and/or
00037 other materials provided with the distribution.
00038
00039 4. Usage of the binary form on proprietary applications shall require explicit
00040 prior written permission from the the copyright holders, and due credit should
00041 be given to the copyright holders.
00042
00043 5. Neither the name of the copyright holder nor the names of its contributors
00044 may be used to endorse or promote products derived from this software without
00045 specific prior written permission.
00046
00047 The copyright holders provide no reassurances that the source code provided does
00048 not infringe any patent, copyright, or any other intellectual property rights of
00049 third parties. The copyright holders disclaim any liability to any recipient for
00050 claims brought against recipient by any third party for infringement of that
00051 parties intellectual property rights.
00052
00053 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00054 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00055 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00056 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00057 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00058 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00059 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00060 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00061 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00062 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00063 */
00064
00065 #ifndef MTK_INCLUDE_LAPACK_ADAPTER_H_
00066 #define MTK_INCLUDE_LAPACK_ADAPTER_H_
00067
00068 #include "mtk_roots.h"
00069 #include "mtk_dense_matrix.h"
00070 #include "mtk_uni_stg_grid_1d.h"
00071
00072 namespace mtk {
00073
00092 class LAPACKAdapter {
00093  public:
00104   static int SolveDenseSystem(mtk::DenseMatrix &mm,
00105                               mtk::Real *rhs);
00106
00117   static int SolveDenseSystem(mtk::DenseMatrix &mm,
00118                               mtk::DenseMatrix &rr);
00119
00130   static int SolveDenseSystem(mtk::DenseMatrix &mm,
00131                               mtk::UniStgGrid1D &rhs);
00132
00144   static int SolveRectangularDenseSystem(const
      mtk::DenseMatrix &aa,
00145                                          mtk::Real *ob_,
00146                                          int ob_ld_);
00147
00159   static mtk::DenseMatrix QRFactorDenseMatrix(
      DenseMatrix &matrix);
00160 };
00161 }
00162 #endif  // End of: MTK_INCLUDE_LAPACK_ADAPTER_H_
```

17.35 include/mtk_matrix.h File Reference

Definition of the representation of a matrix in the MTK.

```
#include <iostream>
#include "mtk_roots.h"
#include "mtk_enums.h"
```

Include dependency graph for mtk_matrix.h:



This graph shows which files directly or indirectly include this file:

Classes

- class mtk::Matrix

  Definition of the representation of a matrix in the MTK.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

17.35.1 Detailed Description

Definition of the representation for the matrices implemented in the MTK.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_matrix.h.

17.36 mtk_matrix.h

```
00001
00010 /*
```

00055
00056 #ifndef MTK_INCLUDE_MATRIX_H_
00057 #define MTK_INCLUDE_MATRIX_H_
00058
00059 #include <iostream>
00060
00061 #include "mtk_roots.h"
00062 #include "mtk_enums.h"
00063
00064 namespace mtk {
00065
00075 class Matrix {
00076 public:
00078   Matrix();
00079
00085   Matrix(const Matrix &in);
00086
00088   ~Matrix() noexcept ;
00089
00095   MatrixStorage storage() const noexcept;
00096
00102   MatrixOrdering ordering() const noexcept;
00103
00109   int num_rows() const noexcept;
00110
00116   int num_cols() const noexcept;
00117
00123   int num_values() const noexcept;
00124
00134   int ld() const noexcept;
00135
00141   int num_zero() const noexcept;
00142
00148   int num_non_zero() const noexcept;
00149
00157   int num_null() const noexcept;
00158

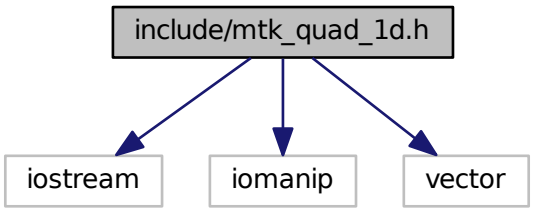
```
00166   int num_non_null() const noexcept;
00167
00173   int kl() const noexcept;
00174
00180   int ku() const noexcept;
00181
00187   int bandwidth() const noexcept;
00188
00196   Real abs_density() const noexcept;
00197
00205   Real rel_density() const noexcept;
00206
00214   Real abs_sparsity() const noexcept;
00215
00223   Real rel_sparsity() const noexcept;
00224
00232   void set_storage(const MatrixStorage &tt) noexcept;
00233
00241   void set_ordering(const MatrixOrdering &oo) noexcept;
00242
00248   void set_num_rows(const int &num_rows) noexcept;
00249
00255   void set_num_cols(const int &num_cols) noexcept;
00256
00262   void set_num_zero(const int &in) noexcept;
00263
00269   void set_num_null(const int &in) noexcept;
00270
00272   void IncreaseNumZero() noexcept;
00273
00275   void IncreaseNumNull() noexcept;
00276
00277  private:
00278   MatrixStorage storage_;
00279
00280   MatrixOrdering ordering_;
00281
00282   int num_rows_;
00283   int num_cols_;
00284   int num_values_;
00285   int ld_;
00286
00287   int num_zero_;
00288   int num_non_zero_;
00289   int num_null_;
00290   int num_non_null_;
00291
00292   int kl_;
00293   int ku_;
00294   int bandwidth_;
00295
00296   Real abs_density_;
00297   Real rel_density_;
00298   Real abs_sparsity_;
00299   Real rel_sparsity_;
00300 };
00301 }
00302 #endif  // End of: MTK_INCLUDE_MATRIX_H_
```

17.37 include/mtk_quad_1d.h File Reference

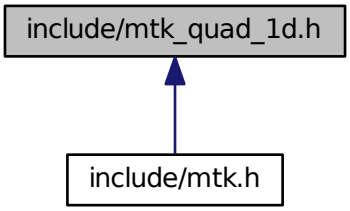
Includes the definition of the class Quad1D.

```
#include <iostream>
#include <iomanip>
#include <vector>
```

Include dependency graph for mtk_quad_1d.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mtk::Quad1D

  Implements a 1D mimetic quadrature.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

17.37.1 Detailed Description

This class implements a 1D quadrature solver based on the mimetic discretization of the gradient operator.

See also

mtk::Grad1D

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Todo Implement this class.

Definition in file mtk_quad_1d.h.

17.38 mtk_quad_1d.h

```
00001
00015 /*
00016 Copyright (C) 2015, Computational Science Research Center, San Diego State
00017 University. All rights reserved.
00018
00019 Redistribution and use in source and binary forms, with or without modification,
00020 are permitted provided that the following conditions are met:
00021
00022 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00023 and a copy of the modified files should be reported once modifications are
00024 completed, unless these modifications are made through the project's GitHub
00025 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00026 should be developed and included in any deliverable.
00027
00028 2. Redistributions of source code must be done through direct
00029 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00030
00031 3. Redistributions in binary form must reproduce the above copyright notice,
00032 this list of conditions and the following disclaimer in the documentation and/or
00033 other materials provided with the distribution.
00034
00035 4. Usage of the binary form on proprietary applications shall require explicit
00036 prior written permission from the the copyright holders, and due credit should
00037 be given to the copyright holders.
00038
00039 5. Neither the name of the copyright holder nor the names of its contributors
00040 may be used to endorse or promote products derived from this software without
00041 specific prior written permission.
00042
00043 The copyright holders provide no reassurances that the source code provided does
00044 not infringe any patent, copyright, or any other intellectual property rights of
00045 third parties. The copyright holders disclaim any liability to any recipient for
00046 claims brought against recipient by any third party for infringement of that
00047 parties intellectual property rights.
00048
00049 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00050 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00051 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00052 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00053 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00054 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00055 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00056 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00057 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00058 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00059 */
00060
00061 #ifndef MTK_INCLUDE_QUAD_1D_H_
00062 #define MTK_INCLUDE_QUAD_1D_H_
00063
00064 #include <iostream>
00065 #include <iomanip>
00066
00067 #include <vector>
00068
00069 namespace mtk {
00070
00081 class Quad1D {
00082  public:
00084   friend std::ostream& operator <<(std::ostream& stream, Quad1D &in);
00085
00087   Quad1D();
00088
00094   Quad1D(const Quad1D &quad);
00095
```

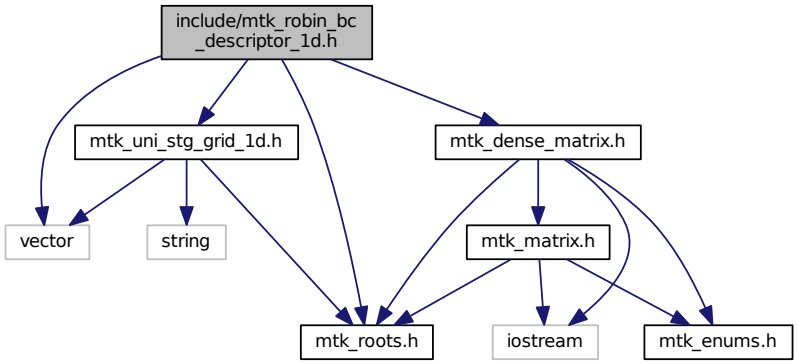
```
00097   ~Quad1D();
00098
00104   int degree_approximation() const;
00105
00111   Real *weights() const;
00112
00121   Real Integrate(Real (*Integrand)(Real xx), UniStgGrid1D grid) const;
00122
00123  private:
00124   int degree_approximation_;
00125
00126   std::vector<Real> weights_;
00127 };
00128 }
00129 #endif  // End of: MTK_INCLUDE_QUAD_1D_H_
```

17.39 include/mtk_robin_bc_descriptor_1d.h File Reference

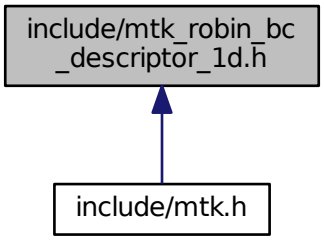
Impose Robin boundary conditions on the operators and on the grids.

```
#include <vector>
#include "mtk_roots.h"
#include "mtk_dense_matrix.h"
#include "mtk_uni_stg_grid_1d.h"
```

Include dependency graph for mtk_robin_bc_descriptor_1d.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mtk::BCDescriptor1D

  Imposes boundary conditions on the operators or on the grids.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

17.39.1 Detailed Description

This class presents an interface for the user to specify Robin boundary conditions on 1D mimetic operators and the grids they are acting on.

Def. Let $u(\mathbf{x},t) : \Omega \times [t_0,t_n] \mapsto \mathbb{R}$ be the solution to an ordinary or partial differential equation of interest. We say that u satisfies a **Robin boundary condition on** $\partial\Omega$ if and only if there exists $\beta(\mathbf{x},t) : \Omega \times [t_0,t_n] \mapsto \mathbb{R}$ so that:

$$\forall t \in [t_0,t_n] \; \forall \mathbf{x} \in \partial\Omega : \delta(\mathbf{x},t)u(\mathbf{x},t) + \eta(\mathbf{x},t)(\hat{\mathbf{n}} \cdot \nabla u) = \beta(\mathbf{x},t).$$

Intuitively, a **Robin boundary condition** is a constraint that must be satisfied by any linear combination of any scalar field u and its first normal derivative, in order for u to represent a unique solution to a given ordinary or partial differential equation of interest.

In a 1D context ($\partial\Omega = \{a,b\} \subset \mathbb{R}$), this condition can be written as follows:

$$\delta_a(a,t)u(a,t) - \eta_a(a,t)u'(a,t) = \beta_a(a,t),$$

$$\delta_b(b,t)u(b,t) + \eta_b(b,t)u'(b,t) = \beta_b(b,t).$$

Instances of this class receive information about the coefficient functions and each condition for any subset of the boundary (west and east, in 1D). These instances then handle the complexity of placing the coefficients in the differentiation matrices and condition in the grids.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_robin_bc_descriptor_1d.h.

17.40 mtk_robin_bc_descriptor_1d.h

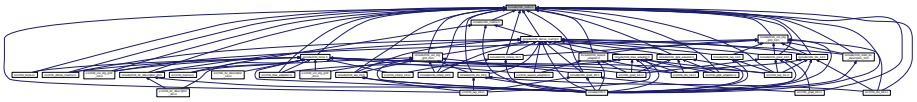
```
00001
00041 /*
00042 Copyright (C) 2015, Computational Science Research Center, San Diego State
00043 University. All rights reserved.
00044
00045 Redistribution and use in source and binary forms, with or without modification,
00046 are permitted provided that the following conditions are met:
00047
00048 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00049 and a copy of the modified files should be reported once modifications are
00050 completed, unless these modifications are made through the project's GitHub
00051 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00052 should be developed and included in any deliverable.
00053
00054 2. Redistributions of source code must be done through direct
00055 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00056
00057 3. Redistributions in binary form must reproduce the above copyright notice,
00058 this list of conditions and the following disclaimer in the documentation and/or
00059 other materials provided with the distribution.
00060
00061 4. Usage of the binary form on proprietary applications shall require explicit
00062 prior written permission from the the copyright holders, and due credit should
00063 be given to the copyright holders.
00064
00065 5. Neither the name of the copyright holder nor the names of its contributors
00066 may be used to endorse or promote products derived from this software without
00067 specific prior written permission.
00068
00069 The copyright holders provide no reassurances that the source code provided does
00070 not infringe any patent, copyright, or any other intellectual property rights of
00071 third parties. The copyright holders disclaim any liability to any recipient for
00072 claims brought against recipient by any third party for infringement of that
00073 parties intellectual property rights.
00074
00075 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00076 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00077 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00078 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00079 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00080 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00081 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00082 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00083 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00084 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00085 */
00086
00087 #include <vector>
00088
00089 #include "mtk_roots.h"
00090 #include "mtk_dense_matrix.h"
00091 #include "mtk_uni_stg_grid_1d.h"
00092
00093 #ifndef MTK_INCLUDE_BC_DESCRIPTOR_1D_H_
00094 #define MTK_INCLUDE_BC_DESCRIPTOR_1D_H_
00095
00096 namespace mtk {
00097
00128 class BCDescriptor1D {
00129  public:
00137   static void ImposeOnLaplacianMatrix(DenseMatrix &matrix,
00138                                       const std::vector<Real> &west,
00139                                       const std::vector<Real> &east);
00140
00148   static void ImposeOnGrid(UniStgGrid1D &grid,
00149                            const Real &epsilon,
00150                            const Real &omega);
00151
```

```
00152
00153   int highest_order_diff_west_;
00154   int highest_order_diff_east_;
00155
00156   std::vector<CoefficientFunction2D> west_coefficients_;
00157   std::vector<CoefficientFunction2D> east_coefficients_;
00158
00159   Real (*west_condition_)(Real xx, Real yy);
00160   Real (*east_condition_)(Real xx, Real yy);
00161 };
00162 }
00163 #endif  // End of: MTK_INCLUDE_BC_DESCRIPTOR_1D_H_
```

17.41 include/mtk_roots.h File Reference

Fundamental definitions to be used across all classes of the MTK.

This graph shows which files directly or indirectly include this file:



Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

Typedefs

- typedef float mtk::Real

  Users can simply change this to build a double- or single-precision MTK.

Variables

- const float mtk::kZero {0.0f}

  MTK's zero defined according to selective compilation.
- const float mtk::kOne {1.0f}

  MTK's one defined according to selective compilation.
- const float mtk::kTwo {2.0f}

  MTK's two defined according to selective compilation.
- const float mtk::kDefaultTolerance {1e-7f}

  Considered tolerance for comparisons in numerical methods.
- const int mtk::kDefaultOrderAccuracy {2}

  Default order of accuracy for mimetic operators.
- const float mtk::kDefaultMimeticThreshold {1e-6f}

  Default tolerance for higher-order mimetic operators.
- const int mtk::kCriticalOrderAccuracyDiv {8}

  At this order (and higher) we must use the CBSA to construct.
- const int mtk::kCriticalOrderAccuracyGrad {10}

  At this order (and higher) we must use the CBSA to construct.

17.41.1 Detailed Description

This file contains the fundamental definitions that classes of the MTK rely on to be implemented. Examples of these definitions are the definition of fundamental data types, and global variables affecting the construction of mimetic operators, among others.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at sciences dot sdsu dot edu

Todo Documentation should (better?) capture effects from selective compilation.

Todo Test selective precision mechanisms.

Definition in file mtk_roots.h.

17.42 mtk_roots.h

```
00001
00017 /*
00018 Copyright (C) 2015, Computational Science Research Center, San Diego State
00019 University. All rights reserved.
00020
00021 Redistribution and use in source and binary forms, with or without modification,
00022 are permitted provided that the following conditions are met:
00023
00024 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00025 and a copy of the modified files should be reported once modifications are
00026 completed, unless these modifications are made through the project's GitHub
00027 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00028 should be developed and included in any deliverable.
00029
00030 2. Redistributions of source code must be done through direct
00031 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00032
00033 3. Redistributions in binary form must reproduce the above copyright notice,
00034 this list of conditions and the following disclaimer in the documentation and/or
00035 other materials provided with the distribution.
00036
00037 4. Usage of the binary form on proprietary applications shall require explicit
00038 prior written permission from the the copyright holders, and due credit should
00039 be given to the copyright holders.
00040
00041 5. Neither the name of the copyright holder nor the names of its contributors
00042 may be used to endorse or promote products derived from this software without
00043 specific prior written permission.
00044
00045 The copyright holders provide no reassurances that the source code provided does
00046 not infringe any patent, copyright, or any other intellectual property rights of
00047 third parties. The copyright holders disclaim any liability to any recipient for
00048 claims brought against recipient by any third party for infringement of that
00049 parties intellectual property rights.
00050
00051 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00052 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00053 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00054 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00055 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00056 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00057 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00058 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00059 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00060 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00061 */
00062
00063 #ifndef MTK_INCLUDE_ROOTS_H_
00064 #define MTK_INCLUDE_ROOTS_H_
00065
00071 namespace mtk {
```

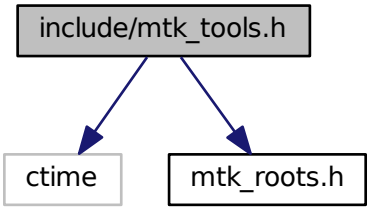
```
00072
00080 #ifdef MTK_PRECISION_DOUBLE
00081 typedef double Real;
00082 #else
00083 typedef float Real;
00084 #endif
00085
00111 #ifdef MTK_PRECISION_DOUBLE
00112 const double kZero{0.0};
00113 const double kOne{1.0};
00114 const double kTwo{2.0};
00115 #else
00116 const float kZero{0.0f};
00117 const float kOne{1.0f};
00118 const float kTwo{2.0f};
00119 #endif
00120
00128 #ifdef MTK_PRECISION_DOUBLE
00129 const double kDefaultTolerance{1e-7};
00130 #else
00131 const float kDefaultTolerance{1e-7f};
00132 #endif
00133
00143 const int kDefaultOrderAccuracy{2};
00144
00154 #ifdef MTK_PRECISION_DOUBLE
00155 const double kDefaultMimeticThreshold{1e-6};
00156 #else
00157 const float kDefaultMimeticThreshold{1e-6f};
00158 #endif
00159
00167 const int kCriticalOrderAccuracyDiv{8};
00168
00176 const int kCriticalOrderAccuracyGrad{10};
00177 }
00178 #endif  // End of: MTK_INCLUDE_ROOTS_H_
```

17.43 include/mtk_tools.h File Reference

Tool manager class.

```
#include <ctime>
#include "mtk_roots.h"
```

Include dependency graph for mtk_tools.h:



This graph shows which files directly or indirectly include this file:

Classes

- class mtk::Tools

  Tool manager class.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

17.43.1 Detailed Description

Basic utilities.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Note

Performance Tip 8.1. If they do not need to be modified by the called function, pass large objects using pointers to constant data or references to constant data, to obtain the performance benefits of pass-by-reference.

Definition in file mtk_tools.h.

17.44 mtk_tools.h

```
00001
00014 /*
00015 Copyright (C) 2015, Computational Science Research Center, San Diego State
00016 University. All rights reserved.
00017
00018 Redistribution and use in source and binary forms, with or without modification,
00019 are permitted provided that the following conditions are met:
00020
00021 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00022 and a copy of the modified files should be reported once modifications are
00023 completed, unless these modifications are made through the project's GitHub
00024 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00025 should be developed and included in any deliverable.
00026
00027 2. Redistributions of source code must be done through direct
00028 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00029
00030 3. Redistributions in binary form must reproduce the above copyright notice,
00031 this list of conditions and the following disclaimer in the documentation and/or
00032 other materials provided with the distribution.
00033
00034 4. Usage of the binary form on proprietary applications shall require explicit
00035 prior written permission from the the copyright holders, and due credit should
00036 be given to the copyright holders.
```

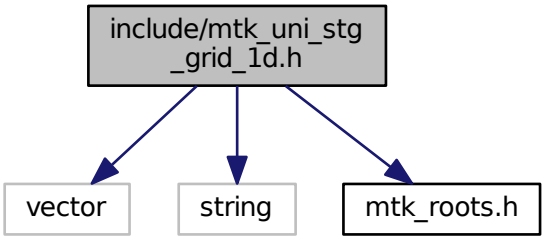
```
00037
00038 5. Neither the name of the copyright holder nor the names of its contributors
00039 may be used to endorse or promote products derived from this software without
00040 specific prior written permission.
00041
00042 The copyright holders provide no reassurances that the source code provided does
00043 not infringe any patent, copyright, or any other intellectual property rights of
00044 third parties. The copyright holders disclaim any liability to any recipient for
00045 claims brought against recipient by any third party for infringement of that
00046 parties intellectual property rights.
00047
00048 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00049 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00050 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00051 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00052 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00053 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00054 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00055 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00056 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00057 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00058 */
00059
00060 #ifndef MTK_INCLUDE_TOOLS_H_
00061 #define MTK_INCLUDE_TOOLS_H_
00062
00063 #include <ctime>
00064
00065 #include "mtk_roots.h"
00066
00067 namespace mtk {
00068
00078 class Tools {
00079  public:
00090   static void Prevent(const bool complement,
00091                       const char *const fname,
00092                       int lineno,
00093                       const char *const fxname) noexcept;
00094
00100   static void BeginUnitTestNo(const int &nn) noexcept;
00101
00107   static void EndUnitTestNo(const int &nn) noexcept;
00108
00114   static void Assert(const bool &condition) noexcept;
00115
00116  private:
00117   static int test_number_;
00118
00119   static Real duration_;
00120
00121   static clock_t begin_time_;
00122 };
00123 }
00124 #endif  // End of: MTK_INCLUDE_TOOLS_H_
```

17.45 include/mtk_uni_stg_grid_1d.h File Reference

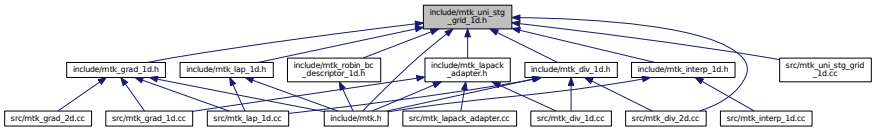
Definition of an 1D uniform staggered grid.

```
#include <vector>
#include <string>
#include "mtk_roots.h"
```

Include dependency graph for mtk_uni_stg_grid_1d.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mtk::UniStgGrid1D

  Uniform 1D Staggered Grid.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

17.45.1 Detailed Description

Definition of an 1D uniform staggered grid.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Todo Create overloaded binding routines that read data from files.

Definition in file mtk_uni_stg_grid_1d.h.

17.46 mtk_uni_stg_grid_1d.h

```
00001
00012 /*
00013 Copyright (C) 2015, Computational Science Research Center, San Diego State
00014 University. All rights reserved.
00015
00016 Redistribution and use in source and binary forms, with or without modification,
00017 are permitted provided that the following conditions are met:
00018
00019 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00020 and a copy of the modified files should be reported once modifications are
00021 completed, unless these modifications are made through the project's GitHub
00022 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00023 should be developed and included in any deliverable.
00024
00025 2. Redistributions of source code must be done through direct
00026 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00027
00028 3. Redistributions in binary form must reproduce the above copyright notice,
00029 this list of conditions and the following disclaimer in the documentation and/or
00030 other materials provided with the distribution.
00031
00032 4. Usage of the binary form on proprietary applications shall require explicit
00033 prior written permission from the the copyright holders, and due credit should
00034 be given to the copyright holders.
00035
00036 5. Neither the name of the copyright holder nor the names of its contributors
00037 may be used to endorse or promote products derived from this software without
00038 specific prior written permission.
00039
00040 The copyright holders provide no reassurances that the source code provided does
00041 not infringe any patent, copyright, or any other intellectual property rights of
00042 third parties. The copyright holders disclaim any liability to any recipient for
00043 claims brought against recipient by any third party for infringement of that
00044 parties intellectual property rights.
00045
00046 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00047 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00048 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00049 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00050 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00051 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00052 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00053 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00054 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00055 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00056 */
00057
00058 #ifndef MTK_INCLUDE_UNI_STG_GRID_1D_H_
00059 #define MTK_INCLUDE_UNI_STG_GRID_1D_H_
00060
00061 #include <vector>
00062 #include <string>
00063
00064 #include "mtk_roots.h"
00065
00066 namespace mtk {
00067
00077 class UniStgGrid1D {
00078  public:
00080   friend std::ostream& operator <<(std::ostream& stream, UniStgGrid1D &in);
00081
00083   UniStgGrid1D();
00084
00090   UniStgGrid1D(const UniStgGrid1D &grid);
00091
00102   UniStgGrid1D(const Real &west_bndy_x,
00103                const Real &east_bndy_x,
00104                const int &num_cells_x,
00105                const mtk::FieldNature &nature = mtk::SCALAR);
00106
00108   ~UniStgGrid1D();
00109
00115   Real west_bndy_x() const;
00116
00122   Real east_bndy_x() const;
00123
00129   Real delta_x() const;
00130
```

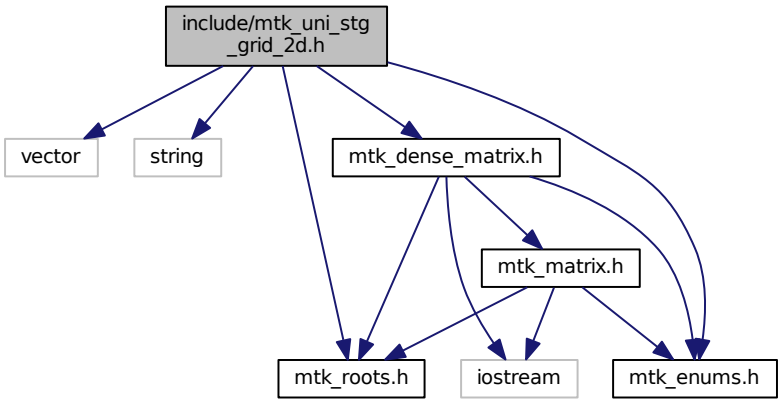
```
00138   const Real *discrete_domain_x() const;
00139
00147   Real *discrete_field_u();
00148
00154   int num_cells_x() const;
00155
00161   void BindScalarField(Real (*ScalarField)(Real xx));
00162
00174   void BindVectorField(Real (*VectorField)(Real xx));
00175
00187   bool WriteToFile(std::string filename,
00188                    std::string space_name,
00189                    std::string field_name) const;
00190
00191  private:
00192   FieldNature nature_;
00193
00194   std::vector<Real> discrete_domain_x_;
00195   std::vector<Real> discrete_field_u_;
00196
00197   Real west_bndy_x_;
00198   Real east_bndy_x_;
00199   Real num_cells_x_;
00200   Real delta_x_;
00201 };
00202 }
00203 #endif  // End of: MTK_INCLUDE_UNI_STG_GRID_1D_H_
```

17.47 include/mtk_uni_stg_grid_2d.h File Reference

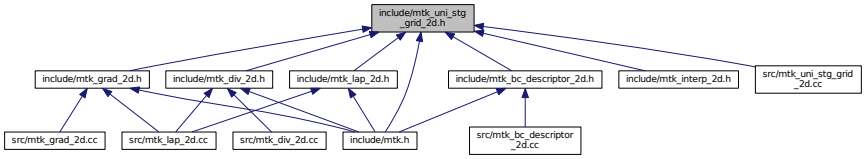
Definition of an 2D uniform staggered grid.

```
#include <vector>
#include <string>
#include "mtk_roots.h"
#include "mtk_enums.h"
#include "mtk_dense_matrix.h"
```

Include dependency graph for mtk_uni_stg_grid_2d.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mtk::UniStgGrid2D

  Uniform 2D Staggered Grid.

Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

17.47.1 Detailed Description

Definition of an 2D uniform staggered grid.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Todo Create overloaded binding routines that read data from files.

Definition in file mtk_uni_stg_grid_2d.h.

17.48 mtk_uni_stg_grid_2d.h

```
00001
00012 /*
00013 Copyright (C) 2015, Computational Science Research Center, San Diego State
00014 University. All rights reserved.
00015
00016 Redistribution and use in source and binary forms, with or without modification,
00017 are permitted provided that the following conditions are met:
00018
00019 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00020 and a copy of the modified files should be reported once modifications are
00021 completed, unless these modifications are made through the project's GitHub
00022 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00023 should be developed and included in any deliverable.
00024
00025 2. Redistributions of source code must be done through direct
00026 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00027
00028 3. Redistributions in binary form must reproduce the above copyright notice,
00029 this list of conditions and the following disclaimer in the documentation and/or
00030 other materials provided with the distribution.
00031
00032 4. Usage of the binary form on proprietary applications shall require explicit
```

```
00033 prior written permission from the the copyright holders, and due credit should
00034 be given to the copyright holders.
00035
00036 5. Neither the name of the copyright holder nor the names of its contributors
00037 may be used to endorse or promote products derived from this software without
00038 specific prior written permission.
00039
00040 The copyright holders provide no reassurances that the source code provided does
00041 not infringe any patent, copyright, or any other intellectual property rights of
00042 third parties. The copyright holders disclaim any liability to any recipient for
00043 claims brought against recipient by any third party for infringement of that
00044 parties intellectual property rights.
00045
00046 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00047 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00048 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00049 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00050 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00051 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00052 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00053 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00054 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00055 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00056 */
00057
00058 #ifndef MTK_INCLUDE_UNI_STG_GRID_2D_H_
00059 #define MTK_INCLUDE_UNI_STG_GRID_2D_H_
00060
00061 #include <vector>
00062 #include <string>
00063
00064 #include "mtk_roots.h"
00065 #include "mtk_enums.h"
00066 #include "mtk_dense_matrix.h"
00067
00068 namespace mtk {
00069
00079 class UniStgGrid2D {
00080  public:
00082   friend std::ostream& operator <<(std::ostream& stream, UniStgGrid2D &in);
00083
00085   UniStgGrid2D();
00086
00092   UniStgGrid2D(const UniStgGrid2D &grid);
00093
00107   UniStgGrid2D(const Real &west_bndy_x,
00108                const Real &east_bndy_x,
00109                const int &num_cells_x,
00110                const Real &south_bndy_y,
00111                const Real &north_bndy_y,
00112                const int &num_cells_y,
00113                const mtk::FieldNature &nature =
     mtk::SCALAR);
00114
00116   ~UniStgGrid2D();
00117
00125   const Real *discrete_domain_x() const;
00126
00134   const Real *discrete_domain_y() const;
00135
00141   Real *discrete_field();
00142
00150   FieldNature nature() const;
00151
00157   Real west_bndy() const;
00158
00164   Real east_bndy() const;
00165
00171   int num_cells_x() const;
00172
00178   Real delta_x() const;
00179
00185   Real south_bndy() const;
00186
00192   Real north_bndy() const;
00193
00199   int num_cells_y() const;
00200
00206   Real delta_y() const;
00207
00213   bool Bound() const;
```

```
00214
00220   void BindScalarField(Real (*ScalarField)(Real xx, Real yy));
00221
00236   void BindVectorField(Real (*VectorFieldPComponent)(Real xx,
    Real yy),
00237                        Real (*VectorFieldQComponent)(Real xx,Real yy));
00238
00251   bool WriteToFile(std::string filename,
00252                    std::string space_name_x,
00253                    std::string space_name_y,
00254                    std::string field_name) const;
00255
00256  private:
00269   void BindVectorFieldPComponent(
00270     Real (*VectorFieldPComponent)(Real xx, Real yy));
00271
00284   void BindVectorFieldQComponent(
00285     Real (*VectorFieldQComponent)(Real xx, Real yy));
00286
00287   std::vector<Real> discrete_domain_x_;
00288   std::vector<Real> discrete_domain_y_;
00289   std::vector<Real> discrete_field_;
00290
00291   FieldNature nature_;
00292
00293   Real west_bndy_;
00294   Real east_bndy_;
00295   int num_cells_x_;
00296   Real delta_x_;
00297
00298   Real south_bndy_;
00299   Real north_bndy_;
00300   int num_cells_y_;
00301   Real delta_y_;
00302 };
00303 }
00304 #endif  // End of: MTK_INCLUDE_UNI_STG_GRID_2D_H_
```

17.49 Makefile.inc File Reference

17.50 Makefile.inc

```
00001 # Makefile setup file for MTK.
00002
00003 SHELL := /bin/bash
00004
00005 # Please set the following variables up:
00006
00007 #   1. Absolute path to base directory of the MTK.
00008 #   ____________________________________________________________
00009
00010 BASE = /home/esanchez/Dropbox/MTK
00011
00012 #   2. The machine (platform) identifier and required machine precision.
00013 #   ____________________________________________________________
00014
00015 # Options are:
00016 # - LINUX: A LINUX box installation.
00017 # - OSX: Uses OS X optimized solvers.
00018
00019 PLAT = LINUX
00020
00021 # Options are:
00022 # - SINGLE: Use 4 B floating point numbers.
00023 # - DOUBLE: Use 8 B floating point numbers.
00024
00025 PRECISION = DOUBLE
00026
00027 #   3. Optimized solvers and operations by means of ATLAS in Linux?
00028 #   ____________________________________________________________
00029
00030 # If you have selected OSX in step 1, then you don't need to worry about this.
00031
00032 # Options are ON xor OFF:
00033
```

```
00034 ATL_OPT = OFF
00035
00036 #   4. Paths to dependencies (header files for compiling).
00037 #   ____________________________________________________________________
00038
00039 # GLPK include path (soon to go):
00040
00041 GLPK_INC = $(HOME)/Libraries/glpk-4.35/include
00042
00043 # Linux: If ATLAS optimization is ON, users should only provide the path to
00044 # ATLAS:
00045
00046 ATLAS_INC = $(HOME)/Libraries/ATLAS_3.8.4-CORE/include
00047
00048 # OS X: Do nothing.
00049
00050 #   5. Paths to dependencies (archive files for (static) linking).
00051 #   ____________________________________________________________________
00052
00053 # GLPK linking path (soon to go):
00054
00055 GLPK_LIB = $(HOME)/Libraries/glpk-4.35/lib/lib64/libglpk.a
00056
00057 # If optimization is OFF, then provide the paths for:
00058
00059 BLAS_LIB = $(HOME)/Libraries/BLAS-3.5.0/libblas.a
00060 LAPACK_LIB = $(HOME)/Libraries/lapack-3.5.0/liblapack.a
00061
00062 # WARNING: Vendor libraries should be used whenever they are available.
00063
00064 # However, if optimization is ON, please provide the path the ATLAS' archive:
00065
00066 ATLAS_LIB   = $(HOME)/Libraries/ATLAS_3.8.4-CORE/ATLAS_3.8.4-BUILD-Citadel/lib
00067
00068 #   6. Compiler and its flags.
00069 #   ____________________________________________________________________
00070
00071 CC = g++
00072
00073 # Debug Level. Options are:
00074 # 0. NO debug at all NOR any run-time checks... be cautious!
00075 # 1. Verbose (execution messages) AND run-time checks.
00076 # 2. Level 1 plus intermediate scalar-valued results.
00077 # 3. Level 2 plus intermediate array-valued results.
00078
00079 DEBUG_LEVEL = 3
00080
00081 # Flags recommended for release code:
00082
00083 CCFLAGS = -Wall -Werror -O3
00084
00085 # Flags recommended for debugging code:
00086
00087 CCFLAGS = -Wall -Werror -g
00088
00089 #   7. Archiver, its flags, and ranlib:
00090 #   ____________________________________________________________________
00091
00092 ARCH      = ar
00093 ARCHFLAGS = cr
00094
00095 # If your system does not have "ranlib" then set: "RANLIB = echo":
00096
00097 RANLIB = echo
00098
00099 # But, if possible:
00100
00101 RANLIB = ranlib
00102
00103 #   8. Valgrind's memcheck options (optional):
00104 #   ____________________________________________________________________
00105
00106 MEMCHECK_OPTS = -v --tool=memcheck --leak-check=full --show-leak-kinds=all \
00107 --track-origins=yes --freelist-vol=20000000
00108
00109 # Done! User, please, do not mess with the definitions from this point on.
00110
00111 #   ____________________________________________________________________
00112 #   ____________________________________________________________________
00113 #   ____________________________________________________________________
00114
```

```
00115 #   MTK-related.
00116 #   ____________________________________________________________________
00117
00118 SRC       = $(BASE)/src
00119 INCLUDE   = $(BASE)/include
00120 LIB       = $(BASE)/lib
00121 MTK_LIB   = $(LIB)/libmtk.a
00122 TESTS     = $(BASE)/tests
00123 EXAMPLES  = $(BASE)/examples
00124
00125 #   Compiling-related.
00126 #   ____________________________________________________________________
00127
00128 CCFLAGS += -std=c++11 -fPIC -DMTK_DEBUG_LEVEL=$(DEBUG_LEVEL) -I$(INCLUDE)  -c
00129
00130 ifeq ($(PRECISION),DOUBLE)
00131   CCFLAGS += -DMTK_PRECISION_DOUBLE
00132 else
00133   CCFLAGS += -DMTK_PRECISION_SINGLE
00134 endif
00135
00136 # Only the GLPK is included because the other dependencies are coded in Fortran.
00137
00138 ifeq ($(ATL_OPT),ON)
00139   CCFLAGS  += -I$(GLPK_INC) $(ATLAS_INC)
00140 else
00141   CCFLAGS  += -I$(GLPK_INC)
00142 endif
00143
00144 #   Linking-related.
00145 #   ____________________________________________________________________
00146
00147 NOOPT_LIBS  = $(LAPACK_LIB) $(BLAS_LIB) -lm $(GLPK_LIB) -lstdc++
00148
00149 OPT_LIBS    = -L$(ATLAS_LIB) -latlas -llapack -lblas -lm -latlas -lstdc++
00150
00151 ifeq ($(PLAT),OSX)
00152   LINKER  = g++
00153   LINKER  += -framework Accelerate $(GLPK_LIB) $(MTK_LIB)
00154 else
00155   ifeq ($(ATL_OPT),ON)
00156     LINKER  = g++
00157     LIBS = $(MTK_LIB)
00158     LIBS += $(OPT_LIBS)
00159   else
00160     LINKER  = gfortran
00161     LIBS = $(MTK_LIB)
00162     LIBS += $(NOOPT_LIBS)
00163   endif
00164 endif
00165
00166 #   Documentation-related.
00167 #   ____________________________________________________________________
00168
00169 DOCGEN      = doxygen
00170 DOCFILENAME = doc_config.dxcf
00171 DOC         = $(BASE)/doc
00172 DOCFILE     = $(BASE)/$(DOCFILENAME)
```

17.51 README.md File Reference

17.52 README.md

```
00001 # The Mimetic Methods Toolkit (MTK)
00002
00003 By: **Eduardo J. Sanchez, Ph.D. - esanchez at mail dot sdsu dot edu**
00004     ____________________________________________________________________
00005
00006 ## 1. Description
00007
00008 We define numerical methods that are based on discretizations preserving the
00009 properties of their continuum counterparts to be **mimetic**.
00010
00011 The **Mimetic Methods Toolkit (MTK)** is a C++ library for mimetic numerical
00012 methods. It is arranged as a set of classes for **mimetic quadratures**,
```

```
00013 **mimetic interpolation**, and **mimetic finite differences** methods for the
00014 numerical solution of ordinary and partial differential equations.
00015
00016 An older version of this library is available outside of GitHub... just email me
00017 about it, and you can have it... it is ugly, yet it is functional and more
00018 complete.
00019     ____________________________________________________________________
00020
00021 ## 2. Dependencies
00022
00023 This README assumes all of these dependencies are installed in the following
00024 folder:
00025
00026 ```
00027 $(HOME)/Libraries/
00028 ```
00029
00030 In this version, the MTK optionally uses ATLAS-optimized BLAS and LAPACK
00031 routines for the internal computation on some of the layers. However, ATLAS
00032 requires both BLAS and LAPACK in order to create their optimized distributions.
00033 Therefore, the following dependencies tree arises:
00034
00035 ### For Linux:
00036
00037 1. LAPACK - Available from: http://www.netlib.org/lapack/
00038   1. BLAS - Available from: http://www.netlib.org/blas/
00039
00040 2. GLPK - Available from: https://www.gnu.org/software/glpk/
00041
00042 3. (Optional) ATLAS - Available from: http://math-atlas.sourceforge.net/
00043   1. LAPACK - Available from: http://www.netlib.org/lapack/
00044     1. BLAS - Available from: http://www.netlib.org/blas
00045
00046 4. (Optional) Valgrind - Available from: http://valgrind.org/
00047
00048 5. (Optional) Doxygen - Available from http://www.stack.nl/~dimitri/doxygen/
00049
00050 ### For OS X:
00051
00052 1. GLPK - Available from: https://www.gnu.org/software/glpk/
00053     ____________________________________________________________________
00054
00055 ## 3. Installation
00056
00057 ### PART 1. CONFIGURATION OF THE MAKEFILE.
00058
00059 The following steps are required to build and test the MTK. Please use the
00060 accompanying ‘Makefile.inc‘ file, which should provide a solid template to
00061 start with. The following command provides help on the options for make:
00062
00063 ```
00064 $ make help
00065 -----
00066 Makefile for the MTK.
00067
00068 Options are:
00069 - all: builds the library, the tests, and examples.
00070 - mtklib: builds the library.
00071 - test: builds the test files.
00072 - example: builds the examples.
00073
00074 - testall: runs all the tests.
00075
00076 - gendoc: generates the documentation for the library.
00077
00078 - clean: cleans all the generated files.
00079 - cleanlib: cleans the generated archive and object files.
00080 - cleantest: cleans the generated tests executables.
00081 - cleanexample: cleans the generated examples executables.
00082 -----
00083 ```
00084
00085 ### PART 2. BUILD THE LIBRARY.
00086
00087 ```
00088 $ make
00089 ```
00090
00091 If successful you’ll read (before building the tests and examples):
00092
00093 ```
```

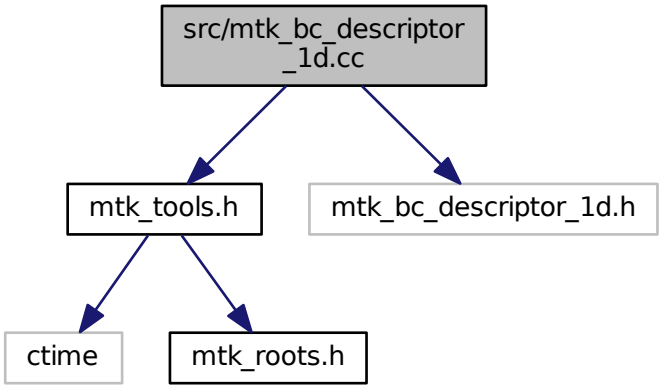
```
00094 ----- Library created! Check in /home/ejspeiro/Dropbox/MTK/lib
00095 ```
00096
00097 Examples and tests will also be built.
00098     ____________________________________________________________
00099
00100 ## 4. Frequently Asked Questions
00101
00102 Q: Why haven't you guys implemented GBS to build the library?
00103 A: I'm on it as we speak! ;)
00104
00105 Q: Is there any main reference when it comes to the theory on Mimetic Methods?
00106 A: Yes! Check: http://www.csrc.sdsu.edu/mimetic-book
00107
00108 Q: Do I need to generate the documentation myself?
00109 A: You can if you want to... but if you DO NOT want to, just go to our website.
00110     ____________________________________________________________
00111
00112 ## 5. Contact, Support, and Credits
00113
00114 The MTK is developed by researchers and adjuncts to the
00115 [Computational Science Research Center (CSRC)](http://www.csrc.sdsu.edu/)
00116 at [San Diego State University (SDSU)](http://www.sdsu.edu/).
00117
00118 Developers are members of:
00119
00120 1. Mimetic Numerical Methods Research and Development Group.
00121 2. Computational Geoscience Research and Development Group.
00122 3. Ocean Modeling Research and Development Group.
00123
00124 Currently the developers are:
00125
00126 - **Eduardo J. Sanchez, Ph.D. - esanchez at mail dot sdsu dot edu** - @ejspeiro
00127 - Jose E. Castillo, Ph.D. - jcastillo at mail dot sdsu dot edu
00128 - Guillermo F. Miranda, Ph.D. - unigrav at hotmail dot com
00129 - Christopher P. Paolini, Ph.D. - paolini at engineering dot sdsu dot edu
00130 - Angel Boada.
00131 - Johnny Corbino.
00132 - Raul Vargas-Navarro.
00133
00134 Finally, please feel free to contact me with suggestions or corrections:
00135
00136 **Eduardo J. Sanchez, Ph.D. - esanchez at mail dot sdsu dot edu** - @ejspeiro
00137
00138 Thanks and happy coding!
```

17.53 src/mtk_bc_descriptor_1d.cc File Reference

Enforces boundary conditions in either the operator or the grid.

```
#include "mtk_tools.h"
#include "mtk_bc_descriptor_1d.h"
```

Include dependency graph for mtk_bc_descriptor_1d.cc:



17.53.1 Detailed Description

This class presents an interface for the user to specify boundary conditions on 1D mimetic operators and the grids they are acting on.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_bc_descriptor_1d.cc.

17.54 mtk_bc_descriptor_1d.cc

```
00001
00011 /*
00012 Copyright (C) 2015, Computational Science Research Center, San Diego State
00013 University. All rights reserved.
00014
00015 Redistribution and use in source and binary forms, with or without modification,
00016 are permitted provided that the following conditions are met:
00017
00018 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00019 and a copy of the modified files should be reported once modifications are
00020 completed, unless these modifications are made through the project's GitHub
00021 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00022 should be developed and included in any deliverable.
00023
00024 2. Redistributions of source code must be done through direct
00025 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00026
00027 3. Redistributions in binary form must reproduce the above copyright notice,
00028 this list of conditions and the following disclaimer in the documentation and/or
00029 other materials provided with the distribution.
00030
00031 4. Usage of the binary form on proprietary applications shall require explicit
00032 prior written permission from the the copyright holders, and due credit should
00033 be given to the copyright holders.
```

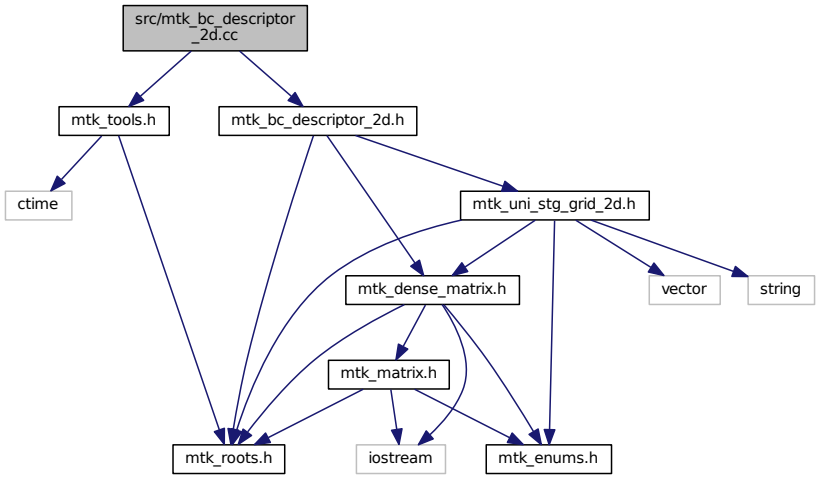
```
00034
00035 5. Neither the name of the copyright holder nor the names of its contributors
00036 may be used to endorse or promote products derived from this software without
00037 specific prior written permission.
00038
00039 The copyright holders provide no reassurances that the source code provided does
00040 not infringe any patent, copyright, or any other intellectual property rights of
00041 third parties. The copyright holders disclaim any liability to any recipient for
00042 claims brought against recipient by any third party for infringement of that
00043 parties intellectual property rights.
00044
00045 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00046 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00047 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00048 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00049 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00050 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00051 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00052 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00053 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00054 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00055 */
00056
00057 #include "mtk_tools.h"
00058
00059 #include "mtk_bc_descriptor_1d.h"
00060
00061 void mtk::BCDescriptor1D::ImposeOnLaplacianMatrix(
00062     mtk::DenseMatrix &matrix,
00063     const std::vector<mtk::Real> &west,
00064     const std::vector<mtk::Real> &east) {
00065
00066   #if MTK_DEBUG_LEVEL > 0
00067   mtk::Tools::Prevent(matrix.num_rows() == 0, __FILE__, __LINE__, __func__);
00068   mtk::Tools::Prevent(west.size() > (unsigned int) matrix.
      num_cols(),
00069                       __FILE__, __LINE__, __func__);
00070   mtk::Tools::Prevent(east.size() > (unsigned int) matrix.
      num_cols(),
00071                       __FILE__, __LINE__, __func__);
00072   #endif
00073
00075
00076   for (unsigned int ii = 0; ii < west.size(); ++ii) {
00077     matrix.SetValue(0, ii, west[ii]);
00078   }
00079
00081
00082   for (unsigned int ii = 0; ii < east.size(); ++ii) {
00083     matrix.SetValue(matrix.num_rows() - 1,
00084                     matrix.num_cols() - 1 - ii,
00085                     east[ii]);
00086   }
00087 }
00088
00089 void mtk::BCDescriptor1D::ImposeOnGrid(
      mtk::UniStgGrid1D &grid,
00090                                        const mtk::Real &omega,
00091                                        const mtk::Real &epsilon) {
00092
00093   #if MTK_DEBUG_LEVEL > 0
00094   mtk::Tools::Prevent(grid.num_cells_x() == 0, __FILE__, __LINE__, __func__);
00095   #endif
00096
00098
00099   grid.discrete_field_u()[0] = omega;
00100
00102
00103   grid.discrete_field_u()[grid.num_cells_x() + 2 - 1] = epsilon;
00104 }
```

17.55 src/mtk_bc_descriptor_2d.cc File Reference

Enforces boundary conditions in either the operator or the grid.

```
#include "mtk_tools.h"
#include "mtk_bc_descriptor_2d.h"
```

Include dependency graph for mtk_bc_descriptor_2d.cc:



17.55.1 Detailed Description

This class presents an interface for the user to specify boundary conditions on 2D mimetic operators and the grids they are acting on.

Def. Let f be any scalar or vector field defined over a domain Ω. We can specify any linear combination of f and its n derivatives to fulfill a condition, which we define as a **boundary condition**:

$$\forall \mathbf{x} \in \partial\Omega : \sum_{i=0}^{n} c_i(\mathbf{x}) < \hat{\mathbf{n}}, \frac{\partial^i f}{\partial x^i}(\mathbf{x}) >= \beta(\mathbf{x}).$$

This class receives information about the highest-order of differentiation, n, all possible coefficient functions, $c_i(\mathbf{x})$ for any subset of the boundary (south, north, west and east), and each condition for any subset of the boundary, and takes care of assigning them to both, the differentiation matrices and the grids.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_bc_descriptor_2d.cc.

17.56 mtk_bc_descriptor_2d.cc

```
00001
00030 /*
00031 Copyright (C) 2015, Computational Science Research Center, San Diego State
00032 University. All rights reserved.
00033
00034 Redistribution and use in source and binary forms, with or without modification,
```

```
00035 are permitted provided that the following conditions are met:
00036
00037 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00038 and a copy of the modified files should be reported once modifications are
00039 completed, unless these modifications are made through the project's GitHub
00040 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00041 should be developed and included in any deliverable.
00042
00043 2. Redistributions of source code must be done through direct
00044 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00045
00046 3. Redistributions in binary form must reproduce the above copyright notice,
00047 this list of conditions and the following disclaimer in the documentation and/or
00048 other materials provided with the distribution.
00049
00050 4. Usage of the binary form on proprietary applications shall require explicit
00051 prior written permission from the the copyright holders, and due credit should
00052 be given to the copyright holders.
00053
00054 5. Neither the name of the copyright holder nor the names of its contributors
00055 may be used to endorse or promote products derived from this software without
00056 specific prior written permission.
00057
00058 The copyright holders provide no reassurances that the source code provided does
00059 not infringe any patent, copyright, or any other intellectual property rights of
00060 third parties. The copyright holders disclaim any liability to any recipient for
00061 claims brought against recipient by any third party for infringement of that
00062 parties intellectual property rights.
00063
00064 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00065 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00066 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00067 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00068 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00069 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00070 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00071 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00072 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00073 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00074 */
00075
00076 #include "mtk_tools.h"
00077
00078 #include "mtk_bc_descriptor_2d.h"
00079
00080 mtk::BCDescriptor2D::BCDescriptor2D():
00081   highest_order_diff_west_(-1),
00082   highest_order_diff_east_(-1),
00083   highest_order_diff_south_(-1),
00084   highest_order_diff_north_(-1),
00085   west_condition_(),
00086   east_condition_(),
00087   south_condition_(),
00088   north_condition_() {}
00089
00090 mtk::BCDescriptor2D::BCDescriptor2D(const
      mtk::BCDescriptor2D &desc) {}
00091
00092 mtk::BCDescriptor2D::~BCDescriptor2D() noexcept {}
00093
00094 int mtk::BCDescriptor2D::highest_order_diff_west() const
      noexcept {
00095
00096   return highest_order_diff_west_;
00097 }
00098
00099 int mtk::BCDescriptor2D::highest_order_diff_east() const
      noexcept {
00100
00101   return highest_order_diff_east_;
00102 }
00103
00104 int mtk::BCDescriptor2D::highest_order_diff_south() const
      noexcept {
00105
00106   return highest_order_diff_south_;
00107 }
00108
00109 int mtk::BCDescriptor2D::highest_order_diff_north() const
      noexcept {
00110
```

```
00111   return highest_order_diff_north_;
00112 }
00113
00114 void mtk::BCDescriptor2D::PushBackWestCoeff(
      mtk::CoefficientFunction2D cw) {
00115
00116   #if MTK_DEBUG_LEVEL > 0
00117   mtk::Tools::Prevent(cw == nullptr, __FILE__, __LINE__, __func__);
00118   mtk::Tools::Prevent(highest_order_diff_west_ > 1,
00119                       __FILE__, __LINE__, __func__);
00120   #endif
00121
00122   west_coefficients_.push_back(cw);
00123
00124   highest_order_diff_west_++;
00125 }
00126
00127 void mtk::BCDescriptor2D::PushBackEastCoeff(
      mtk::CoefficientFunction2D ce) {
00128
00129   #if MTK_DEBUG_LEVEL > 0
00130   mtk::Tools::Prevent(ce == nullptr, __FILE__, __LINE__, __func__);
00131   mtk::Tools::Prevent(highest_order_diff_east_ > 1,
00132                       __FILE__, __LINE__, __func__);
00133   #endif
00134
00135   east_coefficients_.push_back(ce);
00136
00137   highest_order_diff_east_++;
00138 }
00139
00140 void mtk::BCDescriptor2D::PushBackSouthCoeff(
      mtk::CoefficientFunction2D cs) {
00141
00142   #if MTK_DEBUG_LEVEL > 0
00143   mtk::Tools::Prevent(cs == nullptr, __FILE__, __LINE__, __func__);
00144   mtk::Tools::Prevent(highest_order_diff_south_ > 1,
00145                       __FILE__, __LINE__, __func__);
00146   #endif
00147
00148   south_coefficients_.push_back(cs);
00149
00150   highest_order_diff_south_++;
00151 }
00152
00153 void mtk::BCDescriptor2D::PushBackNorthCoeff(
      mtk::CoefficientFunction2D cn) {
00154
00155   #if MTK_DEBUG_LEVEL > 0
00156   mtk::Tools::Prevent(cn == nullptr, __FILE__, __LINE__, __func__);
00157   mtk::Tools::Prevent(highest_order_diff_north_ > 1,
00158                       __FILE__, __LINE__, __func__);
00159   #endif
00160
00161   north_coefficients_.push_back(cn);
00162
00163   highest_order_diff_north_++;
00164 }
00165
00166 void mtk::BCDescriptor2D::set_west_condition(
00167     mtk::Real (*west_condition)(mtk::Real xx, mtk::Real yy)) noexcept {
00168
00169   #if MTK_DEBUG_LEVEL > 0
00170   mtk::Tools::Prevent(west_condition == nullptr, __FILE__, __LINE__, __func__);
00171   #endif
00172
00173   west_condition_ = west_condition;
00174 }
00175
00176 void mtk::BCDescriptor2D::set_east_condition(
00177     mtk::Real (*east_condition)(mtk::Real xx, mtk::Real yy)) noexcept {
00178
00179   #if MTK_DEBUG_LEVEL > 0
00180   mtk::Tools::Prevent(east_condition == nullptr, __FILE__, __LINE__, __func__);
00181   #endif
00182
00183   east_condition_ = east_condition;
00184 }
00185
00186 void mtk::BCDescriptor2D::set_south_condition(
00187     mtk::Real (*south_condition)(mtk::Real xx, mtk::Real yy)) noexcept {
```

```
00188
00189   #if MTK_DEBUG_LEVEL > 0
00190   mtk::Tools::Prevent(south_condition == nullptr,
00191                       __FILE__, __LINE__, __func__);
00192   #endif
00193
00194   south_condition_ = south_condition;
00195 }
00196
00197 void mtk::BCDescriptor2D::set_north_condition(
00198     mtk::Real (*north_condition)(mtk::Real xx, mtk::Real yy)) noexcept {
00199
00200   #if MTK_DEBUG_LEVEL > 0
00201   mtk::Tools::Prevent(north_condition == nullptr,
00202                       __FILE__, __LINE__, __func__);
00203   #endif
00204
00205   north_condition_ = north_condition;
00206 }
00207
00208 void mtk::BCDescriptor2D::ImposeOnSouthBoundaryNoSpace(
00209     const mtk::UniStgGrid2D &grid,
00210     mtk::DenseMatrix &matrix,
00211     const int &order_accuracy) const {
00212
00213   // At this point we have all of the information we need to fully impose the
00214   // south boundary condition:
00215   // 1. We have the collection of coefficients. The size of this collection
00216   // tells us the type of BC for this boundary.
00217   // 2. We have the grid that we can use to evaluate the coefficients at.
00218   // 3. We have the matrix where to place them.
00219
00220   // For now, we are sure that we will NOT have more than 2 coefficients per
00221   // boundary. That is, we only support Robin type FOR NOW.
00222
00224
00225   // For the south-west corner:
00226   auto cc = (south_coefficients_[0])(grid.west_bndy(), grid.south_bndy());
00227
00228   #if MTK_DEBUG_LEVEL > 0
00229   std::cout << "Matrix has " << matrix.num_rows() << " rows and " <<
00230     matrix.num_cols() << " columns." << std::endl;
00231   std::cout << "Setting at " << 0 << ' ' << 0 << std::endl;
00232   #endif
00233
00234   matrix.SetValue(0, 0, cc);
00235
00236   // Compute first centers per dimension.
00237   auto first_center_x = grid.west_bndy() + grid.delta_x()/
      mtk::kTwo;
00238
00239   // For each entry on the diagonal (south boundary):
00240   for (int ii = 0; ii < grid.num_cells_x(); ++ii) {
00241     // Evaluate next set spatial coordinates to evaluate the coefficient.
00242     mtk::Real xx = first_center_x + ii*grid.delta_x();
00243     // Evaluate and assign the Dirichlet coefficient.
00244     cc = (south_coefficients_[0])(xx, grid.south_bndy());
00245
00246     #if MTK_DEBUG_LEVEL > 0
00247     std::cout << "Setting at " << ii + 1 << ' ' << ii + 1 << std::endl;
00248     #endif
00249
00250     matrix.SetValue(ii + 1, ii + 1, cc);
00251   }
00252
00253   // For the south-east corner:
00254   cc = (south_coefficients_[0])(grid.east_bndy(), grid.south_bndy());
00255
00256   #if MTK_DEBUG_LEVEL > 0
00257   std::cout << "Setting at " << grid.num_cells_x() + 1 << ' ' <<
00258     grid.num_cells_x() + 1 << std::endl;
00259   #endif
00260
00261   matrix.SetValue(grid.num_cells_x() + 1, grid.num_cells_x() + 1, cc);
00262
00264
00266 }
00267
00268 void mtk::BCDescriptor2D::ImposeOnSouthBoundaryWithSpace
      (
00269     const mtk::UniStgGrid2D &grid,
```

```
00270     mtk::DenseMatrix &matrix,
00271     const int &order_accuracy) const {
00272
00274
00277
00278   // For each entry on the diagonal:
00279   for (int ii = 0; ii < grid.num_cells_x() + 2; ++ii) {
00280     // Evaluate next set spatial coordinates to evaluate the coefficient.
00281     mtk::Real xx{(grid.discrete_domain_x())[ii]};
00282     // Evaluate and assign the Dirichlet coefficient.
00283     mtk::Real cc = (south_coefficients_[0])(xx,grid.south_bndy());
00284     matrix.SetValue(ii, ii, cc);
00285   }
00286
00288 }
00289
00290 void mtk::BCDescriptor2D::ImposeOnNorthBoundaryNoSpace(
00291     const mtk::UniStgGrid2D &grid,
00292     mtk::DenseMatrix &matrix,
00293     const int &order_accuracy) const {
00294
00295   // At this point we have all of the information we need to fully impose the
00296   // north boundary condition:
00297   // 1. We have the collection of coefficients. The size of this collection
00298   // tells us the type of BC for this boundary.
00299   // 2. We have the grid that we can use to evaluate the coefficients at.
00300   // 3. We have the matrix where to place them.
00301
00302   // For now, we are sure that we will NOT have more than 2 coefficients per
00303   // boundary. That is, we only support Robin type FOR NOW.
00304
00305   int north_offset{(grid.num_cells_y() + 1)*(grid.num_cells_x() + 2)};
00306
00308
00309   // For the north-west corner:
00310   mtk::Real cc =
00311     (north_coefficients_[0])(grid.west_bndy(), grid.north_bndy());
00312
00313   #if MTK_DEBUG_LEVEL > 0
00314   std::cout << "Matrix has " << matrix.num_rows() << " rows and " <<
00315     matrix.num_cols() << " columns." << std::endl;
00316   std::cout << "Setting at " << north_offset << ' ' << north_offset <<
00317     std::endl;
00318   #endif
00319
00320   matrix.SetValue(north_offset, north_offset, cc);
00321
00322   // Compute first centers per dimension.
00323   auto first_center_x = grid.west_bndy() + grid.delta_x()/
      mtk::kTwo;
00324
00325   // For each entry on the diagonal (north boundary):
00326   for (int ii = 0; ii < grid.num_cells_x(); ++ii) {
00327     // Evaluate next set spatial coordinates to evaluate the coefficient.
00328     mtk::Real xx = first_center_x + ii*grid.delta_x();
00329     // Evaluate and assign the Dirichlet coefficient.
00330     cc = (north_coefficients_[0])(xx, grid.north_bndy());
00331
00332     #if MTK_DEBUG_LEVEL > 0
00333     std::cout << "Setting at " << north_offset + ii + 1 << ' ' <<
00334       north_offset + ii + 1 << std::endl;
00335     #endif
00336
00337     matrix.SetValue(north_offset + ii + 1, north_offset + ii + 1, cc);
00338   }
00339
00340   // For the north-east corner:
00341   cc = (north_coefficients_[0])(grid.east_bndy(), grid.north_bndy());
00342
00343   #if MTK_DEBUG_LEVEL > 0
00344   std::cout << "Setting at " << north_offset + grid.num_cells_x() + 1 <<
00345     ' ' << north_offset + grid.num_cells_x() + 1 << std::endl;
00346   #endif
00347
00348   matrix.SetValue(north_offset + grid.num_cells_x() + 1,
00349                   north_offset + grid.num_cells_x() + 1, cc);
00350
00352 }
00353
00354 void mtk::BCDescriptor2D::ImposeOnNorthBoundaryWithSpace
      (
```

```
00355     const mtk::UniStgGrid2D &grid,
00356     mtk::DenseMatrix &matrix,
00357     const int &order_accuracy) const {
00358
00360
00361   int north_offset{(grid.num_cells_y() + 1)*(grid.num_cells_x() + 2)};
00362
00364   for (int ii = 0; ii < grid.num_cells_x() + 2; ++ii) {
00366     mtk::Real xx{(grid.discrete_domain_x())[ii]};
00368     mtk::Real cc = (north_coefficients_[0])(xx, grid.north_bndy());
00369     matrix.SetValue(north_offset + ii, north_offset + ii, cc);
00370   }
00371
00373 }
00374
00375 void mtk::BCDescriptor2D::ImposeOnWestBoundaryNoSpace(
00376     const mtk::UniStgGrid2D &grid,
00377     mtk::DenseMatrix &matrix,
00378     const int &order_accuracy) const {
00379
00380   // At this point we have all of the information we need to fully impose the
00381   // west boundary condition:
00382   // 1. We have the collection of coefficients. The size of this collection
00383   // tells us the type of BC for this boundary.
00384   // 2. We have the grid that we can use to evaluate the coefficients at.
00385   // 3. We have the matrix where to place them.
00386
00388
00389   // For the south-west corner:
00390   auto cc = (west_coefficients_[0])(grid.west_bndy(), grid.south_bndy());
00391
00392   #if MTK_DEBUG_LEVEL > 0
00393   std::cout << "Matrix has " << matrix.num_rows() << " rows and " <<
00394     matrix.num_cols() << " columns." << std::endl;
00395   std::cout << "Setting at " << 0 << ' ' << 0 << std::endl;
00396   #endif
00397
00401
00402   mtk::Real harmonic_mean = mtk::kOne/matrix.GetValue(0, 0) +
      mtk::kOne/cc;
00403   harmonic_mean = mtk::kTwo/harmonic_mean;
00404
00405   matrix.SetValue(0, 0, harmonic_mean);
00406
00407   int west_offset{grid.num_cells_x() + 1};
00408
00409   auto first_center_y = grid.south_bndy() + grid.delta_y()/
      mtk::kTwo;
00410
00411   // For each west entry on the diagonal (west boundary):
00412   for (int ii = 0; ii < grid.num_cells_y(); ++ii) {
00413     // Evaluate next set spatial coordinates to evaluate the coefficient.
00414     mtk::Real yy = first_center_y + ii*grid.delta_y();
00415     // Evaluate and assign the Dirichlet coefficient.
00416     cc = (west_coefficients_[0])(grid.west_bndy(), yy);
00417
00418     #if MTK_DEBUG_LEVEL > 0
00419     std::cout << "Setting at " << west_offset + ii + 1 << ' ' <<
00420       west_offset + ii + 1 << std::endl;
00421     #endif
00422
00423     matrix.SetValue(west_offset + ii + 1, west_offset + ii + 1, cc);
00424
00425     west_offset += grid.num_cells_x() + 1;
00426   }
00427
00428   // For the north-west corner:
00429   cc = (west_coefficients_[0])(grid.west_bndy(), grid.north_bndy());
00430
00431   west_offset += grid.num_cells_x() + 1;
00432   int aux{west_offset};
00433   #if MTK_DEBUG_LEVEL > 0
00434   std::cout << "Setting at " << aux << ' ' << aux << std::endl;
00435   #endif
00436
00437   harmonic_mean = mtk::kOne/matrix.GetValue(aux, aux) +
      mtk::kOne/cc;
00438   harmonic_mean = mtk::kTwo/harmonic_mean;
00439
00440   matrix.SetValue(aux, aux, harmonic_mean);
00441
```

```
00443 }
00444
00445 void mtk::BCDescriptor2D::ImposeOnWestBoundaryWithSpace(
00446     const mtk::UniStgGrid2D &grid,
00447     mtk::DenseMatrix &matrix,
00448     const int &order_accuracy) const {
00449
00451
00452   int west_offset{grid.num_cells_x() + 1};
00453   // For each west entry on the diagonal:
00454   for (int ii = 0; ii < grid.num_cells_y() + 2; ++ii) {
00455     // Evaluate next set spatial coordinates to evaluate the coefficient.
00456     mtk::Real yy{(grid.discrete_domain_y())[ii]};
00457     // Evaluate and assign the Dirichlet coefficient.
00458     mtk::Real cc = (west_coefficients_[0])(grid.west_bndy(), yy);
00459     matrix.SetValue(west_offset + ii, west_offset + ii, cc);
00460     west_offset += grid.num_cells_x() + 1;
00461   }
00462
00464 }
00465
00466 void mtk::BCDescriptor2D::ImposeOnEastBoundaryNoSpace(
00467     const mtk::UniStgGrid2D &grid,
00468     mtk::DenseMatrix &matrix,
00469     const int &order_accuracy) const {
00470
00471   // At this point we have all of the information we need to fully impose the
00472   // east boundary condition:
00473   // 1. We have the collection of coefficients. The size of this collection
00474   // tells us the type of BC for this boundary.
00475   // 2. We have the grid that we can use to evaluate the coefficients at.
00476   // 3. We have the matrix where to place them.
00477
00479
00480   // For the south-east corner:
00481   auto cc = (east_coefficients_[0])(grid.east_bndy(), grid.south_bndy());
00482
00483   int east_offset{grid.num_cells_x() + 1};
00484   #if MTK_DEBUG_LEVEL > 0
00485   std::cout << "Matrix has " << matrix.num_rows() << " rows and " <<
00486     matrix.num_cols() << " columns." << std::endl;
00487   std::cout << "Setting at " << east_offset << ' ' << east_offset <<
00488     std::endl;
00489   #endif
00490
00491   mtk::Real harmonic_mean =
00492     mtk::kOne/matrix.GetValue(east_offset,east_offset) +
      mtk::kOne/cc;
00493   harmonic_mean = mtk::kTwo/harmonic_mean;
00494
00495   matrix.SetValue(east_offset, east_offset, harmonic_mean);
00496
00497   auto first_center_y = grid.south_bndy() + grid.delta_y()/
      mtk::kTwo;
00498
00499   // For each east entry on the diagonal (east boundary):
00500   for (int ii = 0; ii < grid.num_cells_y(); ++ii) {
00501
00502     east_offset += grid.num_cells_x() + 1;
00503
00504     // Evaluate next set spatial coordinates to evaluate the coefficient.
00505     mtk::Real yy = first_center_y + ii*grid.delta_y();
00506     // Evaluate and assign the Dirichlet coefficient.
00507     cc = (east_coefficients_[0])(grid.east_bndy(), yy);
00508
00509     #if MTK_DEBUG_LEVEL > 0
00510     std::cout << "Setting at " << east_offset + ii + 1 << ' ' <<
00511       east_offset + ii + 1 << std::endl;
00512     #endif
00513
00514     matrix.SetValue(east_offset + ii + 1, east_offset + ii + 1, cc);
00515   }
00516
00517   // For the north-east corner:
00518   cc = (east_coefficients_[0])(grid.east_bndy(), grid.north_bndy());
00519
00520   east_offset += grid.num_cells_x() + 1;
00521   east_offset += grid.num_cells_x() + 1;
00522   int aux{east_offset};
00523   #if MTK_DEBUG_LEVEL > 0
00524   std::cout << "Setting at " << aux << ' ' << aux << std::endl;
```

```
00525   #endif
00526
00527   harmonic_mean =
00528     mtk::kOne/matrix.GetValue(aux, aux) + mtk::kOne/cc;
00529   harmonic_mean = mtk::kTwo/harmonic_mean;
00530
00531   matrix.SetValue(aux, aux, harmonic_mean);
00532
00534 }
00535
00536 void mtk::BCDescriptor2D::ImposeOnEastBoundaryWithSpace(
00537     const mtk::UniStgGrid2D &grid,
00538     mtk::DenseMatrix &matrix,
00539     const int &order_accuracy) const {
00540
00542
00543   int east_offset{grid.num_cells_x() + 1};
00544   // For each west entry on the diagonal:
00545   for (int ii = 0; ii < grid.num_cells_y() + 2; ++ii) {
00546     east_offset += grid.num_cells_x() + 1;
00547     // Evaluate next set spatial coordinates to evaluate the coefficient.
00548     mtk::Real yy{(grid.discrete_domain_y())[ii]};
00549     // Evaluate and assign the arithmetic mean of Dirichlet coefficients.
00550     mtk::Real cc = (east_coefficients_[0])(grid.east_bndy(), yy);
00551     matrix.SetValue(east_offset + ii, east_offset + ii, cc);
00552   }
00553
00555 }
00556
00557 void mtk::BCDescriptor2D::ImposeOnLaplacianMatrix(
00558     const mtk::UniStgGrid2D &grid,
00559     mtk::DenseMatrix &matrix,
00560     const int &order_accuracy) const {
00561
00562   #if MTK_DEBUG_LEVEL > 0
00563   mtk::Tools::Prevent(highest_order_diff_south_ == -1,
00564                       __FILE__, __LINE__, __func__);
00565   mtk::Tools::Prevent(highest_order_diff_north_ == -1,
00566                       __FILE__, __LINE__, __func__);
00567   mtk::Tools::Prevent(highest_order_diff_west_ == -1,
00568                       __FILE__, __LINE__, __func__);
00569   mtk::Tools::Prevent(highest_order_diff_east_ == -1,
00570                       __FILE__, __LINE__, __func__);
00571   mtk::Tools::Prevent(grid.nature() != mtk::SCALAR,
00572                       __FILE__, __LINE__, __func__);
00573   mtk::Tools::Prevent(grid.num_cells_x() == 0, __FILE__, __LINE__, __func__);
00574   mtk::Tools::Prevent(grid.num_cells_y() == 0, __FILE__, __LINE__, __func__);
00575   mtk::Tools::Prevent(matrix.num_rows() == 0, __FILE__, __LINE__, __func__);
00576   mtk::Tools::Prevent(matrix.num_cols() == 0, __FILE__, __LINE__, __func__);
00577   #endif
00578
00581
00582   if (!grid.Bound()) {
00583     ImposeOnSouthBoundaryNoSpace(grid, matrix, order_accuracy);
00584     ImposeOnNorthBoundaryNoSpace(grid, matrix, order_accuracy);
00585     ImposeOnWestBoundaryNoSpace(grid, matrix, order_accuracy);
00586     ImposeOnEastBoundaryNoSpace(grid, matrix, order_accuracy);
00587   } else {
00588     ImposeOnSouthBoundaryWithSpace(grid, matrix, order_accuracy);
00589     ImposeOnNorthBoundaryWithSpace(grid, matrix, order_accuracy);
00590     ImposeOnWestBoundaryWithSpace(grid, matrix, order_accuracy);
00591     ImposeOnEastBoundaryWithSpace(grid, matrix, order_accuracy);
00592   }
00593 }
00594
00595 void mtk::BCDescriptor2D::ImposeOnGrid(
      mtk::UniStgGrid2D &grid) const {
00596
00597   #if MTK_DEBUG_LEVEL > 0
00598   mtk::Tools::Prevent(grid.num_cells_x() == 0, __FILE__, __LINE__, __func__);
00599   mtk::Tools::Prevent(grid.num_cells_y() == 0, __FILE__, __LINE__, __func__);
00600   mtk::Tools::Prevent(west_condition_ == nullptr, __FILE__, __LINE__, __func__);
00601   mtk::Tools::Prevent(east_condition_ == nullptr, __FILE__, __LINE__, __func__);
00602   mtk::Tools::Prevent(south_condition_ == nullptr,
00603                       __FILE__, __LINE__, __func__);
00604   mtk::Tools::Prevent(north_condition_ == nullptr,
00605                       __FILE__, __LINE__, __func__);
00606   #endif
00607
00609   if (grid.nature() == mtk::SCALAR) {
00610
```

```
00612
00614     mtk::Real xx = grid.west_bndy();
00615     mtk::Real yy = grid.south_bndy();
00616     (grid.discrete_field())[0] = south_condition_(xx, yy);
00617
00619     xx = xx + grid.delta_x()/mtk::kTwo;
00620     // For every point on the south boundary:
00621     for (int ii = 0; ii < grid.num_cells_x(); ++ii) {
00622       (grid.discrete_field())[ii + 1] =
00623         south_condition_(xx + ii*grid.delta_x(), yy);
00624     }
00625
00627     xx = grid.east_bndy();
00628     (grid.discrete_field())[grid.num_cells_x() + 1] = south_condition_(xx, yy);
00629
00631
00633     xx = grid.west_bndy();
00634     yy = grid.north_bndy();
00635     int north_offset{(grid.num_cells_y() + 1)*(grid.num_cells_x() + 2)};
00636     (grid.discrete_field())[north_offset] = north_condition_(xx, yy);
00637
00639     xx = xx + grid.delta_x()/mtk::kTwo;
00640     for (int ii = 0; ii < grid.num_cells_x(); ++ii) {
00641       (grid.discrete_field())[north_offset + ii + 1] =
00642         north_condition_(xx + ii*grid.delta_x(), yy);
00643     }
00644
00646     xx = grid.east_bndy();
00647     yy = grid.north_bndy();
00648     (grid.discrete_field())[north_offset + grid.num_cells_x() + 1] =
00649         north_condition_(xx, yy);
00650
00652
00656     xx = grid.west_bndy();
00657     yy = grid.south_bndy();
00658     (grid.discrete_field())[0] =
00659       ((grid.discrete_field())[0] + west_condition_(xx,yy))/
      mtk::kTwo;
00660
00662     int west_offset{grid.num_cells_x() + 1 + 1};
00663     yy = yy + grid.delta_y()/mtk::kTwo;
00664     for (int ii = 0; ii < grid.num_cells_y(); ++ii) {
00665       #if MTK_DEBUG_LEVEL > 0
00666       std::cout << "Adding on " << west_offset << "-th position." << std::endl;
00667       #endif
00668       (grid.discrete_field())[west_offset] =
00669         west_condition_(xx, yy + ii*grid.delta_y());
00670       west_offset += grid.num_cells_x() + 1 + 1;
00671     }
00672
00674     xx = grid.west_bndy();
00675     yy = grid.north_bndy();
00676     north_offset = (grid.num_cells_y() + 1)*(grid.num_cells_x() + 2);
00677     (grid.discrete_field())[north_offset] =
00678       ((grid.discrete_field())[north_offset] + west_condition_(xx, yy))/
00679         mtk::kTwo;
00680
00682
00684     xx = grid.east_bndy();
00685     yy = grid.south_bndy();
00686     int east_offset{grid.num_cells_x() + 1};
00687     (grid.discrete_field())[east_offset] =
00688       ((grid.discrete_field())[east_offset] + east_condition_(xx, yy))/
00689         mtk::kTwo;
00690
00692     yy = yy + grid.delta_y()/mtk::kTwo;
00693     for (int ii = 0; ii < grid.num_cells_y(); ++ii) {
00694       east_offset += grid.num_cells_x() + 1 + 1;
00695       #if MTK_DEBUG_LEVEL > 0
00696       std::cout << "Adding on " << east_offset << "-th position." << std::endl;
00697       #endif
00698       (grid.discrete_field())[east_offset] =
00699         east_condition_(xx, yy + ii*grid.delta_y());
00700     }
00701
00703     xx = grid.east_bndy();
00704     yy = grid.north_bndy();
00705     (grid.discrete_field())[north_offset + grid.num_cells_x() + 1] =
00706       ((grid.discrete_field())[north_offset + grid.num_cells_x() + 1] +
00707       east_condition_(xx, yy))/mtk::kTwo;
00708
```

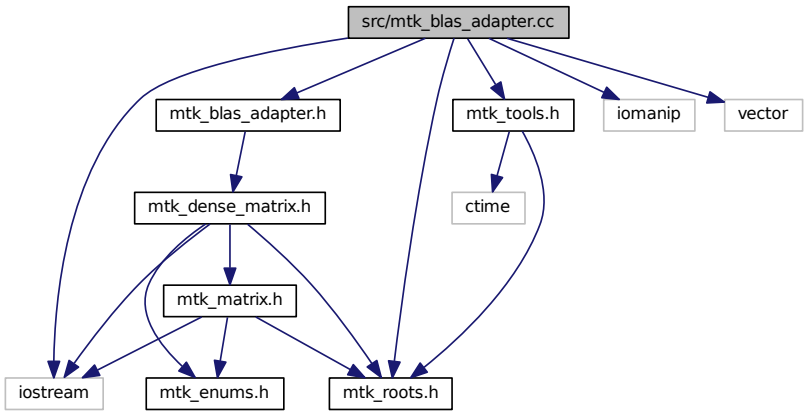
```
00709   } else {
00710
00712
00714   }
00715 }
```

17.57 src/mtk_blas_adapter.cc File Reference

Adapter class for the BLAS API.

```
#include <iostream>
#include <iomanip>
#include <vector>
#include "mtk_roots.h"
#include "mtk_tools.h"
#include "mtk_blas_adapter.h"
```

Include dependency graph for mtk_blas_adapter.cc:



Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

Functions

- float mtk::snrm2_ (int *n, float *x, int *incx)
- void mtk::saxpy_ (int *n, float *sa, float *sx, int *incx, float *sy, int *incy)
- void mtk::sgemv_ (char *trans, int *m, int *n, float *alpha, float *a, int *lda, float *x, int *incx, float *beta, float *y, int *incy)
- void mtk::sgemm_ (char *transa, char *transb, int *m, int *n, int *k, double *alpha, double *a, int *lda, double *b, aamm int *ldb, double *beta, double *c, int *ldc)

17.57.1 Detailed Description

This class contains a collection of static classes, that posses direct access to the underlying structure of the matrices, thus allowing programmers to exploit some of the numerical methods implemented in the BLAS.

The **BLAS (Basic Linear Algebra Subprograms)** are routines that provide standard building blocks for performing basic vector and matrix operations. The Level 1 BLAS perform scalar, vector and vector-vector operations, the Level 2 BLAS perform matrix-vector operations, and the Level 3 BLAS perform matrix-matrix operations.

The BLAS can be installed from links given in the See Also section of this page.

See also

http://www.netlib.org/blas/
https://software.intel.com/en-us/non-commercial-software-development

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_blas_adapter.cc.

17.58 mtk_blas_adapter.cc

```
00001
00024 /*
00025 Copyright (C) 2015, Computational Science Research Center, San Diego State
00026 University. All rights reserved.
00027
00028 Redistribution and use in source and binary forms, with or without modification,
00029 are permitted provided that the following conditions are met:
00030
00031 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00032 and a copy of the modified files should be reported once modifications are
00033 completed, unless these modifications are made through the project's GitHub
00034 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00035 should be developed and included in any deliverable.
00036
00037 2. Redistributions of source code must be done through direct
00038 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00039
00040 3. Redistributions in binary form must reproduce the above copyright notice,
00041 this list of conditions and the following disclaimer in the documentation and/or
00042 other materials provided with the distribution.
00043
00044 4. Usage of the binary form on proprietary applications shall require explicit
00045 prior written permission from the the copyright holders, and due credit should
00046 be given to the copyright holders.
00047
00048 5. Neither the name of the copyright holder nor the names of its contributors
00049 may be used to endorse or promote products derived from this software without
00050 specific prior written permission.
00051
00052 The copyright holders provide no reassurances that the source code provided does
00053 not infringe any patent, copyright, or any other intellectual property rights of
00054 third parties. The copyright holders disclaim any liability to any recipient for
00055 claims brought against recipient by any third party for infringement of that
00056 parties intellectual property rights.
00057
00058 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00059 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00060 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00061 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00062 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00063 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00064 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00065 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00066 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
```

```
00067 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00068 */
00069
00070 #include <iostream>
00071 #include <iomanip>
00072
00073 #include <vector>
00074
00075 #include "mtk_roots.h"
00076 #include "mtk_tools.h"
00077 #include "mtk_blas_adapter.h"
00078
00079 namespace mtk {
00080
00081 extern "C" {
00082
00083 #ifdef MTK_PRECISION_DOUBLE
00084
00097 double dnrm2_(int *n, double *x, int *incx);
00098 #else
00099
00112 float snrm2_(int *n, float *x, int *incx);
00113 #endif
00114
00115 #ifdef MTK_PRECISION_DOUBLE
00116
00135 void daxpy_(int *n, double *da, double *dx, int *incx, double *dy, int *incy);
00136 #else
00137
00156 void saxpy_(int *n, float *sa, float *sx, int *incx, float *sy, int *incy);
00157 #endif
00158
00159 #ifdef MTK_PRECISION_DOUBLE
00160
00188 void dgemv_(char *trans,
00189             int *m,
00190             int *n,
00191             double *alpha,
00192             double *a,
00193             int *lda,
00194             double *x,
00195             int *incx,
00196             double *beta,
00197             double *y,
00198             int *incy);
00199 #else
00200
00228 void sgemv_(char *trans,
00229             int *m,
00230             int *n,
00231             float *alpha,
00232             float *a,
00233             int *lda,
00234             float *x,
00235             int *incx,
00236             float *beta,
00237             float *y,
00238             int *incy);
00239 #endif
00240
00241 #ifdef MTK_PRECISION_DOUBLE
00242
00267 void dgemm_(char *transa,
00268             char* transb,
00269             int *m,
00270             int *n,
00271             int *k,
00272             double *alpha,
00273             double *a,
00274             int *lda,
00275             double *b,
00276             int *ldb,
00277             double *beta,
00278             double *c,
00279             int *ldc);
00280 }
00281 #else
00282
00307 void sgemm_(char *transa,
00308             char* transb,
00309             int *m,
```

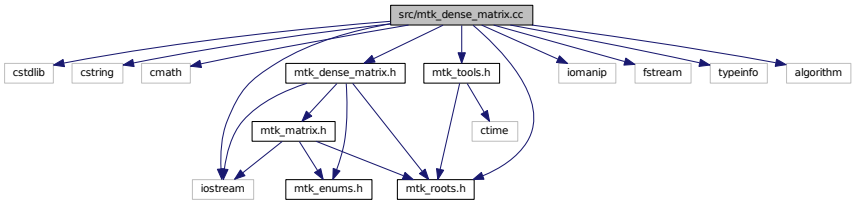
```
00310             int *n,
00311             int *k,
00312             double *alpha,
00313             double *a,
00314             int *lda,
00315             double *b,aamm
00316             int *ldb,
00317             double *beta,
00318             double *c,
00319             int *ldc);
00320 }
00321 #endif
00322 }
00323
00324 mtk::Real mtk::BLASAdapter::RealNRM2(Real *in, int &in_length) {
00325
00326   #if MTK_DEBUG_LEVEL > 0
00327   mtk::Tools::Prevent(in_length <= 0, __FILE__, __LINE__, __func__);
00328   #endif
00329
00330   int incx{1};  // Increment for the elements of xx. ix >= 0.
00331
00332   #ifdef MTK_PRECISION_DOUBLE
00333   return dnrm2_(&in_length, in, &incx);
00334   #else
00335   return snrm2_(&in_length, in, &incx);
00336   #endif
00337 }
00338
00339 void mtk::BLASAdapter::RealAXPY(mtk::Real alpha,
00340                                 mtk::Real *xx,
00341                                 mtk::Real *yy,
00342                                 int &in_length) {
00343
00344   #if MTK_DEBUG_LEVEL > 0
00345   mtk::Tools::Prevent(xx == nullptr, __FILE__, __LINE__, __func__);
00346   mtk::Tools::Prevent(yy == nullptr, __FILE__, __LINE__, __func__);
00347   #endif
00348
00349   int incx{1};  // Increment for the elements of xx. ix >= 0.
00350
00351   #ifdef MTK_PRECISION_DOUBLE
00352   daxpy_(&in_length, &alpha, xx, &incx, yy, &incx);
00353   #else
00354   saxpy_(&in_length, &alpha, xx, &incx, yy, &incx);
00355   #endif
00356 }
00357
00358 mtk::Real mtk::BLASAdapter::RelNorm2Error(
      mtk::Real *computed,
00359                                         mtk::Real *known,
00360                                         int length) {
00361
00362   #if MTK_DEBUG_LEVEL > 0
00363   mtk::Tools::Prevent(computed == nullptr, __FILE__, __LINE__, __func__);
00364   mtk::Tools::Prevent(known == nullptr, __FILE__, __LINE__, __func__);
00365   #endif
00366
00367   mtk::Real norm_2_computed{mtk::BLASAdapter::RealNRM2(known, length)};
00368
00369   mtk::Real alpha{-mtk::kOne};
00370
00371   mtk::BLASAdapter::RealAXPY(alpha, known, computed, length);
00372
00373   mtk::Real norm_2_difference{mtk::BLASAdapter::RealNRM2(computed,
      length)};
00374
00375   return norm_2_difference/norm_2_computed;
00376 }
00377
00378 void mtk::BLASAdapter::RealDenseMV(mtk::Real &alpha,
00379                                    mtk::DenseMatrix &aa,
00380                                    mtk::Real *xx,
00381                                    mtk::Real &beta,
00382                                    mtk::Real *yy) {
00383
00384   // Make sure input matrices are row-major ordered.
00385
00386   if (aa.matrix_properties().ordering() ==
      mtk::COL_MAJOR) {
00387     aa.OrderRowMajor();
```

```
00388   }
00389
00390   char transa{'T'}; // State that now, the input WILL be in row-major ordering.
00391
00392   int mm{aa.num_rows()};                  // Rows of aa.
00393   int nn{aa.num_cols()};                  // Columns of aa.
00394   int lda{(aa.matrix_properties()).ld()}; // Leading dimension.
00395   int incx{1};                            // Increment of values in x.
00396   int incy{1};                            // Increment of values in y.
00397
00398   std::swap(mm,nn);
00399   #ifdef MTK_PRECISION_DOUBLE
00400   dgemv_(&transa, &mm, &nn, &alpha, aa.data(), &lda,
00401          xx, &incx, &beta, yy, &incy);
00402   #else
00403   sgemv_(&transa, &mm, &nn, &alpha, aa.data(), &lda,
00404         xx, &incx, &beta, yy, &incy);
00405   #endif
00406   std::swap(mm,nn);
00407 }
00408
00409 mtk::DenseMatrix mtk::BLASAdapter::RealDenseMM(
      mtk::DenseMatrix &aa,
00410                                                mtk::DenseMatrix &bb) {
00411
00412   #if MTK_DEBUG_LEVEL > 0
00413   mtk::Tools::Prevent(aa.num_cols() != bb.num_rows(),
00414                       __FILE__, __LINE__, __func__);
00415   #endif
00416
00417   // Make sure input matrices are row-major ordered.
00418
00419   if (aa.matrix_properties().ordering() ==
      mtk::COL_MAJOR) {
00420     aa.OrderRowMajor();
00421   }
00422   if (bb.matrix_properties().ordering() ==
      mtk::COL_MAJOR) {
00423     bb.OrderRowMajor();
00424   }
00425
00426   char ta{'T'}; // State that input matrix aa is in row-wise ordering.
00427   char tb{'T'}; // State that input matrix bb is in row-wise ordering.
00428
00429   int mm{aa.num_rows()};  // Rows of aa and rows of cc.
00430   int nn{bb.num_cols()};  // Cols of bb and cols of cc.
00431   int kk{aa.num_cols()};  // Cols of aa and rows of bb.
00432
00433   int cc_num_rows{mm};  // Rows of cc.
00434   int cc_num_cols{nn};  // Columns of cc.
00435
00436   int lda{std::max(1,kk)};  // Leading dimension of the aa matrix.
00437   int ldb{std::max(1,nn)};  // Leading dimension of the bb matrix.
00438   int ldc{std::max(1,mm)};  // Leading dimension of the cc matrix.
00439
00440   mtk::Real alpha{1.0}; // First scalar coefficient.
00441   mtk::Real beta{0.0};  // Second scalar coefficient.
00442
00443   mtk::DenseMatrix cc_col_maj_ord(cc_num_rows,cc_num_cols); // Output matrix.
00444
00445   cc_col_maj_ord.SetOrdering(mtk::COL_MAJOR);
00446
00447   #ifdef MTK_PRECISION_DOUBLE
00448   dgemm_(&ta, &tb, &mm, &nn, &kk, &alpha, aa.data(), &lda,
00449          bb.data(), &ldb, &beta, cc_col_maj_ord.data(), &ldc);
00450   #else
00451   sgemm_(&ta, &tb, &mm, &nn, &kk, &alpha, aa.data(), &lda,
00452          bb.data(), &ldb, &beta, cc_col_maj_ord.data(), &ldc);
00453   #endif
00454
00455   #if MTK_DEBUG_LEVEL > 0
00456   std::cout << "cc_col_maj_ord =" << std::endl;
00457   std::cout << cc_col_maj_ord << std::endl;
00458   #endif
00459
00460   cc_col_maj_ord.OrderRowMajor();
00461
00462   return cc_col_maj_ord;
00463 }
```

17.59 src/mtk_dense_matrix.cc File Reference

```
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <typeinfo>
#include <algorithm>
#include "mtk_roots.h"
#include "mtk_dense_matrix.h"
#include "mtk_tools.h"
```

Include dependency graph for mtk_dense_matrix.cc:



Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

Functions

- std::ostream & mtk::operator<< (std::ostream &stream, mtk::DenseMatrix &in)

17.60 mtk_dense_matrix.cc

```
00001
00013 /*
00014 Copyright (C) 2015, Computational Science Research Center, San Diego State
00015 University. All rights reserved.
00016
00017 Redistribution and use in source and binary forms, with or without modification,
00018 are permitted provided that the following conditions are met:
00019
00020 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00021 and a copy of the modified files should be reported once modifications are
00022 completed, unless these modifications are made through the project's GitHub
00023 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00024 should be developed and included in any deliverable.
00025
00026 2. Redistributions of source code must be done through direct
00027 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00028
00029 3. Redistributions in binary form must reproduce the above copyright notice,
00030 this list of conditions and the following disclaimer in the documentation and/or
00031 other materials provided with the distribution.
```

```
00032
00033 4. Usage of the binary form on proprietary applications shall require explicit
00034 prior written permission from the the copyright holders, and due credit should
00035 be given to the copyright holders.
00036
00037 5. Neither the name of the copyright holder nor the names of its contributors
00038 may be used to endorse or promote products derived from this software without
00039 specific prior written permission.
00040
00041 The copyright holders provide no reassurances that the source code provided does
00042 not infringe any patent, copyright, or any other intellectual property rights of
00043 third parties. The copyright holders disclaim any liability to any recipient for
00044 claims brought against recipient by any third party for infringement of that
00045 parties intellectual property rights.
00046
00047 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00048 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00049 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00050 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00051 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00052 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00053 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00054 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00055 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00056 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00057 */
00058
00059 #include <cstdlib>
00060 #include <cstring>
00061 #include <cmath>
00062
00063 #include <iostream>
00064 #include <iomanip>
00065 #include <fstream>
00066
00067 #include <typeinfo>
00068
00069 #include <algorithm>
00070
00071 #include "mtk_roots.h"
00072 #include "mtk_dense_matrix.h"
00073 #include "mtk_tools.h"
00074
00075 namespace mtk {
00076
00077 std::ostream& operator <<(std::ostream &stream, mtk::DenseMatrix &in) {
00078
00079   int mm{in.matrix_properties_.num_rows()};  // Auxiliary.
00080   int nn{in.matrix_properties_.num_cols()};  // Auxiliary.
00081
00082   if (in.matrix_properties_.ordering() ==
      mtk::COL_MAJOR) {
00083     std::swap(mm, nn);
00084   }
00085   for (int ii = 0; ii < mm; ii++) {
00086     int offset{ii*nn};
00087     for (int jj = 0; jj < nn; jj++) {
00088       mtk::Real value = in.data_[offset + jj];
00089       stream << std::setw(9) << value;
00090     }
00091     stream << std::endl;
00092   }
00093   if (in.matrix_properties_.ordering() ==
      mtk::COL_MAJOR) {
00094     std::swap(mm, nn);
00095   }
00096   return stream;
00097 }
00098 }
00099
00100 mtk::DenseMatrix& mtk::DenseMatrix::operator =(const
      mtk::DenseMatrix &in) {
00101
00102   if(this == &in) {
00103     return *this;
00104   }
00105
00106   matrix_properties_.set_storage(in.
      matrix_properties_.storage());
00107
00108   matrix_properties_.set_ordering(in.
```

```
      matrix_properties_.ordering());
00109
00110   auto aux = in.matrix_properties_.num_rows();
00111   matrix_properties_.set_num_rows(aux);
00112
00113   aux = in.matrix_properties().num_cols();
00114   matrix_properties_.set_num_cols(aux);
00115
00116   aux = in.matrix_properties().num_zero();
00117   matrix_properties_.set_num_zero(aux);
00118
00119   aux = in.matrix_properties().num_null();
00120   matrix_properties_.set_num_null(aux);
00121
00122   auto num_rows = matrix_properties_.num_rows();
00123   auto num_cols = matrix_properties_.num_cols();
00124
00125   delete [] data_;
00126
00127   try {
00128     data_ = new mtk::Real[num_rows*num_cols];
00129   } catch (std::bad_alloc &memory_allocation_exception) {
00130     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00131       std::endl;
00132     std::cerr << memory_allocation_exception.what() << std::endl;
00133   }
00134   memset(data_, mtk::kZero, sizeof(data_[0])*num_rows*
      num_cols);
00135
00136   std::copy(in.data_, in.data_ + num_rows*num_cols, data_);
00137
00138   return *this;
00139 }
00140
00141 bool mtk::DenseMatrix::operator ==(const
      DenseMatrix &in) {
00142
00143   bool ans{true};
00144
00145   auto mm = in.num_rows();
00146   auto nn = in.num_cols();
00147
00148   if (mm != matrix_properties_.num_rows() ||
00149       nn != matrix_properties_.num_cols()) {
00150     return false;
00151   }
00152
00153   for (int ii = 0; ii < mm && ans; ++ii) {
00154     for (int jj = 0; jj < nn && ans; ++jj) {
00155       ans = ans &&
00156         abs(data_[ii*nn + jj] - in.data()[ii*nn + jj]) <
      mtk::kDefaultTolerance;
00157     }
00158   }
00159   return ans;
00160 }
00161
00162 mtk::DenseMatrix::DenseMatrix(): data_(nullptr) {
00163
00164   matrix_properties_.set_storage(mtk::DENSE);
00165   matrix_properties_.set_ordering(mtk::ROW_MAJOR);
00166 }
00167
00168 mtk::DenseMatrix::DenseMatrix(const
      mtk::DenseMatrix &in) {
00169
00170   matrix_properties_.set_storage(in.matrix_properties_.storage());
00171
00172   matrix_properties_.set_ordering(in.matrix_properties_.
      ordering());
00173
00174   auto aux = in.matrix_properties_.num_rows();
00175   matrix_properties_.set_num_rows(aux);
00176
00177   aux = in.matrix_properties().num_cols();
00178   matrix_properties_.set_num_cols(aux);
00179
00180   aux = in.matrix_properties().num_zero();
00181   matrix_properties_.set_num_zero(aux);
00182
00183   aux = in.matrix_properties().num_null();
```

```
00184   matrix_properties_.set_num_null(aux);
00185
00186   auto num_rows = in.matrix_properties_.num_rows();
00187   auto num_cols = in.matrix_properties_.num_cols();
00188
00189   try {
00190     data_ = new mtk::Real[num_rows*num_cols];
00191   } catch (std::bad_alloc &memory_allocation_exception) {
00192     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00193       std::endl;
00194     std::cerr << memory_allocation_exception.what() << std::endl;
00195   }
00196   memset(data_, mtk::kZero, sizeof(data_[0])*num_rows*num_cols);
00197
00198   std::copy(in.data_,in.data_ + num_rows*num_cols,data_);
00199 }
00200
00201 mtk::DenseMatrix::DenseMatrix(const int &num_rows, const int &num_cols) {
00202
00203   #if MTK_DEBUG_LEVEL > 0
00204   mtk::Tools::Prevent(num_rows < 1, __FILE__, __LINE__, __func__);
00205   mtk::Tools::Prevent(num_cols < 1, __FILE__, __LINE__, __func__);
00206   #endif
00207
00208   matrix_properties_.set_storage(mtk::DENSE);
00209   matrix_properties_.set_ordering(mtk::ROW_MAJOR);
00210   matrix_properties_.set_num_rows(num_rows);
00211   matrix_properties_.set_num_cols(num_cols);
00212
00213   try {
00214     data_ = new mtk::Real[num_rows*num_cols];
00215   } catch (std::bad_alloc &memory_allocation_exception) {
00216     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00217       std::endl;
00218     std::cerr << memory_allocation_exception.what() << std::endl;
00219   }
00220   memset(data_, mtk::kZero, sizeof(data_[0])*num_rows*num_cols);
00221 }
00222
00223 mtk::DenseMatrix::DenseMatrix(const int &rank,
00224                               const bool &padded,
00225                               const bool &transpose) {
00226
00227   #if MTK_DEBUG_LEVEL > 0
00228   mtk::Tools::Prevent(rank < 1, __FILE__, __LINE__, __func__);
00229   #endif
00230
00231   int aux{};  // Used to control the padding.
00232
00233   if (padded) {
00234     aux = 1;
00235   }
00236
00237   matrix_properties_.set_storage(mtk::DENSE);
00238   matrix_properties_.set_ordering(mtk::ROW_MAJOR);
00239   matrix_properties_.set_num_rows(aux + rank + aux);
00240   matrix_properties_.set_num_cols(rank);
00241
00242   try {
00243     data_ = new mtk::Real[matrix_properties_.num_values()];
00244   } catch (std::bad_alloc &memory_allocation_exception) {
00245     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00246       std::endl;
00247     std::cerr << memory_allocation_exception.what() << std::endl;
00248   }
00249   memset(data_,
00250          mtk::kZero,
00251          sizeof(data_[0])*(matrix_properties_.num_values()));
00252
00253   for (auto ii =0 ; ii < matrix_properties_.num_rows(); ++ii) {
00254     for (auto jj = 0; jj < matrix_properties_.num_cols(); ++jj) {
00255       data_[ii*matrix_properties_.num_cols() + jj] =
00256         (ii == jj + aux)? mtk::kOne: mtk::kZero;
00257     }
00258   }
00259   if (transpose) {
00260     Transpose();
00261   }
00262 }
00263
00264 mtk::DenseMatrix::DenseMatrix(const mtk::Real *const gen,
```

```
00265                              const int &gen_length,
00266                              const int &pro_length,
00267                              const bool &transpose) {
00268
00269   #if MTK_DEBUG_LEVEL > 0
00270   mtk::Tools::Prevent(gen == nullptr, __FILE__, __LINE__, __func__);
00271   mtk::Tools::Prevent(gen_length < 1, __FILE__, __LINE__, __func__);
00272   mtk::Tools::Prevent(pro_length < 1, __FILE__, __LINE__, __func__);
00273   #endif
00274
00275   matrix_properties_.set_storage(mtk::DENSE);
00276   matrix_properties_.set_ordering(mtk::ROW_MAJOR);
00277   if (!transpose) {
00278     matrix_properties_.set_num_rows(gen_length);
00279     matrix_properties_.set_num_cols(pro_length);
00280   } else {
00281     matrix_properties_.set_num_rows(pro_length);
00282     matrix_properties_.set_num_cols(gen_length);
00283   }
00284
00285   int mm = matrix_properties_.num_rows(); // Used to construct this matrix.
00286   int nn = matrix_properties_.num_cols(); // Used to construct this matrix.
00287
00288   try {
00289     data_ = new mtk::Real[mm*nn];
00290   } catch (std::bad_alloc &memory_allocation_exception) {
00291     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00292       std::endl;
00293     std::cerr << memory_allocation_exception.what() << std::endl;
00294   }
00295   memset(data_, mtk::kZero, sizeof(data_[0])*mm*nn);
00296
00297   if (!transpose) {
00298     for (auto ii = 0; ii < mm; ii++) {
00299       for (auto jj = 0; jj < nn; jj++) {
00300         data_[ii*nn + jj] = pow(gen[ii], (double) jj);
00301       }
00302     }
00303   } else {
00304     for (auto ii = 0; ii < mm; ii++) {
00305       for (auto jj = 0; jj < nn; jj++) {
00306         data_[ii*nn + jj] = pow(gen[jj], (double) ii);
00307       }
00308     }
00309   }
00310 }
00311
00312 mtk::DenseMatrix::~DenseMatrix() {
00313
00314   delete [] data_;
00315   data_ = nullptr;
00316 }
00317
00318 mtk::Matrix mtk::DenseMatrix::matrix_properties() const
     noexcept {
00319
00320   return matrix_properties_;
00321 }
00322
00323 void mtk::DenseMatrix::SetOrdering(
     mtk::MatrixOrdering oo) noexcept {
00324
00325   #if MTK_DEBUG_LEVEL > 0
00326   mtk::Tools::Prevent(!(oo == mtk::ROW_MAJOR || oo ==
     mtk::COL_MAJOR),
00327                       __FILE__, __LINE__, __func__);
00328   #endif
00329
00330   matrix_properties_.set_ordering(oo);
00331 }
00332
00333 int mtk::DenseMatrix::num_rows() const noexcept {
00334
00335   return matrix_properties_.num_rows();
00336 }
00337
00338 int mtk::DenseMatrix::num_cols() const noexcept {
00339
00340   return matrix_properties_.num_cols();
00341 }
00342
```

```
00343 mtk::Real* mtk::DenseMatrix::data() const noexcept {
00344
00345   return data_;
00346 }
00347
00348 mtk::Real mtk::DenseMatrix::GetValue(
00349     const int &mm,
00350     const int &nn) const noexcept {
00351
00352   #if MTK_DEBUG_LEVEL > 0
00353   mtk::Tools::Prevent(mm < 0, __FILE__, __LINE__, __func__);
00354   mtk::Tools::Prevent(nn < 0, __FILE__, __LINE__, __func__);
00355   #endif
00356
00357   return data_[mm*matrix_properties_.num_cols() + nn];
00358 }
00359
00360 void  mtk::DenseMatrix::SetValue(
00361     const int &mm,
00362     const int &nn,
00363     const mtk::Real &val) noexcept {
00364
00365   #if MTK_DEBUG_LEVEL > 0
00366   mtk::Tools::Prevent(mm < 0, __FILE__, __LINE__, __func__);
00367   mtk::Tools::Prevent(nn < 0, __FILE__, __LINE__, __func__);
00368   #endif
00369
00370   data_[mm*matrix_properties_.num_cols() + nn] = val;
00371 }
00372
00373 void mtk::DenseMatrix::Transpose() {
00374
00376
00377   mtk::Real *data_transposed{}; // Buffer.
00378
00379   int mm = matrix_properties_.num_rows(); // Used to construct this matrix.
00380   int nn = matrix_properties_.num_cols(); // Used to construct this matrix.
00381
00382   try {
00383     data_transposed = new mtk::Real[mm*nn];
00384   } catch (std::bad_alloc &memory_allocation_exception) {
00385     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00386       std::endl;
00387     std::cerr << memory_allocation_exception.what() << std::endl;
00388   }
00389   memset(data_transposed,
00390          mtk::kZero,
00391          sizeof(data_transposed[0])*mm*nn);
00392
00393   // Assign the values to their transposed position.
00394   for (auto ii = 0; ii < mm; ++ii) {
00395     for (auto jj = 0; jj < nn; ++jj) {
00396       data_transposed[jj*mm + ii] = data_[ii*nn + jj];
00397     }
00398   }
00399
00400   // Swap pointers.
00401   auto tmp = data_; // Temporal holder.
00402   data_ = data_transposed;
00403   delete [] tmp;
00404   tmp = nullptr;
00405
00406   matrix_properties_.set_num_rows(nn);
00407   matrix_properties_.set_num_cols(mm);
00408 }
00409
00410 void mtk::DenseMatrix::OrderRowMajor() {
00411
00412   if (matrix_properties_.ordering() == mtk::COL_MAJOR) {
00413
00415
00416     mtk::Real *data_transposed{}; // Buffer.
00417
00418     int mm = matrix_properties_.num_rows(); // Used to construct this matrix.
00419     int nn = matrix_properties_.num_cols(); // Used to construct this matrix.
00420
00421     try {
00422       data_transposed = new mtk::Real[mm*nn];
00423     } catch (std::bad_alloc &memory_allocation_exception) {
00424       std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00425         std::endl;
```

```
00426       std::cerr << memory_allocation_exception.what() << std::endl;
00427     }
00428     memset(data_transposed,
00429           mtk::kZero,
00430           sizeof(data_transposed[0])*mm*nn);
00431
00432     // Assign the values to their transposed position.
00433     std::swap(mm, nn);
00434     for (auto ii = 0; ii < mm; ++ii) {
00435       for (auto jj = 0; jj < nn; ++jj) {
00436         data_transposed[jj*mm + ii] = data_[ii*nn + jj];
00437       }
00438     }
00439     std::swap(mm, nn);
00440
00441     // Swap pointers.
00442     auto tmp = data_; // Temporal holder.
00443     data_ = data_transposed;
00444     delete [] tmp;
00445     tmp = nullptr;
00446
00447     matrix_properties_.set_ordering(mtk::ROW_MAJOR);
00448   }
00449 }
00450
00451 void mtk::DenseMatrix::OrderColMajor() {
00452
00453   if (matrix_properties_.ordering() == ROW_MAJOR) {
00454
00456
00457     mtk::Real *data_transposed{}; // Buffer.
00458
00459     int mm = matrix_properties_.num_rows(); // Used to construct this matrix.
00460     int nn = matrix_properties_.num_cols(); // Used to construct this matrix.
00461
00462     try {
00463       data_transposed = new mtk::Real[mm*nn];
00464     } catch (std::bad_alloc &memory_allocation_exception) {
00465       std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00466         std::endl;
00467       std::cerr << memory_allocation_exception.what() << std::endl;
00468     }
00469     memset(data_transposed,
00470           mtk::kZero,
00471           sizeof(data_transposed[0])*mm*nn);
00472
00473     // Assign the values to their transposed position.
00474     for (auto ii = 0; ii < mm; ++ii) {
00475       for (auto jj = 0; jj < nn; ++jj) {
00476         data_transposed[jj*mm + ii] = data_[ii*nn + jj];
00477       }
00478     }
00479
00480     // Swap pointers.
00481     auto tmp = data_; // Temporal holder.
00482     data_ = data_transposed;
00483     delete [] tmp;
00484     tmp = nullptr;
00485
00486     matrix_properties_.set_ordering(mtk::COL_MAJOR);
00487   }
00488 }
00489
00490 mtk::DenseMatrix mtk::DenseMatrix::Kron(const
      mtk::DenseMatrix &aa,
00491                                         const mtk::DenseMatrix &bb) {
00492
00493   int row_offset{}; // Offset for rows.
00494   int col_offset{}; // Offset for rows.
00495
00496   mtk::Real aa_factor{}; // Used in computation.
00497
00498   // Auxiliary variables:
00499   auto aux1 = aa.matrix_properties_.num_rows()*bb.
      matrix_properties_.num_rows();
00500   auto aux2 = aa.matrix_properties_.num_cols()*bb.
      matrix_properties_.num_cols();
00501
00502   mtk::DenseMatrix output(aux1,aux2); // Output matrix.
00503
00504   int kk_num_cols{output.matrix_properties_.num_cols()}; // Aux.
```

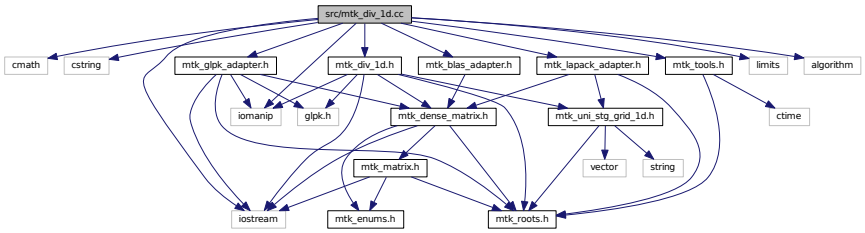
```
00505
00506   auto mm = aa.matrix_properties_.num_rows(); // Rows of aa.
00507   auto nn = aa.matrix_properties_.num_cols(); // Cols of aa.
00508   auto pp = bb.matrix_properties_.num_rows(); // Rows of bb.
00509   auto qq = bb.matrix_properties_.num_cols(); // Cols of bb.
00510
00511   for (auto ii = 0; ii < mm; ++ii) {
00512     row_offset = ii*pp;
00513     for (auto jj = 0; jj < nn; ++jj) {
00514       col_offset = jj*qq;
00515       aa_factor = aa.data_[ii*nn + jj];
00516       for (auto ll = 0; ll < pp; ++ll) {
00517         for (auto oo = 0; oo < qq; ++oo) {
00518           auto index = (ll + row_offset)*kk_num_cols + (oo + col_offset);
00519           output.data_[index] = aa_factor*bb.data_[ll*qq + oo];
00520         }
00521       }
00522     }
00523   }
00524
00525   output.matrix_properties_.set_storage(mtk::DENSE);
00526   output.matrix_properties_.set_ordering(
      mtk::ROW_MAJOR);
00527
00528   return output;
00529 }
00530
00531 bool mtk::DenseMatrix::WriteToFile(const std::string &filename) const {
00532
00533   std::ofstream output_dat_file;  // Output file.
00534
00535   output_dat_file.open(filename);
00536
00537   if (!output_dat_file.is_open()) {
00538     return false;
00539   }
00540
00541   int mm{matrix_properties_.num_rows()};
00542   int nn{matrix_properties_.num_cols()};
00543
00544   for (int ii = 0; ii < mm; ++ii) {
00545     int offset{ii*nn};
00546     for (int jj = 0; jj < nn; ++jj) {
00547       output_dat_file << ii << ' ' << jj << ' ' << data_[offset + jj] <<
00548         std::endl;
00549     }
00550   }
00551
00552   output_dat_file.close();
00553
00554   return true;
00555 }
```

17.61 src/mtk_div_1d.cc File Reference

Implements the class Div1D.

```
#include <cmath>
#include <cstring>
#include <iostream>
#include <iomanip>
#include <limits>
#include <algorithm>
#include "mtk_tools.h"
#include "mtk_blas_adapter.h"
#include "mtk_lapack_adapter.h"
#include "mtk_glpk_adapter.h"
#include "mtk_div_1d.h"
```

Include dependency graph for mtk_div_1d.cc:



Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

Functions

- std::ostream & mtk::operator<< (std::ostream &stream, mtk::Div1D &in)

17.61.1 Detailed Description

This class implements a 1D divergence matrix operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Todo Overload ostream operator as in mtk::Lap1D.

Todo Implement creation of ■ w. mtk::BLASAdapter.

Definition in file mtk_div_1d.cc.

17.62 mtk_div_1d.cc

```
00001
00015 /*
00016 Copyright (C) 2015, Computational Science Research Center, San Diego State
00017 University. All rights reserved.
00018
00019 Redistribution and use in source and binary forms, with or without modification,
00020 are permitted provided that the following conditions are met:
00021
00022 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00023 and a copy of the modified files should be reported once modifications are
00024 completed, unless these modifications are made through the project's GitHub
00025 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00026 should be developed and included in any deliverable.
00027
00028 2. Redistributions of source code must be done through direct
```

```
00029 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00030
00031 3. Redistributions in binary form must reproduce the above copyright notice,
00032 this list of conditions and the following disclaimer in the documentation and/or
00033 other materials provided with the distribution.
00034
00035 4. Usage of the binary form on proprietary applications shall require explicit
00036 prior written permission from the the copyright holders, and due credit should
00037 be given to the copyright holders.
00038
00039 5. Neither the name of the copyright holder nor the names of its contributors
00040 may be used to endorse or promote products derived from this software without
00041 specific prior written permission.
00042
00043 The copyright holders provide no reassurances that the source code provided does
00044 not infringe any patent, copyright, or any other intellectual property rights of
00045 third parties. The copyright holders disclaim any liability to any recipient for
00046 claims brought against recipient by any third party for infringement of that
00047 parties intellectual property rights.
00048
00049 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00050 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00051 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00052 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00053 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00054 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00055 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00056 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00057 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00058 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00059 */
00060
00061 #include <cmath>
00062 #include <cstring>
00063
00064 #include <iostream>
00065 #include <iomanip>
00066 #include <limits>
00067 #include <algorithm>
00068
00069 #include "mtk_tools.h"
00070
00071 #include "mtk_blas_adapter.h"
00072 #include "mtk_lapack_adapter.h"
00073 #include "mtk_glpk_adapter.h"
00074
00075 #include "mtk_div_1d.h"
00076
00077 namespace mtk {
00078
00079 std::ostream& operator <<(std::ostream &stream, mtk::Div1D &in) {
00080
00082
00083   stream << "divergence_[0] = " << std::setw(9) << in.divergence_[0] <<
00084     std::endl;
00085
00087
00088   stream << "divergence_[1:" << in.order_accuracy_ << "] = ";
00089   for (auto ii = 1; ii <= in.order_accuracy_; ++ii) {
00090     stream << std::setw(9) << in.divergence_[ii] << " ";
00091   }
00092   stream << std::endl;
00093
00094   if (in.order_accuracy_ > 2) {
00095
00097
00098     stream << "divergence_[" << in.order_accuracy_ + 1 << ":" <<
00099       2*in.order_accuracy_ << "] = ";
00100     for (auto ii = in.order_accuracy_ + 1; ii <= 2*in.
      order_accuracy_; ++ii) {
00101       stream << std::setw(9) << in.divergence_[ii] << " ";
00102     }
00103     stream << std::endl;
00104
00106
00107     auto offset = (2*in.order_accuracy_ + 1);
00108     int mm{};
00109     for (auto ii = 0; ii < in.dim_null_; ++ii) {
00110       stream << "divergence_[" << offset + mm << ":" <<
00111         offset + mm + in.num_bndy_coeffs_ - 1 << "] = ";
00112       for (auto jj = 0; jj < in.num_bndy_coeffs_; ++jj) {
```

```
00113         auto value = in.divergence_[offset + mm];
00114         stream << std::setw(9) << value << " ";
00115         ++mm;
00116       }
00117       stream << std::endl;
00118     }
00119   }
00120
00121   return stream;
00122 }
00123 }
00124
00125 mtk::Div1D::Div1D():
00126   order_accuracy_(mtk::kDefaultOrderAccuracy),
00127   dim_null_(),
00128   num_bndy_coeffs_(),
00129   divergence_length_(),
00130   minrow_(),
00131   row_(),
00132   coeffs_interior_(),
00133   prem_apps_(),
00134   weights_crs_(),
00135   weights_cbs_(),
00136   mim_bndy_(),
00137   divergence_(),
00138   mimetic_threshold_(mtk::kDefaultMimeticThreshold) {}
00139
00140 mtk::Div1D::Div1D(const Div1D &div):
00141   order_accuracy_(div.order_accuracy_),
00142   dim_null_(div.dim_null_),
00143   num_bndy_coeffs_(div.num_bndy_coeffs_),
00144   divergence_length_(div.divergence_length_),
00145   minrow_(div.minrow_),
00146   row_(div.row_),
00147   coeffs_interior_(div.coeffs_interior_),
00148   prem_apps_(div.prem_apps_),
00149   weights_crs_(div.weights_crs_),
00150   weights_cbs_(div.weights_cbs_),
00151   mim_bndy_(div.mim_bndy_),
00152   divergence_(div.divergence_),
00153   mimetic_threshold_(div.mimetic_threshold_) {}
00154
00155 mtk::Div1D::~Div1D() {
00156
00157   delete[] coeffs_interior_;
00158   coeffs_interior_ = nullptr;
00159
00160   delete[] prem_apps_;
00161   prem_apps_ = nullptr;
00162
00163   delete[] weights_crs_;
00164   weights_crs_ = nullptr;
00165
00166   delete[] weights_cbs_;
00167   weights_cbs_ = nullptr;
00168
00169   delete[] mim_bndy_;
00170   mim_bndy_ = nullptr;
00171
00172   delete[] divergence_;
00173   divergence_ = nullptr;
00174 }
00175
00176 bool mtk::Div1D::ConstructDiv1D(int order_accuracy,
00177                                 mtk::Real mimetic_threshold) {
00178
00179   #if MTK_DEBUG_LEVEL > 0
00180   mtk::Tools::Prevent(order_accuracy < 2, __FILE__, __LINE__, __func__);
00181   mtk::Tools::Prevent((order_accuracy%2) != 0, __FILE__, __LINE__, __func__);
00182   mtk::Tools::Prevent(mimetic_threshold <= mtk::kZero,
00183                       __FILE__, __LINE__, __func__);
00184
00185   if (order_accuracy >= mtk::kCriticalOrderAccuracyDiv) {
00186     std::cout << "WARNING: Numerical accuracy is critical." << std::endl;
00187   }
00188
00189   std::cout << "order_accuracy_ = " << order_accuracy << std::endl;
00190   std::cout << "mimetic_threshold_ = " << mimetic_threshold << std::endl;
00191   #endif
00192
00193   order_accuracy_ = order_accuracy;
```

```
00194   mimetic_threshold_ = mimetic_threshold;
00195
00197
00198   bool abort_construction = ComputeStencilInteriorGrid();
00199
00200   #if MTK_DEBUG_LEVEL > 0
00201   if (!abort_construction) {
00202     std::cerr << "Could NOT complete stage 1." << std::endl;
00203     std::cerr << "Exiting..." << std::endl;
00204     return false;
00205   }
00206   #endif
00207
00208   // At this point, we already have the values for the interior stencil stored
00209   // in the coeffs_interior_ array.
00210
00211   // It is noteworthy, that the 2nd-order-accurate divergence operator has NO
00212   // approximation at the boundary, thus it has no weights. For this case, the
00213   // dimension of the null-space of the Vandermonde matrices used to compute the
00214   // approximating coefficients at the boundary is 0. Ergo, we compute this
00215   // number first and then decide if we must compute anything at the boundary.
00216
00217   dim_null_ = order_accuracy_/2 - 1;
00218
00219   if (dim_null_ > 0) {
00220
00221     #ifdef MTK_PRECISION_DOUBLE
00222     num_bndy_coeffs_ = (int) (3.0*((mtk::Real) order_accuracy_)/2.0);
00223     #else
00224     num_bndy_coeffs_ = (int) (3.0f*((mtk::Real) order_accuracy_)/2.0f);
00225     #endif
00226
00228
00229     // For this we will follow recommendations given in:
00230     //
00231     // http://icl.cs.utk.edu/lapack-forum/viewtopic.php?f=5&t=4506
00232     //
00233     // We will compute the QR Factorization of the transpose, as in the
00234     // following (MATLAB) pseudo-code:
00235     //
00236     // [Q,R] = qr(V'); % Full QR as defined in
00237     // % http://www.stanford.edu/class/ee263/notes/qr_matlab.pdf
00238     //
00239     // null-space = Q(:, last (order_accuracy_/2 - 1) columns of Q );
00240     //
00241     // However, given the nature of the Vandermonde matrices we've just
00242     // computed, they all posses the same null-space. Therefore, we impose the
00243     // convention of computing the null-space of the first Vandermonde matrix
00244     // (west boundary).
00245
00246     abort_construction = ComputeRationalBasisNullSpace();
00247
00248     #if MTK_DEBUG_LEVEL > 0
00249     if (!abort_construction) {
00250       std::cerr << "Could NOT complete stage 2.1." << std::endl;
00251       std::cerr << "Exiting..." << std::endl;
00252       return false;
00253     }
00254     #endif
00255
00257
00258     abort_construction = ComputePreliminaryApproximations();
00259
00260     #if MTK_DEBUG_LEVEL > 0
00261     if (!abort_construction) {
00262       std::cerr << "Could NOT complete stage 2.2." << std::endl;
00263       std::cerr << "Exiting..." << std::endl;
00264       return false;
00265     }
00266     #endif
00267
00269
00270     abort_construction = ComputeWeights();
00271
00272     #if MTK_DEBUG_LEVEL > 0
00273     if (!abort_construction) {
00274       std::cerr << "Could NOT complete stage 2.3." << std::endl;
00275       std::cerr << "Exiting..." << std::endl;
00276       return false;
00277     }
00278     #endif
```

```
00279
00281
00282     abort_construction = ComputeStencilBoundaryGrid();
00283
00284     #if MTK_DEBUG_LEVEL > 0
00285     if (!abort_construction) {
00286       std::cerr << "Could NOT complete stage 2.4." << std::endl;
00287       std::cerr << "Exiting..." << std::endl;
00288       return false;
00289     }
00290     #endif
00291
00292   } // End of: if (dim_null_ > 0);
00293
00295
00296   // Once we have the following three collections of data:
00297   //   (a) the coefficients for the interior,
00298   //   (b) the coefficients for the boundary (if it applies),
00299   //   (c) and the weights (if it applies),
00300   // we will store everything in the output array:
00301
00302   abort_construction = AssembleOperator();
00303
00304   #if MTK_DEBUG_LEVEL > 0
00305   if (!abort_construction) {
00306     std::cerr << "Could NOT complete stage 3." << std::endl;
00307     std::cerr << "Exiting..." << std::endl;
00308     return false;
00309   }
00310   #endif
00311
00312   return true;
00313 }
00314
00315 int mtk::Div1D::num_bndy_coeffs() const {
00316
00317   return num_bndy_coeffs_;
00318 }
00319
00320 mtk::Real *mtk::Div1D::coeffs_interior() const {
00321
00322   return coeffs_interior_;
00323 }
00324
00325 mtk::Real *mtk::Div1D::weights_crs() const {
00326
00327   return weights_crs_;
00328 }
00329
00330 mtk::Real *mtk::Div1D::weights_cbs() const {
00331
00332
00333   return weights_cbs_;
00334 }
00335
00336 mtk::DenseMatrix mtk::Div1D::mim_bndy() const {
00337
00338   mtk::DenseMatrix xx(dim_null_, 3*order_accuracy_/2);
00339
00340   auto counter = 0;
00341   for (auto ii = 0; ii < dim_null_; ++ii) {
00342     for(auto jj = 0; jj < 3*order_accuracy_/2; ++jj) {
00343       xx.SetValue(ii,jj, divergence_[2*order_accuracy_ + 1 + counter]);
00344       counter++;
00345     }
00346   }
00347
00348   return xx;
00349 }
00350
00351 mtk::DenseMatrix mtk::Div1D::ReturnAsDenseMatrix(
00352   const UniStgGrid1D &grid) const {
00353
00354   int nn{grid.num_cells_x()}; // Number of cells on the grid.
00355
00356   #if MTK_DEBUG_LEVEL > 0
00357   mtk::Tools::Prevent(nn <= 0, __FILE__, __LINE__, __func__);
00358   mtk::Tools::Prevent(nn < 3*order_accuracy_ - 1, __FILE__, __LINE__, __func__);
00359   #endif
00360
00361   mtk::Real inv_delta_x{mtk::kOne/grid.delta_x()};
```

```
00362
00363   int dd_num_rows = nn + 2;
00364   int dd_num_cols = nn + 1;
00365   int elements_per_row = num_bndy_coeffs_;
00366   int num_extra_rows = dim_null_;
00367
00368   // Output matrix featuring sizes for divergence operators.
00369   mtk::DenseMatrix out(dd_num_rows, dd_num_cols);
00370
00372
00373   auto ee_index = 0;
00374   for (auto ii = 1; ii < num_extra_rows + 1; ii++) {
00375     auto cc = 0;
00376     for(auto jj = 0 ; jj < dd_num_rows; jj++) {
00377       if( cc >= elements_per_row) {
00378         out.SetValue(ii, jj, mtk::kZero);
00379       } else {
00380         out.SetValue(ii,jj, mim_bndy_[ee_index++]*inv_delta_x);
00381         cc++;
00382       }
00383     }
00384   }
00385
00387
00388   for (auto ii = num_extra_rows + 1;
00389        ii < dd_num_rows - num_extra_rows - 1; ii++) {
00390     auto jj = ii - num_extra_rows - 1;
00391     for (auto cc = 0; cc < order_accuracy_; cc++, jj++) {
00392       out.SetValue(ii, jj, coeffs_interior_[cc]*inv_delta_x);
00393     }
00394   }
00395
00397
00398   ee_index = 0;
00399   for (auto ii = dd_num_rows - 2; ii >= dd_num_rows - num_extra_rows - 1; ii--) {
00400     auto cc = 0;
00401     for (auto jj = dd_num_cols - 1; jj >= 0; jj--) {
00402       if( cc >= elements_per_row) {
00403         out.SetValue(ii,jj,0.0);
00404       } else {
00405         out.SetValue(ii,jj,-mim_bndy_[ee_index++]*inv_delta_x);
00406         cc++;
00407       }
00408      }
00409   }
00410
00411   return out;
00412 }
00413
00414 bool mtk::Div1D::ComputeStencilInteriorGrid() {
00415
00417
00418   mtk::Real* pp{}; // Spatial coordinates to create interior stencil.
00419
00420   try {
00421     pp = new mtk::Real[order_accuracy_];
00422   } catch (std::bad_alloc &memory_allocation_exception) {
00423     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00424       std::endl;
00425     std::cerr << memory_allocation_exception.what() << std::endl;
00426   }
00427   memset(pp, mtk::kZero, sizeof(pp[0])*order_accuracy_);
00428
00429   #ifdef MTK_PRECISION_DOUBLE
00430   pp[0] = 1.0/2.0 - ((mtk::Real) order_accuracy_)/2.0;
00431   #else
00432   pp[0] = 1.0f/2.0f - ((mtk::Real) order_accuracy_)/2.0f;
00433   #endif
00434
00435   for (auto ii = 1; ii < order_accuracy_; ++ii) {
00436     pp[ii] = pp[ii - 1] + mtk::kOne;
00437   }
00438
00439   #if MTK_DEBUG_LEVEL > 0
00440   std::cout << "pp =" << std::endl;
00441   for (auto ii = 0; ii < order_accuracy_; ++ii) {
00442     std::cout << std::setw(12) << pp[ii];
00443   }
00444   std::cout << std::endl << std::endl;
00445   #endif
00446
```

```
00448
00449   bool transpose{false};
00450
00451   mtk::DenseMatrix vander_matrix(pp,
00452                                  order_accuracy_,
00453                                  order_accuracy_,
00454                                  transpose);
00455
00456   #if MTK_DEBUG_LEVEL > 0
00457   std::cout << "vander_matrix = " << std::endl;
00458   std::cout << vander_matrix << std::endl;
00459   #endif
00460
00462
00463   try {
00464     coeffs_interior_ = new mtk::Real[order_accuracy_];
00465   } catch (std::bad_alloc &memory_allocation_exception) {
00466     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00467       std::endl;
00468     std::cerr << memory_allocation_exception.what() << std::endl;
00469   }
00470   memset(coeffs_interior_, mtk::kZero, sizeof(coeffs_interior_[0])*order_accuracy_);
00471
00472   coeffs_interior_[1] = mtk::kOne;
00473
00474   #if MTK_DEBUG_LEVEL > 0
00475   std::cout << "oo =" << std::endl;
00476   for (auto ii = 0; ii < order_accuracy_; ++ii) {
00477     std::cout << std::setw(12) << coeffs_interior_[ii] << std::endl;
00478   }
00479   std::cout << std::endl;
00480   #endif
00481
00483
00484   int info{mtk::LAPACKAdapter::SolveDenseSystem(vander_matrix,
00485                                                 coeffs_interior_)};
00486
00487   #if MTK_DEBUG_LEVEL > 0
00488   if (!info) {
00489     std::cout << "System solved! Interior stencil attained!" << std::endl;
00490     std::cout << std::endl;
00491   }
00492   else {
00493     std::cerr << "Something wrong solving system! info = " << info << std::endl;
00494     std::cerr << "Exiting..." << std::endl;
00495     return false;
00496   }
00497   #endif
00498
00499   #if MTK_DEBUG_LEVEL > 0
00500   std::cout << "coeffs_interior_ =" << std::endl;
00501   for (auto ii = 0; ii < order_accuracy_; ++ii) {
00502     std::cout << std::setw(12) << coeffs_interior_[ii];
00503   }
00504   std::cout << std::endl << std::endl;
00505   #endif
00506
00507   delete [] pp;
00508   pp = nullptr;
00509
00510   return true;
00511 }
00512
00513 bool mtk::Div1D::ComputeRationalBasisNullSpace(void) {
00514
00515   mtk::Real* gg{}; // Generator vector for the first Vandermonde matrix.
00516
00518
00519   try {
00520     gg = new mtk::Real[num_bndy_coeffs_];
00521   } catch (std::bad_alloc &memory_allocation_exception) {
00522     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00523       std::endl;
00524     std::cerr << memory_allocation_exception.what() << std::endl;
00525   }
00526   memset(gg, mtk::kZero, sizeof(gg[0])*num_bndy_coeffs_);
00527
00528   #ifdef MTK_PRECISION_DOUBLE
00529   gg[0] = -1.0/2.0;
00530   #else
00531   gg[0] = -1.0f/2.0f;
```

```
00532   #endif
00533   for (auto ii = 1; ii < num_bndy_coeffs_; ++ii) {
00534     gg[ii] = gg[ii - 1] + mtk::kOne;
00535   }
00536
00537   #if MTK_DEBUG_LEVEL > 0
00538   std::cout << "gg =" << std::endl;
00539   for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
00540     std::cout << std::setw(12) << gg[ii];
00541   }
00542   std::cout << std::endl << std::endl;
00543   #endif
00544
00546
00547   bool tran{true}; // Should I transpose the Vandermonde matrix.
00548
00549   mtk::DenseMatrix vv_west_t(gg, num_bndy_coeffs_, order_accuracy_ + 1, tran);
00550
00551   #if MTK_DEBUG_LEVEL > 0
00552   std::cout << "vv_west_t =" << std::endl;
00553   std::cout << vv_west_t << std::endl;
00554   #endif
00555
00557
00558   mtk::DenseMatrix qq_t(mtk::LAPACKAdapter::QRFactorDenseMatrix
      (vv_west_t));
00559
00560   #if MTK_DEBUG_LEVEL > 0
00561   std::cout << "QQ^T = " << std::endl;
00562   std::cout << qq_t << std::endl;
00563   #endif
00564
00566
00567   int KK_num_rows_{num_bndy_coeffs_};
00568   int KK_num_cols_{dim_null_};
00569
00570   mtk::DenseMatrix KK(KK_num_rows_, KK_num_cols_);
00571
00572   for (auto ii = num_bndy_coeffs_ - dim_null_; ii < num_bndy_coeffs_; ++ii) {
00573     for (auto jj = 0; jj < num_bndy_coeffs_; ++jj) {
00574       KK.data()[jj*dim_null_ + (ii - (num_bndy_coeffs_ - dim_null_))] =
00575           qq_t.data()[ii*num_bndy_coeffs_ + jj];
00576     }
00577   }
00578
00579   #if MTK_DEBUG_LEVEL > 0
00580   std::cout << "KK =" << std::endl;
00581   std::cout << KK << std::endl;
00582   std::cout << "KK.num_rows() = " << KK.num_rows() << std::endl;
00583   std::cout << "KK.num_cols() = " << KK.num_cols() << std::endl;
00584   std::cout << std::endl;
00585   #endif
00586
00588
00589   // Scale thus requesting that the last entries of the attained basis for the
00590   // null-space, adopt the pattern we require.
00591   // Essentially we will implement the following MATLAB pseudo-code:
00592   //  scalers = KK(num_bndy_approxs - (dim_null - 1):num_bndy_approxs,:)\B
00593   //  SK = KK*scalers
00594   // where SK is the scaled null-space.
00595
00596   // In this point, we almost have all the data we need correctly allocated
00597   // in memory. We will create the matrix II_, and elements we wish to scale in
00598   // the KK array. Using the concept of the leading dimension, we could just
00599   // use KK, with the correct leading dimension and that is it. BUT I DO NOT
00600   // GET how does it work. So I will just create a matrix with the content of
00601   // this array that we need, solve for the scalers and then scale the
00602   // whole KK:
00603
00604   // We will then create memory for that sub-matrix of KK (SUBK).
00605
00606   mtk::DenseMatrix SUBK(dim_null_,dim_null_);
00607
00608   for (auto ii = num_bndy_coeffs_ - dim_null_; ii < num_bndy_coeffs_; ++ii) {
00609     for (auto jj = 0; jj < dim_null_; ++jj) {
00610       SUBK.data()[(ii - (num_bndy_coeffs_ - dim_null_))*dim_null_ + jj] =
00611           KK.data()[ii*dim_null_ + jj];
00612     }
00613   }
00614
00615   #if MTK_DEBUG_LEVEL > 0
```

```
00616   std::cout << "SUBK =" << std::endl;
00617   std::cout << SUBK << std::endl;
00618   #endif
00619
00620   SUBK.Transpose();
00621
00622   #if MTK_DEBUG_LEVEL > 0
00623   std::cout << "SUBK^T =" << std::endl;
00624   std::cout << SUBK << std::endl;
00625   #endif
00626
00627   bool padded{false};
00628   tran = false;
00629
00630   mtk::DenseMatrix II(dim_null_, padded, tran);
00631
00632   #if MTK_DEBUG_LEVEL > 0
00633   std::cout << "II =" << std::endl;
00634   std::cout << II << std::endl;
00635   #endif
00636
00637   // Solve the system to compute the scalers.
00638   // An example of the system to solve, for k = 8, is:
00639   //
00640   // SUBK*scalers = II_ or
00641   //
00642   // |  0.386018  -0.0339244   -0.129478 |           | 1 0 0 |
00643   // | -0.119774   0.0199423   0.0558632 |*scalers = | 0 1 0 |
00644   // | 0.0155708 -0.00349546 -0.00853182 |           | 0 0 1 |
00645   //
00646   // Notice this is a nrhs = 3 system.
00647   // Noteworthy: we do NOT ACTUALLY ALLOCATE space for the scalers... they
00648   // will be stored in the created identity matrix.
00649   // Let us first transpose SUBK (because of LAPACK):
00650
00651   int info{mtk::LAPACKAdapter::SolveDenseSystem(SUBK, II)};
00652
00653   #if MTK_DEBUG_LEVEL > 0
00654   if (!info) {
00655     std::cout << "System successfully solved!" <<
00656       std::endl;
00657   } else {
00658     std::cerr << "Something went wrong solving system! info = " << info <<
00659       std::endl;
00660     std::cerr << "Exiting..." << std::endl;
00661     return false;
00662   }
00663   std::cout << std::endl;
00664   #endif
00665
00666   #if MTK_DEBUG_LEVEL > 0
00667   std::cout << "Computed scalers:" << std::endl;
00668   std::cout << II << std::endl;
00669   #endif
00670
00671   // Multiply the two matrices to attain a scaled basis for null-space.
00672
00673   rat_basis_null_space_ = mtk::BLASAdapter::RealDenseMM(KK, II);
00674
00675   #if MTK_DEBUG_LEVEL > 0
00676   std::cout << "Rational basis for the null-space:" << std::endl;
00677   std::cout << rat_basis_null_space_ << std::endl;
00678   #endif
00679
00680   // At this point, we have a rational basis for the null-space, with the
00681   // pattern we need! :)
00682
00683   delete [] gg;
00684   gg = nullptr;
00685
00686   return true;
00687 }
00688
00689 bool mtk::Div1D::ComputePreliminaryApproximations(void) {
00690
00692
00693   mtk::Real *gg{}; // Generator vector for the first approximation.
00694
00695   try {
00696     gg = new mtk::Real[num_bndy_coeffs_];
00697   } catch (std::bad_alloc &memory_allocation_exception) {
```

```
00698     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00699 std::endl;
00700     std::cerr << memory_allocation_exception.what() << std::endl;
00701   }
00702   memset(gg, mtk::kZero, sizeof(gg[0])*num_bndy_coeffs_);
00703
00704   #ifdef MTK_PRECISION_DOUBLE
00705   gg[0] = -1.0/2.0;
00706   #else
00707   gg[0] = -1.0f/2.0f;
00708   #endif
00709   for (auto ii = 1; ii < num_bndy_coeffs_; ++ii) {
00710     gg[ii] = gg[ii - 1] + mtk::kOne;
00711   }
00712
00713   #if MTK_DEBUG_LEVEL > 0
00714   std::cout << "gg0 =" << std::endl;
00715   for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
00716     std::cout << std::setw(12) << gg[ii];
00717   }
00718   std::cout << std::endl << std::endl;
00719   #endif
00720
00721   // Allocate 2D array to store the collection of preliminary approximations.
00722   try {
00723     prem_apps_ = new mtk::Real[num_bndy_coeffs_*dim_null_];
00724   } catch (std::bad_alloc &memory_allocation_exception) {
00725     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00726 std::endl;
00727     std::cerr << memory_allocation_exception.what() << std::endl;
00728   }
00729   memset(prem_apps_,
00730          mtk::kZero,
00731          sizeof(prem_apps_[0])*num_bndy_coeffs_*dim_null_);
00732
00734
00735   for (auto ll = 0; ll < dim_null_; ++ll) {
00736
00737     // Re-check new generator vector for every iteration except for the first.
00738     #if MTK_DEBUG_LEVEL > 0
00739     if (ll > 0) {
00740       std::cout << "gg" << ll << " =" << std::endl;
00741       for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
00742         std::cout << std::setw(12) << gg[ii];
00743       }
00744       std::cout << std::endl << std::endl;
00745     }
00746     #endif
00747
00749
00750     bool transpose{false};
00751
00752     mtk::DenseMatrix AA_(gg,
00753                          num_bndy_coeffs_, order_accuracy_ + 1,
00754                          transpose);
00755
00756     #if MTK_DEBUG_LEVEL > 0
00757     std::cout << "AA_" << ll << " =" << std::endl;
00758     std::cout << AA_ << std::endl;
00759     #endif
00760
00762
00763     mtk::Real *ob{};
00764
00765     auto ob_ld = num_bndy_coeffs_;
00766
00767     try {
00768       ob = new mtk::Real[ob_ld];
00769     } catch (std::bad_alloc &memory_allocation_exception) {
00770       std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00771         std::endl;
00772       std::cerr << memory_allocation_exception.what() << std::endl;
00773     }
00774     memset(ob, mtk::kZero, sizeof(ob[0])*ob_ld);
00775
00776     ob[1] = mtk::kOne;
00777
00778     #if MTK_DEBUG_LEVEL > 0
00779     std::cout << "ob = " << std::endl << std::endl;
00780     for (auto ii = 0; ii < ob_ld; ++ii) {
00781       std::cout << std::setw(12) << ob[ii] << std::endl;
```

```
00782     }
00783     std::cout << std::endl;
00784     #endif
00785
00787
00788     // However, this is an under-determined system of equations. So we can not
00789     // use the same LAPACK routine (dgesv_). We will instead use dgels_, through
00790     // our LAPACKAdapter class.
00791
00792     int info_{
00793       mtk::LAPACKAdapter::SolveRectangularDenseSystem(AA_,
      ob, ob_ld)};
00794
00795     #if MTK_DEBUG_LEVEL > 0
00796     if (!info_) {
00797       std::cout << "System successfully solved!" << std::endl << std::endl;
00798     } else {
00799       std::cerr << "Error solving system! info = " << info_ << std::endl;
00800     }
00801     #endif
00802
00803     #if MTK_DEBUG_LEVEL > 0
00804     std::cout << "ob =" << std::endl;
00805     for (auto ii = 0; ii < ob_ld; ++ii) {
00806       std::cout << std::setw(12) << ob[ii] << std::endl;
00807     }
00808     std::cout << std::endl;
00809     #endif
00810
00812
00813     // This implies a DAXPY operation. However, we must construct the arguments
00814     // for this operation.
00815
00817     // Save them into the ob_bottom array:
00818
00819     Real *ob_bottom{}; // Bottom part of the attained kernel used to scale it.
00820
00821     try {
00822       ob_bottom = new mtk::Real[dim_null_];
00823     } catch (std::bad_alloc &memory_allocation_exception) {
00824       std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00825         std::endl;
00826       std::cerr << memory_allocation_exception.what() << std::endl;
00827     }
00828     memset(ob_bottom, mtk::kZero, sizeof(ob_bottom[0])*dim_null_);
00829
00830     for (auto ii = 0; ii < dim_null_; ++ii) {
00831       ob_bottom[(dim_null_ - 1) - ii] = ob[num_bndy_coeffs_ - ii - 1];
00832     }
00833
00834     #if MTK_DEBUG_LEVEL > 0
00835     std::cout << "ob_bottom =" << std::endl;
00836     for (auto ii = 0; ii < dim_null_; ++ii) {
00837       std::cout << std::setw(12) << ob_bottom[ii] << std::endl;
00838     }
00839     std::cout << std::endl;
00840     #endif
00841
00843
00844     // We must computed an scaled ob, sob, using the scaled null-space in
00845     // rat_basis_null_space_.
00846     // Such operation is: sob = ob - rat_basis_null_space_*ob_bottom
00847     // or:                ob = -1.0*rat_basis_null_space_*ob_bottom + 1.0*ob
00848     // thus:               Y =    a*A    *x         +   b*Y (DAXPY).
00849
00850     #if MTK_DEBUG_LEVEL > 0
00851     std::cout << "Rational basis for the null-space:" << std::endl;
00852     std::cout << rat_basis_null_space_ << std::endl;
00853     #endif
00854
00855     mtk::Real alpha{-mtk::kOne};
00856     mtk::Real beta{mtk::kOne};
00857
00858     mtk::BLASAdapter::RealDenseMV(alpha, rat_basis_null_space_,
00859                                   ob_bottom, beta, ob);
00860
00861     #if MTK_DEBUG_LEVEL > 0
00862     std::cout << "scaled ob:" << std::endl;
00863     for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
00864       std::cout << std::setw(12) << ob[ii] << std::endl;
00865     }
```

```
00866     std::cout << std::endl;
00867     #endif
00868
00869     // We save the recently scaled solution, into an array containing these.
00870     // We can NOT start building the pi matrix, simply because I want that part
00871     // to be separated since its construction depends on the algorithm we want
00872     // to implement.
00873
00874     for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
00875       prem_apps_[ii*dim_null_ + ll] = ob[ii];
00876     }
00877
00878     // After the first iteration, simply shift the entries of the last
00879     // generator vector used:
00880     for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
00881       gg[ii]--;
00882     }
00883
00884     // Garbage collection for this loop:
00885     delete[] ob;
00886     ob = nullptr;
00887
00888     delete[] ob_bottom;
00889     ob_bottom = nullptr;
00890   } // End of: for (ll = 0; ll < dim_null; ll++);
00891
00892   #if MTK_DEBUG_LEVEL > 0
00893   std::cout << "Matrix post-scaled preliminary apps: " << std::endl;
00894   for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
00895     for (auto jj = 0; jj < dim_null_; ++jj) {
00896       std::cout << std::setw(12) << prem_apps_[ii*dim_null_ + jj];
00897     }
00898     std::cout << std::endl;
00899   }
00900   std::cout << std::endl;
00901   #endif
00902
00903   delete[] gg;
00904   gg = nullptr;
00905
00906   return true;
00907 }
00908
00909 bool mtk::Div1D::ComputeWeights(void) {
00910
00911   // Matrix to copmpute the weights as in the CRSA.
00912   mtk::DenseMatrix pi(num_bndy_coeffs_, num_bndy_coeffs_ - 1);
00913
00915
00916   // Assemble the pi matrix using:
00917   // 1. The collection of scaled preliminary approximations.
00918   // 2. The collection of coefficients approximating at the interior.
00919   // 3. The scaled basis for the null-space.
00920
00921   // 1.1. Process array of scaled preliminary approximations.
00922
00923   // These are queued in scaled_solutions. Each one of these, will be a column
00924   // of the pi matrix:
00925   for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
00926     for (auto jj = 0; jj < dim_null_; ++jj) {
00927       pi.data()[ii*(2*dim_null_ + (order_accuracy_/2 + 1)) + jj] =
00928         prem_apps_[ii*dim_null_ + jj];
00929     }
00930   }
00931
00932   // 1.2. Add columns from known stencil approximating at the interior.
00933
00934   // However, these must be padded by zeros, according to their position in the
00935   // final pi matrix:
00936   auto mm = 0;
00937   for (auto jj = dim_null_; jj < order_accuracy_; ++jj) {
00938     for (auto ii = 0; ii < order_accuracy_; ++ii) {
00939       pi.data()[(ii + mm)*(2*dim_null_ + (order_accuracy_/2 + 1)) + jj] =
00940         coeffs_interior_[ii];
00941     }
00942     ++mm;
00943   }
00944
00945   rat_basis_null_space_.OrderColMajor();
00946
00947   #if MTK_DEBUG_LEVEL > 0
```

```
00948   std::cout << "Rational basis for the null-space (col. major):" << std::endl;
00949   std::cout << rat_basis_null_space_ << std::endl;
00950   #endif
00951
00952   // 1.3. Add final set of columns: rational basis for null-space.
00953   for (auto jj = dim_null_ + (order_accuracy_/2 + 1); jj < num_bndy_coeffs_ - 1; ++jj) {
00954     for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
00955       auto og =
00956         (jj - (dim_null_ + (order_accuracy_/2 + 1)))*num_bndy_coeffs_ + ii;
00957       auto de = ii*(2*dim_null_ + (order_accuracy_/2 + 1)) + jj;
00958       pi.data()[de] = rat_basis_null_space_.data()[og];
00959     }
00960   }
00961
00962   #if MTK_DEBUG_LEVEL >0
00963   std::cout << "coeffs_interior_ =" << std::endl;
00964   for (auto ii = 0; ii < order_accuracy_; ++ii) {
00965     std::cout << std::setw(12) << coeffs_interior_[ii];
00966   }
00967   std::cout << std::endl << std::endl;
00968   #endif
00969
00970   #if MTK_DEBUG_LEVEL >0
00971   std::cout << "Constructed pi matrix for CRS Algorithm: " << std::endl;
00972   std::cout << pi << std::endl;
00973   #endif
00974
00976
00977   // This imposes the mimetic condition.
00978
00979   mtk::Real *hh{};  // Right-hand side to compute weights in the C{R,B}SA.
00980
00981   try {
00982     hh = new mtk::Real[num_bndy_coeffs_];
00983   } catch (std::bad_alloc &memory_allocation_exception) {
00984     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00985       std::endl;
00986     std::cerr << memory_allocation_exception.what() << std::endl;
00987   }
00988   memset(hh, mtk::kZero, sizeof(hh[0])*num_bndy_coeffs_);
00989
00990   hh[0] = -mtk::kOne;
00991   for (auto ii = (order_accuracy_/2 + 2 - 1); ii < num_bndy_coeffs_; ++ii) {
00992     auto aux_xx = mtk::kZero;
00993     for (auto jj = 0; jj < ((ii - (order_accuracy_/2 - 1)) - 1); ++jj) {
00994       aux_xx += coeffs_interior_[jj];
00995     }
00996     hh[ii] = -mtk::kOne*aux_xx;
00997   }
00998
01000
01001   // That is, we construct a system, to solve for the weights.
01002
01003   // Once again we face the challenge of solving with LAPACK. However, for the
01004   // CRSA, this matrix PI is over-determined, since it has more rows than
01005   // unknowns. However, according to the theory, the solution to this system is
01006   // unique. We will use dgels_.
01007
01008   try {
01009     weights_cbs_ = new mtk::Real[num_bndy_coeffs_];
01010   } catch (std::bad_alloc &memory_allocation_exception) {
01011     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
01012       std::endl;
01013     std::cerr << memory_allocation_exception.what() << std::endl;
01014   }
01015   memset(weights_cbs_, mtk::kZero, sizeof(weights_cbs_[0])*num_bndy_coeffs_);
01016
01017   int weights_ld{pi.num_cols() + 1};
01018
01019   // Preserve hh.
01020   std::copy(hh, hh + weights_ld, weights_cbs_);
01021
01022   pi.Transpose();
01023
01024   int info{mtk::LAPACKAdapter::SolveRectangularDenseSystem(
      pi, weights_cbs_, weights_ld)};
01025
01026   #if MTK_DEBUG_LEVEL > 0
01027   if (!info) {
01028     std::cout << "System successfully solved!" << std::endl << std::endl;
01029   } else {
```

```
01030     std::cerr << "Error solving system! info = " << info << std::endl;
01031   }
01032   #endif
01033
01034   #if MTK_DEBUG_LEVEL > 0
01035   std::cout << "hh =" << std::endl;
01036   for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
01037     std::cout << std::setw(11) << hh[ii] << std::endl;
01038   }
01039   std::cout << std::endl;
01040   #endif
01041
01042   // Preserve the original weights for research.
01043
01044   try {
01045     weights_crs_ = new mtk::Real[num_bndy_coeffs_];
01046   } catch (std::bad_alloc &memory_allocation_exception) {
01047     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
01048       std::endl;
01049     std::cerr << memory_allocation_exception.what() << std::endl;
01050   }
01051   memset(weights_crs_, mtk::kZero, sizeof(weights_crs_[0])*num_bndy_coeffs_);
01052
01053   std::copy(weights_cbs_, weights_cbs_ + (weights_ld - 1), weights_crs_);
01054
01055   #if MTK_DEBUG_LEVEL > 0
01056   std::cout << "weights_CRSA + lambda =" << std::endl;
01057   for (auto ii = 0; ii < weights_ld - 1; ++ii) {
01058     std::cout << std::setw(12) << weights_crs_[ii] << std::endl;
01059   }
01060   std::cout << std::endl;
01061   #endif
01062
01064
01065   if (order_accuracy_ >= mtk::kCriticalOrderAccuracyDiv) {
01066
01067     int minrow_{std::numeric_limits<int>::infinity()};
01068
01069     mtk::Real norm_{mtk::BLASAdapter::RealNRM2(weights_cbs_,
      order_accuracy_)};
01070     mtk::Real minnorm_{std::numeric_limits<mtk::Real>::infinity()};
01071
01073
01074     mtk::DenseMatrix phi(order_accuracy_ + 1, order_accuracy_);
01075
01076     for (auto ii = 0; ii < order_accuracy_ + 1; ++ii) {
01077       for (auto jj = 0; jj < dim_null_; ++jj) {
01078         phi.data()[ii*(order_accuracy_) + jj] = prem_apps_[ii*dim_null_ + jj];
01079       }
01080     }
01081
01082     int aux{};  // Auxiliary variable.
01083     for (auto jj = dim_null_; jj < dim_null_ + 2; ++jj) {
01084       for (auto ii = 0; ii < order_accuracy_; ++ii) {
01085         phi.data()[(ii + aux)*order_accuracy_ + jj] = coeffs_interior_[ii];
01086       }
01087       ++aux;
01088     }
01089
01090     for(auto jj=order_accuracy_ - 1; jj >=order_accuracy_ - dim_null_; jj--) {
01091       for(auto ii=0; ii<order_accuracy_ + 1; ++ii) {
01092         phi.data()[ii*order_accuracy_+jj] = mtk::kZero;
01093       }
01094     }
01095
01096     for (auto jj = 0; jj < order_accuracy_ + 1; ++jj) {
01097       for (auto ii = 0; ii < dim_null_; ++ii) {
01098         phi.data()[(ii + order_accuracy_ - dim_null_ + jj*order_accuracy_)] =
01099           -prem_apps_[(dim_null_ - ii - 1 + jj*dim_null_)];
01100       }
01101     }
01102
01103     for(auto ii = 0; ii < order_accuracy_/2; ++ii) {
01104       for (auto jj = dim_null_ + 2; jj < order_accuracy_; ++jj) {
01105         auto swap = phi.data()[ii*order_accuracy_+jj];
01106         phi.data()[ii*order_accuracy_ + jj] =
01107           phi.data()[(order_accuracy_-ii)*order_accuracy_+jj];
01108         phi.data()[(order_accuracy_-ii)*order_accuracy_+jj] = swap;
01109       }
01110     }
01111
```

```
01112     #if MTK_DEBUG_LEVEL > 0
01113     std::cout << "Constructed PHI matrix for CBS Algorithm: " << std::endl;
01114     std::cout << phi << std::endl;
01115     #endif
01116
01118
01119     mtk::Real *lamed{};  // Used to build big lambda.
01120
01121     try {
01122       lamed = new mtk::Real[dim_null_];
01123     } catch (std::bad_alloc &memory_allocation_exception) {
01124       std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
01125         std::endl;
01126       std::cerr << memory_allocation_exception.what() << std::endl;
01127     }
01128     memset(lamed, mtk::kZero, sizeof(lamed[0])*dim_null_);
01129
01130     for (auto ii = 0; ii < dim_null_; ++ii) {
01131       lamed[ii] = hh[ii + order_accuracy_ + 1] ;
01132     }
01133
01134     #if MTK_DEBUG_LEVEL > 0
01135     std::cout << "lamed =" << std::endl;
01136     for (auto ii = 0; ii < dim_null_; ++ii) {
01137       std::cout << std::setw(12) << lamed[ii] << std::endl;
01138     }
01139     std::cout << std::endl;
01140     #endif
01141
01142     for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
01143       mtk::Real temp = mtk::kZero;
01144       for(auto jj = 0; jj < dim_null_; ++jj) {
01145         temp = temp +
01146           lamed[jj]*rat_basis_null_space_.data()[jj*num_bndy_coeffs_ + ii];
01147       }
01148       hh[ii] = hh[ii] - temp;
01149     }
01150
01151     #if MTK_DEBUG_LEVEL > 0
01152     std::cout << "big_lambda =" << std::endl;
01153     for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
01154       std::cout << std::setw(12) << hh[ii] << std::endl;
01155     }
01156     std::cout << std::endl;
01157     #endif
01158
01159     int copy_result{};
01160
01161     mtk::Real normerr_; // Norm of the error for the solution on each row.
01162
01164
01165     for(auto row_= 0; row_ < order_accuracy_ + 1; ++row_) {
01166       normerr_ = mtk::GLPKAdapter::SolveSimplexAndCompare(phi.
      data(),
01167                                                           order_accuracy_ + 1,
01168                                                           order_accuracy_,
01169                                                           order_accuracy_,
01170                                                           hh,
01171                                                           weights_cbs_,
01172                                                           row_,
01173                                                           mimetic_threshold_,
01174                                                           copy_result);
01175       mtk::Real aux{normerr_/norm_};
01176
01177       #if MTK_DEBUG_LEVEL>0
01178       std::cout << "Relative norm: " << aux << " " << std::endl;
01179       std::cout << std::endl;
01180       #endif
01181
01182       if (aux < minnorm_) {
01183         minnorm_ = aux;
01184         minrow_= row_;
01185       }
01186     }
01187
01188     #if MTK_DEBUG_LEVEL > 0
01189     std::cout << "weights_CBSA + lambda (after brute force search):" <<
01190       std::endl;
01191     for (auto ii = 0; ii < num_bndy_coeffs_ - 1; ++ii) {
01192       std::cout << std::setw(12) << weights_cbs_[ii] << std::endl;
01193     }
```

```
01194     std::cout << std::endl;
01195     #endif
01196
01198
01199     // After we know which row yields the smallest relative norm that row is
01200     // chosen to be the objective function and the result of the optimizer is
01201     // chosen to be the new weights_.
01202
01203     #if MTK_DEBUG_LEVEL > 0
01204     std::cout << "Minimum Relative Norm " << minnorm_ << " found at row " <<
01205       minrow_ + 1 << std::endl;
01206     std::cout << std::endl;
01207     #endif
01208
01209     copy_result = 1;
01210     normerr_ = mtk::GLPKAdapter::SolveSimplexAndCompare(phi.
      data(),
01211                                                         order_accuracy_ + 1,
01212                                                         order_accuracy_,
01213                                                         order_accuracy_,
01214                                                         hh,
01215                                                         weights_cbs_,
01216                                                         minrow_,
01217                                                         mimetic_threshold_,
01218                                                         copy_result);
01219     mtk::Real aux_{normerr_/norm_};
01220     #if MTK_DEBUG_LEVEL > 0
01221     std::cout << "Relative norm: " << aux_ << std::endl;
01222     std::cout << std::endl;
01223     #endif
01224
01225     delete [] lamed;
01226     lamed = nullptr;
01227   }
01228
01229   delete [] hh;
01230   hh = nullptr;
01231
01232   return true;
01233 }
01234
01235 bool mtk::Div1D::ComputeStencilBoundaryGrid(void) {
01236
01237   #if MTK_DEBUG_LEVEL > 0
01238   std::cout << "weights_CBSA + lambda =" << std::endl;
01239   for (auto ii = 0; ii < num_bndy_coeffs_ - 1; ++ii) {
01240     std::cout << std::setw(12) << weights_cbs_[ii] << std::endl;
01241   }
01242   std::cout << std::endl;
01243   #endif
01244
01246
01247   mtk::Real *lambda{}; // Collection of bottom values from weights_.
01248
01249   try {
01250     lambda = new mtk::Real[dim_null_];
01251   } catch (std::bad_alloc &memory_allocation_exception) {
01252     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
01253       std::endl;
01254     std::cerr << memory_allocation_exception.what() << std::endl;
01255   }
01256   memset(lambda, mtk::kZero, sizeof(lambda[0])*dim_null_);
01257
01258   for (auto ii = 0; ii < dim_null_; ++ii) {
01259     lambda[ii] = weights_cbs_[order_accuracy_ + ii];
01260   }
01261
01262   #if MTK_DEBUG_LEVEL > 0
01263   std::cout << "lambda =" << std::endl;
01264   for (auto ii = 0; ii < dim_null_; ++ii) {
01265     std::cout << std::setw(12) << lambda[ii] << std::endl;
01266   }
01267   std::cout << std::endl;
01268   #endif
01269
01271
01272   mtk::Real *alpha{}; // Collection of alpha values.
01273
01274   try {
01275     alpha = new mtk::Real[dim_null_];
01276   } catch (std::bad_alloc &memory_allocation_exception) {
```

```
01277     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
01278       std::endl;
01279     std::cerr << memory_allocation_exception.what() << std::endl;
01280   }
01281   memset(alpha, mtk::kZero, sizeof(alpha[0])*dim_null_);
01282
01283   for (auto ii = 0; ii < dim_null_; ++ii) {
01284     alpha[ii] = lambda[ii]/weights_cbs_[ii] ;
01285   }
01286
01287   #if MTK_DEBUG_LEVEL > 0
01288   std::cout << "alpha =" << std::endl;
01289   for (auto ii = 0; ii < dim_null_; ++ii) {
01290     std::cout << std::setw(12) << alpha[ii] << std::endl;
01291   }
01292   std::cout << std::endl;
01293   #endif
01294
01296
01297   try {
01298     mim_bndy_ = new mtk::Real[num_bndy_coeffs_*dim_null_];
01299   } catch (std::bad_alloc &memory_allocation_exception) {
01300     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
01301       std::endl;
01302     std::cerr << memory_allocation_exception.what() << std::endl;
01303   }
01304   memset(mim_bndy_, mtk::kZero, sizeof(mim_bndy_[0])*num_bndy_coeffs_*dim_null_);
01305
01306   for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
01307     for (auto jj = 0; jj < dim_null_; ++jj) {
01308       mim_bndy_[ii*dim_null_ + jj] =
01309         prem_apps_[ii*dim_null_ + jj] +
01310         alpha[jj]*rat_basis_null_space_.data()[jj*num_bndy_coeffs_ + ii];
01311     }
01312   }
01313
01314   #if MTK_DEBUG_LEVEL >0
01315   std::cout << "Collection of mimetic approximations:" << std::endl;
01316   for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
01317     for (auto jj = 0; jj < dim_null_; ++jj) {
01318       std::cout << std::setw(13) << mim_bndy_[ii*dim_null_ + jj];
01319     }
01320     std::cout << std::endl;
01321   }
01322   std::cout << std::endl;
01323   #endif
01324
01325   delete[] lambda;
01326   lambda = nullptr;
01327
01328   delete[] alpha;
01329   alpha = nullptr;
01330
01331   return true;
01332 }
01333
01334 bool mtk::Div1D::AssembleOperator(void) {
01335
01336   // The output array will have this form:
01337   // 1. The first entry of the array will contain the used order order_accuracy_.
01338   // 2. The second entry of the array will contain the collection of
01339   // approximating coefficients for the interior of the grid.
01340   // 3. IF order_accuracy_ > 2, then the third entry will contain a collection of weights.
01341   // 4. IF order_accuracy_ > 2, the next dim_null_ entries will contain the collections of
01342   // approximating coefficients for the west boundary of the grid.
01343
01344   if (order_accuracy_ > mtk::kDefaultOrderAccuracy) {
01345     divergence_length_ =
01346       1 + order_accuracy_ + order_accuracy_ + dim_null_*num_bndy_coeffs_;
01347   } else {
01348     divergence_length_ = 1 + order_accuracy_;
01349   }
01350
01351   #if MTK_DEBUG_LEVEL > 0
01352   std::cout << "divergence_length_ = " << divergence_length_ << std::endl;
01353   #endif
01354
01355   try {
01356     divergence_ = new double[divergence_length_];
01357   } catch (std::bad_alloc &memory_allocation_exception) {
01358     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
```

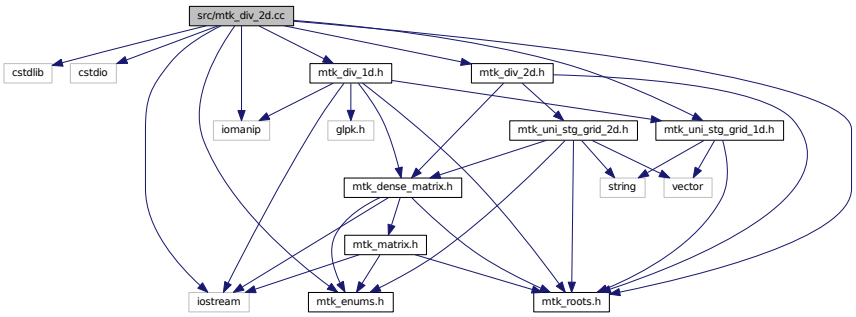
```
01359       std::endl;
01360     std::cerr << memory_allocation_exception.what() << std::endl;
01361   }
01362   memset(divergence_, mtk::kZero, sizeof(divergence_[0])*divergence_length_);
01363
01365
01366   divergence_[0] = order_accuracy_;
01367
01369
01370   for (auto ii = 0; ii < order_accuracy_; ++ii) {
01371   divergence_[ii + 1] = coeffs_interior_[ii];
01372   }
01373
01375
01376   if (order_accuracy_ > 2) {
01377     for (auto ii = 0; ii < order_accuracy_; ++ii) {
01378       divergence_[(1 + order_accuracy_) + ii] = weights_cbs_[ii];
01379     }
01380   }
01381
01384
01385   if (order_accuracy_ > 2) {
01386     auto offset = (2*order_accuracy_ + 1);
01387     int mm{};
01388     for (auto ii = 0; ii < dim_null_; ++ii) {
01389       for (auto jj = 0; jj < num_bndy_coeffs_; ++jj) {
01390         divergence_[offset + (mm)] = mim_bndy_[jj*dim_null_ + ii];
01391         ++mm;
01392       }
01393     }
01394   }
01395
01396   #if MTK_DEBUG_LEVEL > 0
01397   std::cout << "1D " << order_accuracy_ << "-order div built!" << std::endl;
01398   std::cout << std::endl;
01399   #endif
01400
01401   return true;
01402 }
```

17.63 src/mtk_div_2d.cc File Reference

Implements the class Div2D.

```
#include <cstdlib>
#include <cstdio>
#include <iostream>
#include <iomanip>
#include "mtk_roots.h"
#include "mtk_enums.h"
#include "mtk_uni_stg_grid_1d.h"
#include "mtk_div_1d.h"
#include "mtk_div_2d.h"
```

Include dependency graph for mtk_div_2d.cc:



17.63.1 Detailed Description

This class implements a 2D divergence matrix operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_div_2d.cc.

17.64 mtk_div_2d.cc

```
00001
00011 /*
00012 Copyright (C) 2015, Computational Science Research Center, San Diego State
00013 University. All rights reserved.
00014
00015 Redistribution and use in source and binary forms, with or without modification,
00016 are permitted provided that the following conditions are met:
00017
00018 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00019 and a copy of the modified files should be reported once modifications are
00020 completed, unless these modifications are made through the project's GitHub
00021 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00022 should be developed and included in any deliverable.
00023
00024 2. Redistributions of source code must be done through direct
00025 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00026
00027 3. Redistributions in binary form must reproduce the above copyright notice,
00028 this list of conditions and the following disclaimer in the documentation and/or
00029 other materials provided with the distribution.
00030
00031 4. Usage of the binary form on proprietary applications shall require explicit
00032 prior written permission from the the copyright holders, and due credit should
00033 be given to the copyright holders.
00034
00035 5. Neither the name of the copyright holder nor the names of its contributors
00036 may be used to endorse or promote products derived from this software without
00037 specific prior written permission.
00038
00039 The copyright holders provide no reassurances that the source code provided does
00040 not infringe any patent, copyright, or any other intellectual property rights of
00041 third parties. The copyright holders disclaim any liability to any recipient for
00042 claims brought against recipient by any third party for infringement of that
00043 parties intellectual property rights.
```

```
00044
00045 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00046 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00047 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00048 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00049 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00050 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00051 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00052 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00053 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00054 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00055 */
00056
00057 #include <cstdlib>
00058 #include <cstdio>
00059
00060 #include <iostream>
00061 #include <iomanip>
00062
00063 #include "mtk_roots.h"
00064 #include "mtk_enums.h"
00065 #include "mtk_uni_stg_grid_1d.h"
00066 #include "mtk_div_1d.h"
00067 #include "mtk_div_2d.h"
00068
00069 mtk::Div2D::Div2D():
00070   order_accuracy_(),
00071   mimetic_threshold_() {}
00072
00073 mtk::Div2D::Div2D(const Div2D &div):
00074   order_accuracy_(div.order_accuracy_),
00075   mimetic_threshold_(div.mimetic_threshold_) {}
00076
00077 mtk::Div2D::~Div2D() {}
00078
00079 bool mtk::Div2D::ConstructDiv2D(const
      mtk::UniStgGrid2D &grid,
00080                                 int order_accuracy,
00081                                 mtk::Real mimetic_threshold) {
00082
00083   int num_cells_x = grid.num_cells_x();
00084   int num_cells_y = grid.num_cells_y();
00085
00086   int mx = num_cells_x + 2;  // Dx vertical dimension.
00087   int nx = num_cells_x + 1;  // Dx horizontal dimension.
00088   int my = num_cells_y + 2;  // Dy vertical dimension.
00089   int ny = num_cells_y + 1;  // Dy horizontal dimension.
00090
00091   mtk::Div1D div;
00092
00093   bool info = div.ConstructDiv1D(order_accuracy, mimetic_threshold);
00094
00095   if (!info) {
00096     std::cerr << "Mimetic div could not be built." << std::endl;
00097     return info;
00098   }
00099
00100   auto west = grid.west_bndy();
00101   auto east = grid.east_bndy();
00102   auto south = grid.south_bndy();
00103   auto north = grid.east_bndy();
00104
00105   mtk::UniStgGrid1D grid_x(west, east, num_cells_x);
00106   mtk::UniStgGrid1D grid_y(south, north, num_cells_y);
00107
00108   mtk::DenseMatrix dx(div.ReturnAsDenseMatrix(grid_x));
00109   mtk::DenseMatrix dy(div.ReturnAsDenseMatrix(grid_y));
00110
00111   bool padded{true};
00112   bool transpose{false};
00113
00114   mtk::DenseMatrix ix(num_cells_x, padded, transpose);
00115   mtk::DenseMatrix iy(num_cells_y, padded, transpose);
00116
00117   mtk::DenseMatrix dxy(mtk::DenseMatrix::Kron(iy, dx));
00118   mtk::DenseMatrix dyx(mtk::DenseMatrix::Kron(dy, ix));
00119
00120   #if MTK_DEBUG_LEVEL > 0
00121   std::cout << "Dx: " << mx << " by " << nx << std::endl;
00122   std::cout << "Iy : " << num_cells_y<< " by " << ny  << std::endl;
00123   std::cout << "Dy: " << my << " by " << ny << std::endl;
```

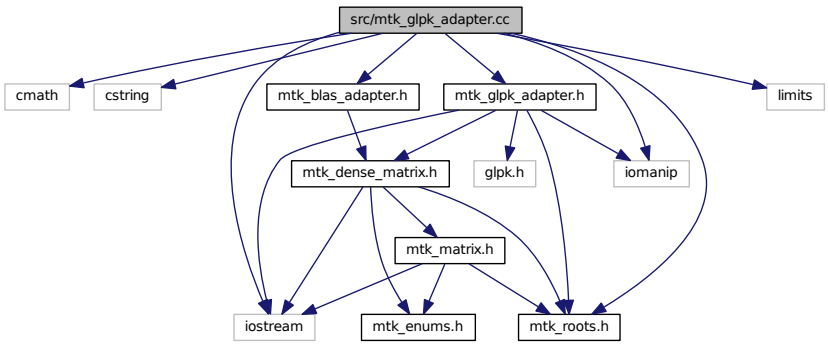
```
00124   std::cout << "Ix : " << num_cells_x<< " by " << nx  << std::endl;
00125   std::cout << "Div 2D: " << mx*num_cells_y + my*num_cells_x << " by " <<
00126     nx*ny <<std::endl;
00127   #endif
00128
00129   mtk::DenseMatrix d2d(mx*my, nx*num_cells_y + ny*num_cells_x);
00130
00131   for (auto ii = 0; ii < mx*my; ii++) {
00132     for (auto jj = 0; jj < nx*num_cells_y; jj++) {
00133       d2d.SetValue(ii, jj, dxy.GetValue(ii,jj));
00134     }
00135     for(auto kk=0; kk<ny*num_cells_x; kk++) {
00136       d2d.SetValue(ii, kk + nx*num_cells_y, dyx.GetValue(ii, kk));
00137     }
00138   }
00139
00140   divergence_ = d2d;
00141
00142   return info;
00143 }
00144
00145 mtk::DenseMatrix mtk::Div2D::ReturnAsDenseMatrix() const {
00146
00147   return divergence_;
00148 }
```

17.65 src/mtk_glpk_adapter.cc File Reference

Adapter class for the GLPK API.

```
#include <cmath>
#include <cstring>
#include <iostream>
#include <iomanip>
#include <limits>
#include "mtk_roots.h"
#include "mtk_blas_adapter.h"
#include "mtk_glpk_adapter.h"
```

Include dependency graph for mtk_glpk_adapter.cc:



17.65.1 Detailed Description

This class contains a collection of static classes, that posses direct access to the underlying structure of the matrices, thus allowing programmers to exploit some of the numerical methods implemented in the GLPK.

The **GLPK (GNU Linear Programming Kit)** package is intended for solving large-scale linear programming (LP), mixed integer programming (MIP), and other related problems. It is a set of routines written in ANSI C and organized in the form of a callable library.

See also

http://www.gnu.org/software/glpk/

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_glpk_adapter.cc.

17.66 mtk_glpk_adapter.cc

```
00001
00019 /*
00020 Copyright (C) 2015, Computational Science Research Center, San Diego State
00021 University. All rights reserved.
00022
00023 Redistribution and use in source and binary forms, with or without modification,
00024 are permitted provided that the following conditions are met:
00025
00026 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00027 and a copy of the modified files should be reported once modifications are
00028 completed, unless these modifications are made through the project's GitHub
00029 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00030 should be developed and included in any deliverable.
00031
00032 2. Redistributions of source code must be done through direct
00033 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00034
00035 3. Redistributions in binary form must reproduce the above copyright notice,
00036 this list of conditions and the following disclaimer in the documentation and/or
00037 other materials provided with the distribution.
00038
00039 4. Usage of the binary form on proprietary applications shall require explicit
00040 prior written permission from the the copyright holders, and due credit should
00041 be given to the copyright holders.
00042
00043 5. Neither the name of the copyright holder nor the names of its contributors
00044 may be used to endorse or promote products derived from this software without
00045 specific prior written permission.
00046
00047 The copyright holders provide no reassurances that the source code provided does
00048 not infringe any patent, copyright, or any other intellectual property rights of
00049 third parties. The copyright holders disclaim any liability to any recipient for
00050 claims brought against recipient by any third party for infringement of that
00051 parties intellectual property rights.
00052
00053 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00054 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00055 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00056 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00057 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00058 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00059 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00060 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00061 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00062 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00063 */
00064
00065 #include <cmath>
00066 #include <cstring>
00067
00068 #include <iostream>
00069 #include <iomanip>
00070 #include <limits>
00071
00072 #include "mtk_roots.h"
```

```
00073 #include "mtk_blas_adapter.h"
00074 #include "mtk_glpk_adapter.h"
00075
00076 mtk::Real mtk::GLPKAdapter::SolveSimplexAndCompare(
      mtk::Real *A,
00077                                                   int nrows,
00078                                                   int ncols,
00079                                                   int kk,
00080                                                   mtk::Real *hh,
00081                                                   mtk::Real *qq,
00082                                                   int robjective,
00083                                                   mtk::Real mimetic_threshold,
00084                                                   int copy) {
00085
00086   #if MTK_DEBUG_LEVEL > 0
00087   char mps_file_name[18]; // File name for the MPS files.
00088   #endif
00089   char rname[5];          // Row name.
00090   char cname[5];          // Column name.
00091
00092   glp_prob *lp; // Linear programming problem.
00093
00094   int *ia;  // Array for the problem.
00095   int *ja;  // Array for the problem.
00096
00097   int problem_size; // Size of the problem.
00098   int lp_nrows;     // Number of rows.
00099   int lp_ncols;     // Number of columns.
00100   int matsize;      // Size of the matrix.
00101   int glp_index{1}; // Index of the objective function.
00102   int ii;           // Iterator.
00103   int jj;           // Iterator.
00104
00105   mtk::Real *ar;            // Array for the problem.
00106   mtk::Real *objective;     // Array containing the objective function.
00107   mtk::Real *rhs;           // Array containing the rhs.
00108   mtk::Real *err;           // Array of errors.
00109
00110   mtk::Real xl;             // Norm-2 of the error.
00111
00112   #if MTK_DEBUG_LEVEL > 0
00113   mtk::Real obj_value;      // Value of the objective function.
00114   #endif
00115
00116   lp_nrows = kk;
00117   lp_ncols = kk;
00118
00119   matsize = lp_nrows*lp_ncols;
00120
00122
00124   problem_size = lp_nrows*lp_ncols + 1;
00125
00126   try {
00127     ia = new int[problem_size];
00128   } catch (std::bad_alloc &memory_allocation_exception) {
00129     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00130       std::endl;
00131     std::cerr << memory_allocation_exception.what() << std::endl;
00132   }
00133   memset(ia, 0, sizeof(ia[0])*problem_size);
00134
00135   try {
00136     ja = new int[problem_size];
00137   } catch (std::bad_alloc &memory_allocation_exception) {
00138     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00139       std::endl;
00140     std::cerr << memory_allocation_exception.what() << std::endl;
00141   }
00142   memset(ja, 0, sizeof(ja[0])*problem_size);
00143
00144   try {
00145     ar = new mtk::Real[problem_size];
00146   } catch (std::bad_alloc &memory_allocation_exception) {
00147     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00148       std::endl;
00149     std::cerr << memory_allocation_exception.what() << std::endl;
00150   }
00151   memset(ar, mtk::kZero, sizeof(ar[0])*problem_size);
00152
00153   try {
00154     objective = new mtk::Real[lp_ncols + 1];
```

```
00155   } catch (std::bad_alloc &memory_allocation_exception) {
00156     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00157       std::endl;
00158     std::cerr << memory_allocation_exception.what() << std::endl;
00159   }
00160   memset(objective, mtk::kZero, sizeof(objective[0])*(lp_ncols + 1));
00161
00162   try {
00163     rhs = new mtk::Real[lp_nrows + 1];
00164   } catch (std::bad_alloc &memory_allocation_exception) {
00165     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00166       std::endl;
00167     std::cerr << memory_allocation_exception.what() << std::endl;
00168   }
00169   memset(rhs, mtk::kZero, sizeof(rhs[0])*(lp_nrows + 1));
00170
00171   try {
00172     err = new mtk::Real[lp_nrows];
00173   } catch (std::bad_alloc &memory_allocation_exception) {
00174     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00175       std::endl;
00176     std::cerr << memory_allocation_exception.what() << std::endl;
00177   }
00178   memset(err, mtk::kZero, sizeof(err[0])*(lp_nrows));
00179
00180   #if MTK_DEBUG_LEVEL > 0
00181   std::cout << "Problem size: " << problem_size << std::endl;
00182   std::cout << "lp_nrows = " << lp_nrows << std::endl;
00183   std::cout << "lp_ncols = " << lp_ncols << std::endl;
00184   std::cout << std::endl;
00185   #endif
00186
00187   lp = glp_create_prob();
00188
00189   glp_set_prob_name (lp, "mtk::GLPKAdapter::Simplex");
00190
00191   glp_set_obj_dir (lp, GLP_MIN);
00192
00194
00195   glp_add_rows(lp, lp_nrows);
00196
00197   for (ii = 1; ii <= lp_nrows; ++ii) {
00198     sprintf(rname, "R%02d",ii);
00199     glp_set_row_name(lp, ii, rname);
00200   }
00201
00202   glp_add_cols(lp, lp_ncols);
00203
00204   for (ii = 1; ii <= lp_ncols; ++ii) {
00205     sprintf(cname, "Q%02d",ii);
00206     glp_set_col_name (lp, ii, cname);
00207   }
00208
00210
00211   #if MTK_DEBUG_LEVEL>0
00212   std::cout << "Using row " << robjective + 1 << " as objective." << std::endl;
00213   #endif
00214   for (jj = 0; jj < kk; ++jj) {
00215     objective[glp_index] = A[jj + robjective * ncols];
00216     glp_index++;
00217   }
00218   #if MTK_DEBUG_LEVEL >0
00219   std::cout << std::endl;
00220   #endif
00221
00223
00224   glp_index = 1;
00225   rhs[0] = mtk::kZero;
00226   for (ii = 0; ii <= lp_nrows; ++ii) {
00227     if (ii != robjective) {
00228       rhs[glp_index] = hh[ii];
00229       glp_set_row_bnds(lp, glp_index, GLP_UP, 0.0, rhs[glp_index]);
00230       glp_index++;
00231     }
00232   }
00233
00234   #if MTK_DEBUG_LEVEL > 0
00235   std::cout << "rhs =" << std::endl;
00236   for (auto ii = 0; ii < lp_nrows; ++ii) {
00237     std::cout << std::setw(15) << rhs[ii] << std::endl;
00238   }
```

```
00239   std::cout << std::endl;
00240   #endif
00241
00243
00244   for (ii = 1; ii <= lp_ncols; ++ii) {
00245     glp_set_obj_coef (lp, ii, objective[ii]);
00246   }
00247
00249
00250   for (ii = 1; ii <= lp_ncols; ++ii) {
00251     glp_set_col_bnds (lp, ii, GLP_LO, mimetic_threshold, 0.0);
00252   }
00253
00255
00256   glp_index = 1;
00257   for (ii = 0; ii <= kk; ++ii) {
00258     for (jj = 0; jj < kk; ++jj) {
00259       if (ii != robjective) {
00260         ar[glp_index] = A[jj + ii * ncols];
00261         glp_index++;
00262       }
00263     }
00264   }
00265
00266   glp_index = 0;
00267
00268   for (ii = 1; ii < problem_size; ++ii) {
00269     if (((ii - 1) % lp_ncols) == 0) {
00270       glp_index++;
00271     }
00272     ia[ii] = glp_index;
00273     ja[ii] = (ii - 1) % lp_ncols + 1;
00274   }
00275
00276   glp_load_matrix (lp, matsize, ia, ja, ar);
00277
00278   #if MTK_DEBUG_LEVEL > 0
00279   sprintf(mps_file_name, "LP_MPS_row_%02d.mps", robjective);
00280   glp_write_mps(lp, GLP_MPS_FILE, nullptr, mps_file_name);
00281   #endif
00282
00284
00285   glp_simplex (lp, nullptr);
00286
00287   // Check status of the solution.
00288
00289   if (glp_get_status(lp) == GLP_OPT) {
00290
00291     for(ii = 1; ii <= lp_ncols; ++ii) {
00292       err[ii - 1] = qq[ii - 1] - glp_get_col_prim(lp,ii);
00293     }
00294
00295     #if MTK_DEBUG_LEVEL > 0
00296     obj_value = glp_get_obj_val (lp);
00297     std::cout << std::setw(12) << "CBS" << std::setw(12) << "CRS" << std::endl;
00298     for (ii = 0; ii < lp_ncols; ++ii) {
00299       std::cout << "q_" << ii + 1 << " = " << std::setw(12) <<
00300         glp_get_col_prim(lp,ii + 1) << std::setw(12) << qq[ii] << std::endl;
00301     }
00302     std::cout << "Objective function value (row " << robjective + 1 << ") = " <<
00303       obj_value << std::endl;
00304     #endif
00305
00306     if (copy) {
00307       for(ii = 0; ii < lp_ncols; ++ii) {
00308         qq[ii] = glp_get_col_prim(lp,ii + 1);
00309       }
00310       // Preserve the bottom values of qq.
00311     }
00312
00313     x1 = mtk::BLASAdapter::RealNRM2(err,lp_ncols);
00314
00315   } else {
00316     x1 = std::numeric_limits<mtk::Real>::infinity();
00317   }
00318
00319   glp_delete_prob (lp);
00320   glp_free_env ();
00321
00322   delete [] ia;
00323   delete [] ja;
```

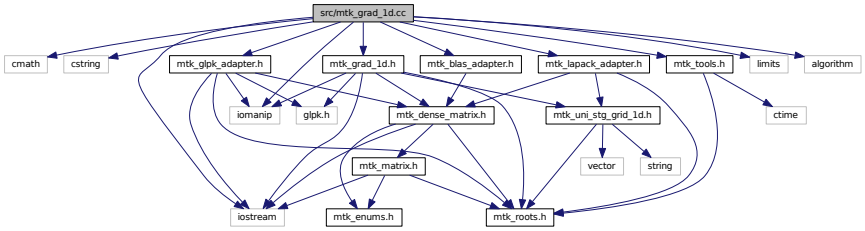
```
00324   delete [] ar;
00325   delete [] objective;
00326   delete [] rhs;
00327   delete [] err;
00328
00329   return x1;
00330 }
```

17.67 src/mtk_grad_1d.cc File Reference

Implements the class Grad1D.

```
#include <cmath>
#include <cstring>
#include <iostream>
#include <iomanip>
#include <limits>
#include <algorithm>
#include "mtk_tools.h"
#include "mtk_blas_adapter.h"
#include "mtk_lapack_adapter.h"
#include "mtk_glpk_adapter.h"
#include "mtk_grad_1d.h"
```

Include dependency graph for mtk_grad_1d.cc:



Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

Functions

- std::ostream & mtk::operator<< (std::ostream &stream, mtk::Grad1D &in)

17.67.1 Detailed Description

This class implements a 1D gradient matrix operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Todo Overload ostream operator as in mtk::Lap1D.

Todo Implement creation of ■ w. mtk::BLASAdapter.

Definition in file mtk_grad_1d.cc.

17.68 mtk_grad_1d.cc

```
00001
00015 /*
00016 Copyright (C) 2015, Computational Science Research Center, San Diego State
00017 University. All rights reserved.
00018
00019 Redistribution and use in source and binary forms, with or without modification,
00020 are permitted provided that the following conditions are met:
00021
00022 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00023 and a copy of the modified files should be reported once modifications are
00024 completed, unless these modifications are made through the project's GitHub
00025 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00026 should be developed and included in any deliverable.
00027
00028 2. Redistributions of source code must be done through direct
00029 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00030
00031 3. Redistributions in binary form must reproduce the above copyright notice,
00032 this list of conditions and the following disclaimer in the documentation and/or
00033 other materials provided with the distribution.
00034
00035 4. Usage of the binary form on proprietary applications shall require explicit
00036 prior written permission from the the copyright holders, and due credit should
00037 be given to the copyright holders.
00038
00039 5. Neither the name of the copyright holder nor the names of its contributors
00040 may be used to endorse or promote products derived from this software without
00041 specific prior written permission.
00042
00043 The copyright holders provide no reassurances that the source code provided does
00044 not infringe any patent, copyright, or any other intellectual property rights of
00045 third parties. The copyright holders disclaim any liability to any recipient for
00046 claims brought against recipient by any third party for infringement of that
00047 parties intellectual property rights.
00048
00049 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00050 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00051 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00052 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00053 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00054 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00055 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00056 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00057 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00058 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00059 */
00060
00061 #include <cmath>
00062 #include <cstring>
00063
00064 #include <iostream>
00065 #include <iomanip>
00066 #include <limits>
00067 #include <algorithm>
00068
00069 #include "mtk_tools.h"
00070
00071 #include "mtk_blas_adapter.h"
00072 #include "mtk_lapack_adapter.h"
00073 #include "mtk_glpk_adapter.h"
00074
```

```
00075 #include "mtk_grad_1d.h"
00076
00077 namespace mtk {
00078
00079 std::ostream& operator <<(std::ostream &stream, mtk::Grad1D &in) {
00080
00082
00083   stream << "gradient_[0] = " << std::setw(9) << in.gradient_[0] << std::endl;
00084
00086
00087   stream << "gradient_[1:" << in.order_accuracy_ << "] = ";
00088   for (auto ii = 1; ii <= in.order_accuracy_; ++ii) {
00089     stream << std::setw(9) << in.gradient_[ii] << " ";
00090   }
00091   stream << std::endl;
00092
00094
00095   stream << "gradient_[" << in.order_accuracy_ + 1 << ":" <<
00096     2*in.order_accuracy_ << "] = ";
00097   for (auto ii = in.order_accuracy_ + 1; ii <= 2*in.
      order_accuracy_; ++ii) {
00098     stream << std::setw(9) << in.gradient_[ii] << " ";
00099   }
00100   stream << std::endl;
00101
00103
00104   int offset{2*in.order_accuracy_ + 1};
00105   int mm {};
00106
00107   stream << "gradient_[" << offset + mm << ":" <<
00108     offset + mm + in.num_bndy_coeffs_ - 1 << "] = ";
00109
00110   if (in.order_accuracy_ > mtk::kDefaultOrderAccuracy) {
00111     for (auto ii = 0; ii < in.num_bndy_approxs_ ; ++ii) {
00112       for (auto jj = 0; jj < in.num_bndy_coeffs_; jj++) {
00113         auto value = in.gradient_[offset + (mm)];
00114         stream << std::setw(9) << value << " ";
00115         mm++;
00116       }
00117     }
00118   } else {
00119     stream << std::setw(9) << in.gradient_[offset + 0] << ' ';
00120     stream << std::setw(9) << in.gradient_[offset + 1] << ' ';
00121     stream << std::setw(9) << in.gradient_[offset + 2] << ' ';
00122   }
00123   stream << std::endl;
00124
00125   return stream;
00126 }
00127 }
00128
00129 mtk::Grad1D::Grad1D():
00130   order_accuracy_(mtk::kDefaultOrderAccuracy),
00131   dim_null_(),
00132   num_bndy_approxs_(),
00133   num_bndy_coeffs_(),
00134   gradient_length_(),
00135   minrow_(),
00136   row_(),
00137   coeffs_interior_(),
00138   prem_apps_(),
00139   weights_crs_(),
00140   weights_cbs_(),
00141   mim_bndy_(),
00142   gradient_(),
00143   mimetic_threshold_(mtk::kDefaultMimeticThreshold) {}
00144
00145 mtk::Grad1D::Grad1D(const Grad1D &grad):
00146   order_accuracy_(grad.order_accuracy_),
00147   dim_null_(grad.dim_null_),
00148   num_bndy_approxs_(grad.num_bndy_approxs_),
00149   num_bndy_coeffs_(grad.num_bndy_coeffs_),
00150   gradient_length_(grad.gradient_length_),
00151   minrow_(grad.minrow_),
00152   row_(grad.row_),
00153   coeffs_interior_(grad.coeffs_interior_),
00154   prem_apps_(grad.prem_apps_),
00155   weights_crs_(grad.weights_crs_),
00156   weights_cbs_(grad.weights_cbs_),
00157   mim_bndy_(grad.mim_bndy_),
00158   gradient_(grad.gradient_),
```

```
00159   mimetic_threshold_(grad.mimetic_threshold_) {}
00160
00161 mtk::Grad1D::~Grad1D() {
00162
00163   delete[] coeffs_interior_;
00164   coeffs_interior_ = nullptr;
00165
00166   delete[] prem_apps_;
00167   prem_apps_ = nullptr;
00168
00169   delete[] weights_crs_;
00170   weights_crs_ = nullptr;
00171
00172   delete[] weights_cbs_;
00173   weights_cbs_ = nullptr;
00174
00175   delete[] mim_bndy_;
00176   mim_bndy_ = nullptr;
00177
00178   delete[] gradient_;
00179   gradient_ = nullptr;
00180 }
00181
00182 bool mtk::Grad1D::ConstructGrad1D(int order_accuracy,
      Real mimetic_threshold) {
00183
00184   #if MTK_DEBUG_LEVEL > 0
00185   mtk::Tools::Prevent(order_accuracy < 2, __FILE__, __LINE__, __func__);
00186   mtk::Tools::Prevent((order_accuracy%2) != 0, __FILE__, __LINE__, __func__);
00187   mtk::Tools::Prevent(mimetic_threshold <= mtk::kZero,
00188                       __FILE__, __LINE__, __func__);
00189
00190   if (order_accuracy >= mtk::kCriticalOrderAccuracyGrad) {
00191     std::cout << "WARNING: Numerical accuracy is high." << std::endl;
00192   }
00193
00194   std::cout << "order_accuracy_ = " << order_accuracy << std::endl;
00195   std::cout << "mimetic_threshold_ = " << mimetic_threshold << std::endl;
00196   #endif
00197
00198   order_accuracy_ = order_accuracy;
00199   mimetic_threshold_ = mimetic_threshold;
00200
00202
00203   bool abort_construction = ComputeStencilInteriorGrid();
00204
00205   #if MTK_DEBUG_LEVEL > 0
00206   if (!abort_construction) {
00207     std::cerr << "Could NOT complete stage 1." << std::endl;
00208     std::cerr << "Exiting..." << std::endl;
00209     return false;
00210   }
00211   #endif
00212
00213   // At this point, we already have the values for the interior stencil stored
00214   // in the coeffs_interior_ array.
00215
00216   dim_null_ = order_accuracy_/2 - 1;
00217
00218   num_bndy_approxs_ = dim_null_ + 1;
00219
00220   #ifdef MTK_PRECISION_DOUBLE
00221   num_bndy_coeffs_ = (int) (3.0*((mtk::Real) order_accuracy_)/2.0);
00222   #else
00223   num_bndy_coeffs_ = (int) (3.0f*((mtk::Real) order_accuracy_)/2.0f);
00224   #endif
00225
00227
00228   // For this we will follow recommendations given in:
00229   //
00230   // http://icl.cs.utk.edu/lapack-forum/viewtopic.php?f=5&t=4506
00231   //
00232   // We will compute the QR Factorization of the transpose, as in the
00233   // following (MATLAB) pseudo-code:
00234   //
00235   // [Q,R] = qr(V'); % Full QR as defined in
00236   // % http://www.stanford.edu/class/ee263/notes/qr_matlab.pdf
00237   //
00238   // null-space = Q(:, last (order_accuracy_/2 - 1) columns of Q );
00239   //
00240   // However, given the nature of the Vandermonde matrices we've just
```

```
00241   // computed, they all posses the same null-space. Therefore, we impose the
00242   // convention of computing the null-space of the first Vandermonde matrix
00243   // (west boundary).
00244
00245   // In the case of the gradient, the first Vandermonde system has a unique
00246   // solution for the case of second-order-accuracy. Ergo, the Vandermonde
00247   // matrix used to assemble said system, will have an empty null-space.
00248
00249   // Therefore, we only compute a rational basis for the case of order higher
00250   // than second.
00251
00252   if (dim_null_ > 0) {
00253
00254     abort_construction = ComputeRationalBasisNullSpace();
00255
00256     #if MTK_DEBUG_LEVEL > 0
00257     if (!abort_construction) {
00258       std::cerr << "Could NOT complete stage 2.1." << std::endl;
00259       std::cerr << "Exiting..." << std::endl;
00260       return false;
00261     }
00262     #endif
00263   }
00264
00266
00267   abort_construction = ComputePreliminaryApproximations();
00268
00269   #if MTK_DEBUG_LEVEL > 0
00270   if (!abort_construction) {
00271     std::cerr << "Could NOT complete stage 2.2." << std::endl;
00272     std::cerr << "Exiting..." << std::endl;
00273     return false;
00274   }
00275   #endif
00276
00278
00279   abort_construction = ComputeWeights();
00280
00281   #if MTK_DEBUG_LEVEL > 0
00282   if (!abort_construction) {
00283     std::cerr << "Could NOT complete stage 2.3." << std::endl;
00284     std::cerr << "Exiting..." << std::endl;
00285     return false;
00286   }
00287   #endif
00288
00290
00291   if (dim_null_ > 0) {
00292
00293     abort_construction = ComputeStencilBoundaryGrid();
00294
00295     #if MTK_DEBUG_LEVEL > 0
00296     if (!abort_construction) {
00297       std::cerr << "Could NOT complete stage 2.4." << std::endl;
00298       std::cerr << "Exiting..." << std::endl;
00299       return false;
00300     }
00301     #endif
00302   }
00303
00305
00306   // Once we have the following three collections of data:
00307   //   (a) the coefficients for the interior,
00308   //   (b) the coefficients for the boundary (if it applies),
00309   //   (c) and the weights (if it applies),
00310   // we will store everything in the output array:
00311
00312   abort_construction = AssembleOperator();
00313
00314   #if MTK_DEBUG_LEVEL > 0
00315   if (!abort_construction) {
00316     std::cerr << "Could NOT complete stage 3." << std::endl;
00317     std::cerr << "Exiting..." << std::endl;
00318     return false;
00319   }
00320   #endif
00321
00322   return true;
00323 }
00324
00325 int mtk::Grad1D::num_bndy_coeffs() const {
```

```
00326
00327   return num_bndy_coeffs_;
00328 }
00329
00330 mtk::Real *mtk::Grad1D::coeffs_interior() const {
00331
00332   return coeffs_interior_;
00333 }
00334
00335 mtk::Real *mtk::Grad1D::weights_crs() const {
00336
00337   return weights_crs_;
00338 }
00339
00340 mtk::Real *mtk::Grad1D::weights_cbs() const {
00341
00342   return weights_cbs_;
00343 }
00344
00345 mtk::DenseMatrix mtk::Grad1D::mim_bndy() const {
00346
00347   mtk::DenseMatrix xx(dim_null_, 3*order_accuracy_/2);
00348
00349   auto counter = 0;
00350   for (auto ii = 0; ii < dim_null_; ++ii) {
00351     for(auto jj = 0; jj < 3*order_accuracy_/2; ++jj) {
00352       xx.SetValue(ii,jj, gradient_[2*order_accuracy_ + 1 + counter]);
00353       counter++;
00354     }
00355   }
00356
00357   return xx;
00358 }
00359
00360 mtk::DenseMatrix mtk::Grad1D::ReturnAsDenseMatrix(
    mtk::Real west,
00361                                                   mtk::Real east,
00362                                                   int num_cells_x) const {
00363
00364   int nn{num_cells_x}; // Number of cells on the grid.
00365
00366   #if MTK_DEBUG_LEVEL > 0
00367   mtk::Tools::Prevent(east < west, __FILE__, __LINE__, __func__);
00368   mtk::Tools::Prevent(nn < 3*order_accuracy_ - 2, __FILE__, __LINE__, __func__);
00369   mtk::Tools::Prevent(nn <= 0, __FILE__, __LINE__, __func__);
00370   #endif
00371
00372   mtk::Real delta_x = (east - west)/((mtk::Real) num_cells_x);
00373
00374   mtk::Real inv_delta_x{mtk::kOne/delta_x};
00375
00376   int gg_num_rows = nn + 1;
00377   int gg_num_cols = nn + 2;
00378   int elements_per_row = num_bndy_coeffs_;
00379   int num_extra_rows = order_accuracy_/2;
00380
00381   // Output matrix featuring sizes for gradient operators.
00382   mtk::DenseMatrix out(gg_num_rows, gg_num_cols);
00383
00385
00386   auto ee_index = 0;
00387   for (auto ii = 0; ii < num_extra_rows; ii++) {
00388     auto cc = 0;
00389     for(auto jj = 0 ; jj < gg_num_cols; jj++) {
00390       if(cc >= elements_per_row) {
00391         out.SetValue(ii, jj, mtk::kZero);
00392       } else {
00393         out.SetValue(ii,jj,
00394                      gradient_[2*order_accuracy_ + 1 + ee_index++]*inv_delta_x);
00395         cc++;
00396       }
00397     }
00398   }
00399
00401
00402   for (auto ii = num_extra_rows; ii < gg_num_rows - num_extra_rows; ii++) {
00403     auto jj = ii - num_extra_rows + 1;
00404     for (auto cc = 0; cc < order_accuracy_; cc++, jj++) {
00405       out.SetValue(ii, jj, coeffs_interior_[cc]*inv_delta_x);
00406     }
00407   }
```

```
00408
00410
00411   ee_index = 0;
00412   for (auto ii = gg_num_rows - 1; ii >= gg_num_rows - num_extra_rows; ii--) {
00413     auto cc = 0;
00414     for (auto jj = gg_num_cols - 1; jj >= 0; jj--) {
00415       if(cc >= elements_per_row) {
00416         out.SetValue(ii,jj,mtk::kZero);
00417       } else {
00418         out.SetValue(ii,jj,
00419                      -gradient_[2*order_accuracy_ + 1 + ee_index++]*inv_delta_x);
00420         cc++;
00421       }
00422      }
00423   }
00424
00425   return out;
00426 }
00427
00428 mtk::DenseMatrix mtk::Grad1D::ReturnAsDenseMatrix(
00429   const UniStgGrid1D &grid) const {
00430
00431   int nn{grid.num_cells_x()}; // Number of cells on the grid.
00432
00433   #if MTK_DEBUG_LEVEL > 0
00434   mtk::Tools::Prevent(nn <= 0, __FILE__, __LINE__, __func__);
00435   mtk::Tools::Prevent(nn < 3*order_accuracy_ - 2, __FILE__, __LINE__, __func__);
00436   #endif
00437
00438   mtk::Real inv_delta_x{mtk::kOne/grid.delta_x()};
00439
00440   int gg_num_rows = nn + 1;
00441   int gg_num_cols = nn + 2;
00442   int elements_per_row = num_bndy_coeffs_;
00443   int num_extra_rows = order_accuracy_/2;
00444
00445   // Output matrix featuring sizes for gradient operators.
00446   mtk::DenseMatrix out(gg_num_rows, gg_num_cols);
00447
00449
00450   auto ee_index = 0;
00451   for (auto ii = 0; ii < num_extra_rows; ii++) {
00452     auto cc = 0;
00453     for(auto jj = 0 ; jj < gg_num_cols; jj++) {
00454       if(cc >= elements_per_row) {
00455         out.SetValue(ii, jj, mtk::kZero);
00456       } else {
00457         out.SetValue(ii,jj,
00458                      gradient_[2*order_accuracy_ + 1 + ee_index++]*inv_delta_x);
00459         cc++;
00460       }
00461     }
00462   }
00463
00465
00466   for (auto ii = num_extra_rows; ii < gg_num_rows - num_extra_rows; ii++) {
00467     auto jj = ii - num_extra_rows + 1;
00468     for (auto cc = 0; cc < order_accuracy_; cc++, jj++) {
00469       out.SetValue(ii, jj, coeffs_interior_[cc]*inv_delta_x);
00470     }
00471   }
00472
00474
00475   ee_index = 0;
00476   for (auto ii = gg_num_rows - 1; ii >= gg_num_rows - num_extra_rows; ii--) {
00477     auto cc = 0;
00478     for (auto jj = gg_num_cols - 1; jj >= 0; jj--) {
00479       if(cc >= elements_per_row) {
00480         out.SetValue(ii,jj,mtk::kZero);
00481       } else {
00482         out.SetValue(ii,jj,
00483                      -gradient_[2*order_accuracy_ + 1 + ee_index++]*inv_delta_x);
00484         cc++;
00485       }
00486      }
00487   }
00488
00489   return out;
00490 }
00491
00492 mtk::DenseMatrix mtk::Grad1D::ReturnAsDimensionlessDenseMatrix
```

```
    (
00493   int num_cells_x) const {
00494
00495   int nn{num_cells_x}; // Number of cells on the grid.
00496
00497   #if MTK_DEBUG_LEVEL > 0
00498   mtk::Tools::Prevent(nn <= 0, __FILE__, __LINE__, __func__);
00499   mtk::Tools::Prevent(nn < 3*order_accuracy_ - 2, __FILE__, __LINE__, __func__);
00500   #endif
00501
00502   int gg_num_rows = nn + 1;
00503   int gg_num_cols = nn + 2;
00504   int elements_per_row = num_bndy_coeffs_;
00505   int num_extra_rows = order_accuracy_/2;
00506
00507   // Output matrix featuring sizes for gradient operators.
00508   mtk::DenseMatrix out(gg_num_rows, gg_num_cols);
00509
00511
00512   auto ee_index = 0;
00513   for (auto ii = 0; ii < num_extra_rows; ii++) {
00514     auto cc = 0;
00515     for(auto jj = 0 ; jj < gg_num_cols; jj++) {
00516       if(cc >= elements_per_row) {
00517         out.SetValue(ii, jj, mtk::kZero);
00518       } else {
00519         out.SetValue(ii,jj,
00520                      gradient_[2*order_accuracy_ + 1 + ee_index++]);
00521         cc++;
00522       }
00523     }
00524   }
00525
00527
00528   for (auto ii = num_extra_rows; ii < gg_num_rows - num_extra_rows; ii++) {
00529     auto jj = ii - num_extra_rows + 1;
00530     for (auto cc = 0; cc < order_accuracy_; cc++, jj++) {
00531       out.SetValue(ii, jj, coeffs_interior_[cc]);
00532     }
00533   }
00534
00536
00537   ee_index = 0;
00538   for (auto ii = gg_num_rows - 1; ii >= gg_num_rows - num_extra_rows; ii--) {
00539     auto cc = 0;
00540     for (auto jj = gg_num_cols - 1; jj >= 0; jj--) {
00541       if(cc >= elements_per_row) {
00542         out.SetValue(ii,jj,mtk::kZero);
00543       } else {
00544         out.SetValue(ii,jj,
00545                      -gradient_[2*order_accuracy_ + 1 + ee_index++]);
00546         cc++;
00547       }
00548      }
00549   }
00550
00551   return out;
00552 }
00553
00554 bool mtk::Grad1D::ComputeStencilInteriorGrid() {
00555
00557
00558   mtk::Real* pp{}; // Spatial coordinates to create interior stencil.
00559
00560   try {
00561     pp = new mtk::Real[order_accuracy_];
00562   } catch (std::bad_alloc &memory_allocation_exception) {
00563     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00564       std::endl;
00565     std::cerr << memory_allocation_exception.what() << std::endl;
00566   }
00567   memset(pp, mtk::kZero, sizeof(pp[0])*order_accuracy_);
00568
00569   #ifdef MTK_PRECISION_DOUBLE
00570   pp[0] = 1.0/2.0 - ((mtk::Real) order_accuracy_)/2.0;
00571   #else
00572   pp[0] = 1.0f/2.0f - ((mtk::Real) order_accuracy_)/2.0f;
00573   #endif
00574
00575   for (auto ii = 1; ii < order_accuracy_; ++ii) {
00576     pp[ii] = pp[ii - 1] + mtk::kOne;
```

```
00577   }
00578
00579   #if MTK_DEBUG_LEVEL > 0
00580   std::cout << "pp =" << std::endl;
00581   for (auto ii = 0; ii < order_accuracy_; ++ii) {
00582     std::cout << std::setw(12) << pp[ii];
00583   }
00584   std::cout << std::endl << std::endl;
00585   #endif
00586
00588
00589   bool transpose{false};
00590
00591   mtk::DenseMatrix vander_matrix(pp,order_accuracy_,order_accuracy_,transpose);
00592
00593   #if MTK_DEBUG_LEVEL > 0
00594   std::cout << "vander_matrix = " << std::endl;
00595   std::cout << vander_matrix << std::endl << std::endl;
00596   #endif
00597
00599
00600   try {
00601     coeffs_interior_ = new mtk::Real[order_accuracy_];
00602   } catch (std::bad_alloc &memory_allocation_exception) {
00603     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00604       std::endl;
00605     std::cerr << memory_allocation_exception.what() << std::endl;
00606   }
00607   memset(coeffs_interior_, mtk::kZero, sizeof(coeffs_interior_[0])*order_accuracy_);
00608
00609   coeffs_interior_[1] = mtk::kOne;
00610
00611   #if MTK_DEBUG_LEVEL > 0
00612   std::cout << "oo =" << std::endl;
00613   for (auto ii = 0; ii < order_accuracy_; ++ii) {
00614     std::cout << std::setw(12) << coeffs_interior_[ii] << std::endl;
00615   }
00616   std::cout << std::endl;
00617   #endif
00618
00620
00621   int info{mtk::LAPACKAdapter::SolveDenseSystem(vander_matrix,
00622                                                 coeffs_interior_)};
00623
00624   #if MTK_DEBUG_LEVEL > 0
00625   if (!info) {
00626     std::cout << "System solved! Interior stencil attained!" << std::endl;
00627     std::cout << std::endl;
00628   }
00629   else {
00630     std::cerr << "Something wrong solving system! info = " << info << std::endl;
00631     std::cerr << "Exiting..." << std::endl;
00632     return false;
00633   }
00634   #endif
00635
00636   #if MTK_DEBUG_LEVEL > 0
00637   std::cout << "coeffs_interior_ =" << std::endl;
00638   for (auto ii = 0; ii < order_accuracy_; ++ii) {
00639     std::cout << std::setw(12) << coeffs_interior_[ii];
00640   }
00641   std::cout << std::endl << std::endl;
00642   #endif
00643
00644   delete [] pp;
00645   pp = nullptr;
00646
00647   return true;
00648 }
00649
00650 bool mtk::Grad1D::ComputeRationalBasisNullSpace(void) {
00651
00653
00654   mtk::Real* gg{}; // Generator vector for the first Vandermonde matrix.
00655
00656   try {
00657     gg = new mtk::Real[num_bndy_coeffs_];
00658   } catch (std::bad_alloc &memory_allocation_exception) {
00659     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00660       std::endl;
00661     std::cerr << memory_allocation_exception.what() << std::endl;
```

```
00662   }
00663   memset(gg, mtk::kZero, sizeof(gg[0])*num_bndy_coeffs_);
00664
00665   #ifdef MTK_PRECISION_DOUBLE
00666   gg[1] = 1.0/2.0;
00667   #else
00668   gg[1] = 1.0f/2.0f;
00669   #endif
00670   for (auto ii = 2; ii < num_bndy_coeffs_; ++ii) {
00671     gg[ii] = gg[ii - 1] + mtk::kOne;
00672   }
00673
00674   #if MTK_DEBUG_LEVEL > 0
00675   std::cout << "gg =" << std::endl;
00676   for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
00677     std::cout << std::setw(12) << gg[ii];
00678   }
00679   std::cout << std::endl << std::endl;
00680   #endif
00681
00683
00684   bool tran{true}; // Should I transpose the Vandermonde matrix.
00685
00686   mtk::DenseMatrix aa_west_t(gg, num_bndy_coeffs_, order_accuracy_ + 1, tran);
00687
00688   #if MTK_DEBUG_LEVEL > 0
00689   std::cout << "aa_west_t =" << std::endl;
00690   std::cout << aa_west_t << std::endl;
00691   #endif
00692
00694
00695   mtk::DenseMatrix qq_t(mtk::LAPACKAdapter::QRFactorDenseMatrix
      (aa_west_t));
00696
00697   #if MTK_DEBUG_LEVEL > 0
00698   std::cout << "qq_t = " << std::endl;
00699   std::cout << qq_t << std::endl;
00700   #endif
00701
00703
00704   int kk_num_rows{num_bndy_coeffs_};
00705   int kk_num_cols{dim_null_};
00706
00707   mtk::DenseMatrix kk(kk_num_rows, kk_num_cols);
00708
00709   // In the case of the gradient, even though we must solve for a null-space
00710   // of dimension 2, we must only extract ONE basis for the kernel.
00711   // We perform this extraction here:
00712
00713   int aux_{kk_num_rows - kk_num_cols};
00714   for (auto ii = kk_num_rows - kk_num_cols; ii < kk_num_rows; ii++) {
00715     aux_--;
00716     for (auto jj = 0; jj < kk_num_rows; jj++) {
00717       kk.data()[jj*kk_num_cols + (kk_num_rows - kk_num_cols - aux_ - 1)] =
00718         qq_t.data()[ii*num_bndy_coeffs_ + jj];
00719     }
00720   }
00721
00722   #if MTK_DEBUG_LEVEL > 0
00723   std::cout << "kk =" << std::endl;
00724   std::cout << kk << std::endl;
00725   std::cout << "kk.num_rows() = " << kk.num_rows() << std::endl;
00726   std::cout << "kk.num_cols() = " << kk.num_cols() << std::endl;
00727   std::cout << std::endl;
00728   #endif
00729
00731
00732   // Scale thus requesting that the last entries of the attained basis for the
00733   // null-space, adopt the pattern we require.
00734   // Essentially we will implement the following MATLAB pseudo-code:
00735   //  scalers = kk(num_bndy_approxs - (dim_null - 1):num_bndy_approxs,:)\B
00736   //  SK = kk*scalers
00737   // where SK is the scaled null-space.
00738
00739   // In this point, we almost have all the data we need correctly allocated
00740   // in memory. We will create the matrix iden_, and elements we wish to scale in
00741   // the kk array. Using the concept of the leading dimension, we could just
00742   // use kk, with the correct leading dimension and that is it. BUT I DO NOT
00743   // GET how does it work. So I will just create a matrix with the content of
00744   // this array that we need, solve for the scalers and then scale the
00745   // whole kk:
```

```
00746
00747   // We will then create memory for that sub-matrix of kk (subk).
00748
00749   mtk::DenseMatrix subk(dim_null_, dim_null_);
00750
00751   auto zz = 0;
00752   for (auto ii = order_accuracy_ + 1; ii < num_bndy_coeffs_; ii++) {
00753     for (auto jj = 0; jj < dim_null_; jj++) {
00754       subk.data()[zz*(dim_null_) + jj] = kk.data()[ii*(dim_null_) + jj];
00755     }
00756     zz++;
00757   }
00758
00759   #if MTK_DEBUG_LEVEL > 0
00760   std::cout << "subk =" << std::endl;
00761   std::cout << subk << std::endl;
00762   #endif
00763
00764   subk.Transpose();
00765
00766   #if MTK_DEBUG_LEVEL > 0
00767   std::cout << "subk_t =" << std::endl;
00768   std::cout << subk << std::endl;
00769   #endif
00770
00771   bool padded{false};
00772   tran = false;
00773
00774   mtk::DenseMatrix iden(dim_null_, padded, tran);
00775
00776   #if MTK_DEBUG_LEVEL > 0
00777   std::cout << "iden =" << std::endl;
00778   std::cout << iden << std::endl;
00779   #endif
00780
00781   // Solve the system to compute the scalers.
00782   // An example of the system to solve, for k = 8, is:
00783   //
00784   // subk*scalers = iden or
00785   //
00786   // |  0.386018  -0.0339244   -0.129478 |           | 1 0 0 |
00787   // | -0.119774   0.0199423   0.0558632 |*scalers = | 0 1 0 |
00788   // | 0.0155708 -0.00349546 -0.00853182 |           | 0 0 1 |
00789   //
00790   // Notice this is a nrhs = 3 system.
00791   // Noteworthy: we do NOT ACTUALLY ALLOCATE space for the scalers... they
00792   // will be stored in the created identity matrix.
00793   // Let us first transpose subk (because of LAPACK):
00794
00795   int info{mtk::LAPACKAdapter::SolveDenseSystem(subk, iden)};
00796
00797   #if MTK_DEBUG_LEVEL > 0
00798   if (!info) {
00799     std::cout << "System successfully solved!" <<
00800       std::endl;
00801   } else {
00802     std::cerr << "Something went wrong solving system! info = " << info <<
00803       std::endl;
00804     std::cerr << "Exiting..." << std::endl;
00805     return false;
00806   }
00807   std::cout << std::endl;
00808   #endif
00809
00810   #if MTK_DEBUG_LEVEL > 0
00811   std::cout << "Computed scalers:" << std::endl;
00812   std::cout << iden << std::endl;
00813   #endif
00814
00815   // Multiply the two matrices to attain a scaled basis for null-space.
00816
00817   rat_basis_null_space_ = mtk::BLASAdapter::RealDenseMM(kk, iden);
00818
00819   #if MTK_DEBUG_LEVEL > 0
00820   std::cout << "Rational basis for the null-space:" << std::endl;
00821   std::cout << rat_basis_null_space_ << std::endl;
00822   #endif
00823
00824   // At this point, we have a rational basis for the null-space, with the
00825   // pattern we need! :)
00826
```

```
00827   delete [] gg;
00828   gg = nullptr;
00829
00830   return true;
00831 }
00832
00833 bool mtk::Grad1D::ComputePreliminaryApproximations() {
00834
00836
00837   mtk::Real *gg{}; // Generator vector for the first approximation.
00838
00839   try {
00840     gg = new mtk::Real[num_bndy_coeffs_];
00841   } catch (std::bad_alloc &memory_allocation_exception) {
00842     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00843       std::endl;
00844     std::cerr << memory_allocation_exception.what() << std::endl;
00845   }
00846   memset(gg, mtk::kZero, sizeof(gg[0])*num_bndy_coeffs_);
00847
00848   #ifdef MTK_PRECISION_DOUBLE
00849   gg[1] = 1.0/2.0;
00850   #else
00851   gg[1] = 1.0f/2.0f;
00852   #endif
00853   for (auto ii = 2; ii < num_bndy_coeffs_; ++ii) {
00854     gg[ii] = gg[ii - 1] + mtk::kOne;
00855   }
00856
00857   #if MTK_DEBUG_LEVEL > 0
00858   std::cout << "gg0 =" << std::endl;
00859   for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
00860     std::cout << std::setw(12) << gg[ii];
00861   }
00862   std::cout << std::endl << std::endl;
00863   #endif
00864
00865   // Allocate 2D array to store the collection of preliminary approximations.
00866   try {
00867     prem_apps_ = new mtk::Real[num_bndy_coeffs_*num_bndy_approxs_];
00868   } catch (std::bad_alloc &memory_allocation_exception) {
00869     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00870 std::endl;
00871     std::cerr << memory_allocation_exception.what() << std::endl;
00872   }
00873   memset(prem_apps_,
00874          mtk::kZero,
00875          sizeof(prem_apps_[0])*num_bndy_coeffs_*num_bndy_approxs_);
00876
00878
00879   for (auto ll = 0; ll < num_bndy_approxs_; ++ll) {
00880
00881     // Re-check new generator vector for every iteration except for the first.
00882     #if MTK_DEBUG_LEVEL > 0
00883     if (ll > 0) {
00884       std::cout << "gg" << ll << " =" << std::endl;
00885       for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
00886         std::cout << std::setw(12) << gg[ii];
00887       }
00888       std::cout << std::endl << std::endl;
00889     }
00890     #endif
00891
00893
00894     bool transpose{false};
00895
00896     mtk::DenseMatrix aa(gg,
00897                         num_bndy_coeffs_, order_accuracy_ + 1,
00898                         transpose);
00899
00900     #if MTK_DEBUG_LEVEL > 0
00901     std::cout << "aa_" << ll << " =" << std::endl;
00902     std::cout << aa << std::endl;
00903     #endif
00904
00906
00907     mtk::Real *ob{};
00908
00909     auto ob_ld = num_bndy_coeffs_;
00910
00911     try {
```

```
00912       ob = new mtk::Real[ob_ld];
00913     } catch (std::bad_alloc &memory_allocation_exception) {
00914       std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00915         std::endl;
00916       std::cerr << memory_allocation_exception.what() << std::endl;
00917     }
00918     memset(ob, mtk::kZero, sizeof(ob[0])*ob_ld);
00919
00920     ob[1] = mtk::kOne;
00921
00922     #if MTK_DEBUG_LEVEL > 0
00923     std::cout << "ob = " << std::endl << std::endl;
00924     for (auto ii = 0; ii < ob_ld; ++ii) {
00925       std::cout << std::setw(12) << ob[ii] << std::endl;
00926     }
00927     std::cout << std::endl;
00928     #endif
00929
00931
00932     // However, this is an under-determined system of equations. So we can not
00933     // use the same LAPACK routine (dgesv_). We will instead use dgels_, through
00934     // our LAPACKAdapter class.
00935
00936     int info_{
00937       mtk::LAPACKAdapter::SolveRectangularDenseSystem(aa, ob
    , ob_ld)};
00938
00939     #if MTK_DEBUG_LEVEL > 0
00940     if (!info_) {
00941       std::cout << "System successfully solved!" << std::endl << std::endl;
00942     } else {
00943       std::cerr << "Error solving system! info = " << info_ << std::endl;
00944     }
00945     #endif
00946
00947     #if MTK_DEBUG_LEVEL > 0
00948     std::cout << "ob =" << std::endl;
00949     for (auto ii = 0; ii < ob_ld; ++ii) {
00950       std::cout << std::setw(12) << ob[ii] << std::endl;
00951     }
00952     std::cout << std::endl;
00953     #endif
00954
00956
00957     // This implies a DAXPY operation. However, we must construct the arguments
00958     // for this operation.
00959
00961     // Save them into the ob_bottom array:
00962
00963     Real *ob_bottom{}; // Bottom part of the attained kernel used to scale it.
00964
00965     try {
00966       ob_bottom = new mtk::Real[dim_null_];
00967     } catch (std::bad_alloc &memory_allocation_exception) {
00968       std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00969         std::endl;
00970       std::cerr << memory_allocation_exception.what() << std::endl;
00971     }
00972     memset(ob_bottom, mtk::kZero, sizeof(ob_bottom[0])*dim_null_);
00973
00974     for (auto ii = 0; ii < dim_null_; ++ii) {
00975       ob_bottom[(dim_null_ - 1) - ii] = ob[num_bndy_coeffs_ - ii - 1];
00976     }
00977
00978     #if MTK_DEBUG_LEVEL > 0
00979     std::cout << "ob_bottom =" << std::endl;
00980     for (auto ii = 0; ii < dim_null_; ++ii) {
00981       std::cout << std::setw(12) << ob_bottom[ii] << std::endl;
00982     }
00983     std::cout << std::endl;
00984     #endif
00985
00987
00988     // We must computed an scaled ob, sob, using the scaled null-space in
00989     // rat_basis_null_space_.
00990     // Such operation is: sob = ob - rat_basis_null_space_*ob_bottom
00991     // or:                ob = -1.0*rat_basis_null_space_*ob_bottom + 1.0*ob
00992     // thus:               Y =    a*A    *x         +   b*Y (DAXPY).
00993
00994     #if MTK_DEBUG_LEVEL > 0
00995     std::cout << "Rational basis for the null-space:" << std::endl;
```

```
00996     std::cout << rat_basis_null_space_ << std::endl;
00997     #endif
00998
00999     mtk::Real alpha{-mtk::kOne};
01000     mtk::Real beta{mtk::kOne};
01001
01002     mtk::BLASAdapter::RealDenseMV(alpha, rat_basis_null_space_,
01003                                   ob_bottom, beta, ob);
01004
01005     #if MTK_DEBUG_LEVEL > 0
01006     std::cout << "scaled ob:" << std::endl;
01007     for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
01008       std::cout << std::setw(12) << ob[ii] << std::endl;
01009     }
01010     std::cout << std::endl;
01011     #endif
01012
01013     // We save the recently scaled solution, into an array containing these.
01014     // We can NOT start building the pi matrix, simply because I want that part
01015     // to be separated since its construction depends on the algorithm we want
01016     // to implement.
01017
01018     for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
01019       prem_apps_[ii*num_bndy_approxs_ + ll] = ob[ii];
01020     }
01021
01022     // After the first iteration, simply shift the entries of the last
01023     // generator vector used:
01024     for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
01025       gg[ii]--;
01026     }
01027
01028     // Garbage collection for this loop:
01029     delete[] ob;
01030     ob = nullptr;
01031
01032     delete[] ob_bottom;
01033     ob_bottom = nullptr;
01034   } // End of: for (ll = 0; ll < dim_null; ll++);
01035
01036   #if MTK_DEBUG_LEVEL > 0
01037   std::cout << "Matrix post-scaled preliminary apps: " << std::endl;
01038   for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
01039     for (auto jj = 0; jj < num_bndy_approxs_; ++jj) {
01040       std::cout << std::setw(12) << prem_apps_[ii*num_bndy_approxs_ + jj];
01041     }
01042     std::cout << std::endl;
01043   }
01044   std::cout << std::endl;
01045   #endif
01046
01047   delete[] gg;
01048   gg = nullptr;
01049
01050   return true;
01051 }
01052
01053 bool mtk::Grad1D::ComputeWeights() {
01054
01055   // Matrix to copmpute the weights as in the CRSA.
01056   mtk::DenseMatrix pi(num_bndy_coeffs_, num_bndy_coeffs_ - 1);
01057
01059
01060   // Assemble the pi matrix using:
01061   // 1. The collection of scaled preliminary approximations.
01062   // 2. The collection of coefficients approximating at the interior.
01063   // 3. The scaled basis for the null-space.
01064
01065   // 1.1. Process array of scaled preliminary approximations.
01066
01067   // These are queued in scaled_solutions. Each one of these, will be a column
01068   // of the pi matrix:
01069   for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
01070     for (auto jj = 0; jj < num_bndy_approxs_; ++jj) {
01071       pi.data()[ii*(2*(num_bndy_approxs_ - 1) + (order_accuracy_/2 + 1)) + jj] =
01072         prem_apps_[ii*num_bndy_approxs_ + jj];
01073     }
01074   }
01075
01076   // 1.2. Add columns from known stencil approximating at the interior.
01077
```

```
01078   // However, these must be padded by zeros, according to their position in the
01079   // final pi matrix:
01080   auto mm = 1;
01081   for (auto jj = num_bndy_approxs_; jj < order_accuracy_; ++jj) {
01082     for (auto ii = 0; ii < order_accuracy_; ++ii) {
01083       auto de = (ii + mm)*(2*(num_bndy_approxs_ - 1) +
01084         (order_accuracy_/2 + 1)) + jj;
01085       pi.data()[de] = coeffs_interior_[ii];
01086     }
01087     ++mm;
01088   }
01089
01090   rat_basis_null_space_.OrderColMajor();
01091
01092   #if MTK_DEBUG_LEVEL > 0
01093   std::cout << "Rational basis for the null-space (col. major):" << std::endl;
01094   std::cout << rat_basis_null_space_ << std::endl;
01095   #endif
01096
01097   // 1.3. Add final set of columns: rational basis for null-space.
01098
01099   for (auto jj = dim_null_ + (order_accuracy_/2 + 1);
01100        jj < num_bndy_coeffs_ - 1; ++jj) {
01101     for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
01102       auto og =
01103         (jj - (dim_null_ + (order_accuracy_/2 + 1)))*num_bndy_coeffs_ + ii;
01104       auto de = ii*(2*dim_null_ + (order_accuracy_/2 + 1)) + jj;
01105       pi.data()[de] = rat_basis_null_space_.data()[og];
01106     }
01107   }
01108
01109   #if MTK_DEBUG_LEVEL >0
01110   std::cout << "coeffs_interior_ =" << std::endl;
01111   for (auto ii = 0; ii < order_accuracy_; ++ii) {
01112     std::cout << std::setw(12) << coeffs_interior_[ii];
01113   }
01114   std::cout << std::endl << std::endl;
01115   #endif
01116
01117   #if MTK_DEBUG_LEVEL >0
01118   std::cout << "Constructed pi matrix for CRS Algorithm: " << std::endl;
01119   std::cout << pi << std::endl;
01120   #endif
01121
01123
01124   // This imposes the mimetic condition.
01125
01126   mtk::Real *hh{};  // Right-hand side to compute weights in the C{R,B}SA.
01127
01128   try {
01129     hh = new mtk::Real[num_bndy_coeffs_];
01130   } catch (std::bad_alloc &memory_allocation_exception) {
01131     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
01132       std::endl;
01133     std::cerr << memory_allocation_exception.what() << std::endl;
01134   }
01135   memset(hh, mtk::kZero, sizeof(hh[0])*num_bndy_coeffs_);
01136
01137   hh[0] = -mtk::kOne;
01138   for (auto ii = (order_accuracy_/2 + 2 - 1); ii < num_bndy_coeffs_; ++ii) {
01139     auto aux_xx = mtk::kZero;
01140     for (auto jj = 0; jj < ((ii - (order_accuracy_/2 - 1)) - 1); ++jj) {
01141       aux_xx += coeffs_interior_[jj];
01142     }
01143     hh[ii] = -mtk::kOne*aux_xx;
01144   }
01145
01147
01148   // That is, we construct a system, to solve for the weights.
01149
01150   // Once again we face the challenge of solving with LAPACK. However, for the
01151   // CRSA, this matrix PI is over-determined, since it has more rows than
01152   // unknowns. However, according to the theory, the solution to this system is
01153   // unique. We will use dgels_.
01154
01155   try {
01156     weights_cbs_ = new mtk::Real[num_bndy_coeffs_];
01157   } catch (std::bad_alloc &memory_allocation_exception) {
01158     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
01159       std::endl;
01160     std::cerr << memory_allocation_exception.what() << std::endl;
```

```
01161   }
01162   memset(weights_cbs_, mtk::kZero, sizeof(weights_cbs_[0])*num_bndy_coeffs_);
01163
01164   int weights_ld{pi.num_cols() + 1};
01165
01166   // Preserve hh.
01167   std::copy(hh, hh + weights_ld, weights_cbs_);
01168
01169   pi.Transpose();
01170
01171   int info{
01172     mtk::LAPACKAdapter::SolveRectangularDenseSystem(pi,
01173                                                     weights_cbs_, weights_ld)
01174   };
01175
01176   #if MTK_DEBUG_LEVEL > 0
01177   if (!info) {
01178     std::cout << "System successfully solved!" << std::endl << std::endl;
01179   } else {
01180     std::cerr << "Error solving system! info = " << info << std::endl;
01181   }
01182   #endif
01183
01184   #if MTK_DEBUG_LEVEL > 0
01185   std::cout << "hh =" << std::endl;
01186   for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
01187     std::cout << std::setw(11) << hh[ii] << std::endl;
01188   }
01189   std::cout << std::endl;
01190   #endif
01191
01192   // Preserve the original weights for research.
01193
01194   try {
01195     weights_crs_ = new mtk::Real[num_bndy_coeffs_];
01196   } catch (std::bad_alloc &memory_allocation_exception) {
01197     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
01198       std::endl;
01199     std::cerr << memory_allocation_exception.what() << std::endl;
01200   }
01201   memset(weights_crs_, mtk::kZero, sizeof(weights_crs_[0])*num_bndy_coeffs_);
01202
01203   std::copy(weights_cbs_, weights_cbs_ + (weights_ld - 1), weights_crs_);
01204
01205   #if MTK_DEBUG_LEVEL > 0
01206   std::cout << "weights_CRSA + lambda =" << std::endl;
01207   for (auto ii = 0; ii < weights_ld - 1; ++ii) {
01208     std::cout << std::setw(12) << weights_crs_[ii] << std::endl;
01209   }
01210   std::cout << std::endl;
01211   #endif
01212
01214
01215   if (order_accuracy_ >= mtk::kCriticalOrderAccuracyGrad) {
01216
01217     int minrow_{std::numeric_limits<int>::infinity()};
01218
01219     mtk::Real norm{mtk::BLASAdapter::RealNRM2(weights_cbs_,
    order_accuracy_)};
01220     mtk::Real minnorm{std::numeric_limits<mtk::Real>::infinity()};
01221
01223
01224     mtk::DenseMatrix phi(order_accuracy_ + 1, order_accuracy_);
01225
01226     // 6.1. Insert preliminary approximations to first set of columns.
01227
01228     for (auto ii = 0; ii < order_accuracy_ + 1; ++ii) {
01229       for (auto jj = 0; jj < num_bndy_approxs_; ++jj) {
01230         phi.data()[ii*(order_accuracy_) + jj] =
01231           prem_apps_[ii*num_bndy_approxs_ + jj];
01232       }
01233     }
01234
01235     // 6.2. Skip a column and negate preliminary approximations.
01236
01237     for (auto jj = 0; jj < order_accuracy_ + 1; jj++) {
01238       for (auto ii = 1; ii < num_bndy_approxs_; ii++) {
01239         auto de = (ii+ order_accuracy_ - num_bndy_approxs_+ jj*order_accuracy_);
01240         auto og = (num_bndy_approxs_ - ii + (jj)*num_bndy_approxs_);
01241         phi.data()[de] = -prem_apps_[og];
01242       }
```

```
01243     }
01244
01245     // 6.3. Flip negative columns up-down.
01246
01247     for (auto ii = 0; ii < order_accuracy_/2; ii++) {
01248       for (auto jj = num_bndy_approxs_ + 1; jj < order_accuracy_; jj++) {
01249         auto aux = phi.data()[ii*order_accuracy_ + jj];
01250         phi.data()[ii*order_accuracy_ + jj] =
01251           phi.data()[(order_accuracy_ - ii)*order_accuracy_ + jj];
01252         phi.data()[(order_accuracy_ - ii)*order_accuracy_ + jj] = aux;
01253       }
01254     }
01255
01256     // 6.4. Insert stencil.
01257
01258     auto mm = 0;
01259     for (auto jj = num_bndy_approxs_; jj < num_bndy_approxs_ +  1; jj++) {
01260       for (auto ii = 0; ii < order_accuracy_ + 1; ii++) {
01261         if (ii == 0) {
01262           phi.data()[jj] = 0.0;
01263         } else {
01264           phi.data()[(ii + mm)*order_accuracy_ + jj] = coeffs_interior_[ii - 1];
01265         }
01266       }
01267       mm++;
01268     }
01269
01270     #if MTK_DEBUG_LEVEL > 0
01271     std::cout << "phi =" << std::endl;
01272     std::cout << phi << std::endl;
01273     #endif
01274
01276
01277     mtk::Real *lamed{};  // Used to build big lambda.
01278
01279     try {
01280       lamed = new mtk::Real[num_bndy_approxs_ - 1];
01281     } catch (std::bad_alloc &memory_allocation_exception) {
01282       std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
01283         std::endl;
01284       std::cerr << memory_allocation_exception.what() << std::endl;
01285     }
01286     memset(lamed, mtk::kZero, sizeof(lamed[0])*(num_bndy_approxs_ - 1));
01287
01288     for (auto ii = 0; ii < num_bndy_approxs_ - 1; ++ii) {
01289       lamed[ii] = hh[ii + order_accuracy_ + 1] ;
01290     }
01291
01292     #if MTK_DEBUG_LEVEL > 0
01293     std::cout << "lamed =" << std::endl;
01294     for (auto ii = 0; ii < num_bndy_approxs_ - 1; ++ii) {
01295       std::cout << std::setw(12) << lamed[ii] << std::endl;
01296     }
01297     std::cout << std::endl;
01298     #endif
01299
01300     for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
01301       mtk::Real temp = mtk::kZero;
01302       for(auto jj = 0; jj < num_bndy_approxs_ - 1; ++jj) {
01303         temp = temp +
01304           lamed[jj]*rat_basis_null_space_.data()[jj*num_bndy_coeffs_ + ii];
01305       }
01306       hh[ii] = hh[ii] - temp;
01307     }
01308
01309     #if MTK_DEBUG_LEVEL > 0
01310     std::cout << "big_lambda =" << std::endl;
01311     for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
01312       std::cout << std::setw(12) << hh[ii] << std::endl;
01313     }
01314     std::cout << std::endl;
01315     #endif
01316
01318
01319     int copy_result{};  // Should I replace the solution... not for now.
01320
01321     mtk::Real normerr_; // Norm of the error for the solution on each row.
01322
01323     for(auto row_= 0; row_ < order_accuracy_ + 1; ++row_) {
01324       normerr_ = mtk::GLPKAdapter::SolveSimplexAndCompare(phi.
      data(),
```

```
01325                                                       order_accuracy_ + 1,
01326                                                       order_accuracy_,
01327                                                       order_accuracy_,
01328                                                       hh,
01329                                                       weights_cbs_,
01330                                                       row_,
01331                                                       mimetic_threshold_,
01332                                                       copy_result);
01333       mtk::Real aux{normerr_/norm};
01334
01335       #if MTK_DEBUG_LEVEL>0
01336       std::cout << "Relative norm: " << aux << " " << std::endl;
01337       std::cout << std::endl;
01338       #endif
01339
01340       if (aux < minnorm) {
01341         minnorm = aux;
01342         minrow_= row_;
01343       }
01344     }
01345
01346     #if MTK_DEBUG_LEVEL > 0
01347     std::cout << "weights_CBSA + lambda (after brute force search):" <<
01348       std::endl;
01349     for (auto ii = 0; ii < num_bndy_coeffs_ - 1; ++ii) {
01350       std::cout << std::setw(12) << weights_cbs_[ii] << std::endl;
01351     }
01352     std::cout << std::endl;
01353     #endif
01354
01356
01357     // After we know which row yields the smallest relative norm that row is
01358     // chosen to be the objective function and the result of the optimizer is
01359     // chosen to be the new weights_.
01360
01361     #if MTK_DEBUG_LEVEL > 0
01362     std::cout << "Minimum Relative Norm " << minnorm << " found at row " <<
01363       minrow_ + 1 << std::endl;
01364     std::cout << std::endl;
01365     #endif
01366
01367     copy_result = 1;
01368     normerr_ = mtk::GLPKAdapter::SolveSimplexAndCompare(phi.
      data(),
01369                                                         order_accuracy_ + 1,
01370                                                         order_accuracy_,
01371                                                         order_accuracy_,
01372                                                         hh,
01373                                                         weights_cbs_,
01374                                                         minrow_,
01375                                                         mimetic_threshold_,
01376                                                         copy_result);
01377     mtk::Real aux_{normerr_/norm};
01378     #if MTK_DEBUG_LEVEL > 0
01379     std::cout << "Relative norm: " << aux_ << std::endl;
01380     std::cout << std::endl;
01381     #endif
01382
01383     delete [] lamed;
01384     lamed = nullptr;
01385   }
01386
01387   delete [] hh;
01388   hh = nullptr;
01389
01390   return true;
01391 }
01392
01393 bool mtk::Grad1D::ComputeStencilBoundaryGrid(void) {
01394
01395   #if MTK_DEBUG_LEVEL > 0
01396   std::cout << "weights_* + lambda =" << std::endl;
01397   for (auto ii = 0; ii < num_bndy_coeffs_ - 1; ++ii) {
01398     std::cout << std::setw(12) << weights_cbs_[ii] << std::endl;
01399   }
01400   std::cout << std::endl;
01401   #endif
01402
01404
01405   mtk::Real *lambda{}; // Collection of bottom values from weights_.
01406
```

```
01407   try {
01408     lambda = new mtk::Real[dim_null_];
01409   } catch (std::bad_alloc &memory_allocation_exception) {
01410     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
01411       std::endl;
01412     std::cerr << memory_allocation_exception.what() << std::endl;
01413   }
01414   memset(lambda, mtk::kZero, sizeof(lambda[0])*dim_null_);
01415
01416   for (auto ii = 0; ii < dim_null_; ++ii) {
01417     lambda[ii] = weights_cbs_[order_accuracy_ + ii];
01418   }
01419
01420   #if MTK_DEBUG_LEVEL > 0
01421   std::cout << "lambda =" << std::endl;
01422   for (auto ii = 0; ii < dim_null_; ++ii) {
01423     std::cout << std::setw(12) << lambda[ii] << std::endl;
01424   }
01425   std::cout << std::endl;
01426   #endif
01427
01429
01430   mtk::Real *alpha{}; // Collection of alpha values.
01431
01432   try {
01433     alpha = new mtk::Real[dim_null_];
01434   } catch (std::bad_alloc &memory_allocation_exception) {
01435     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
01436       std::endl;
01437     std::cerr << memory_allocation_exception.what() << std::endl;
01438   }
01439   memset(alpha, mtk::kZero, sizeof(alpha[0])*dim_null_);
01440
01441   for (auto ii = 0; ii < dim_null_; ++ii) {
01442     alpha[ii] = lambda[ii]/weights_cbs_[ii] ;
01443   }
01444
01445   #if MTK_DEBUG_LEVEL > 0
01446   std::cout << "alpha =" << std::endl;
01447   for (auto ii = 0; ii < dim_null_; ++ii) {
01448     std::cout << std::setw(12) << alpha[ii] << std::endl;
01449   }
01450   std::cout << std::endl;
01451   #endif
01452
01454
01455   try {
01456     mim_bndy_ = new mtk::Real[num_bndy_coeffs_*num_bndy_approxs_];
01457   } catch (std::bad_alloc &memory_allocation_exception) {
01458     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
01459       std::endl;
01460     std::cerr << memory_allocation_exception.what() << std::endl;
01461   }
01462   memset(mim_bndy_,
01463          mtk::kZero,
01464          sizeof(mim_bndy_[0])*num_bndy_coeffs_*num_bndy_approxs_);
01465
01466   for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
01467     for (auto jj = 0; jj < (num_bndy_approxs_ - 1); ++jj) {
01468       mim_bndy_[ii*num_bndy_approxs_ + jj] =
01469         prem_apps_[ii*num_bndy_approxs_ + jj] +
01470         alpha[jj]*rat_basis_null_space_.data()[jj*num_bndy_coeffs_ + ii];
01471     }
01472   }
01473
01474   for(auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
01475     mim_bndy_[ii*num_bndy_approxs_ + (num_bndy_approxs_ - 1)] =
01476       prem_apps_[ii*num_bndy_approxs_ + (num_bndy_approxs_ - 1)];
01477   }
01478
01479   #if MTK_DEBUG_LEVEL > 0
01480   std::cout << "Collection of mimetic approximations:" << std::endl;
01481   for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
01482     for (auto jj = 0; jj < num_bndy_approxs_; ++jj) {
01483       std::cout << std::setw(13) << mim_bndy_[ii*num_bndy_approxs_ + jj];
01484     }
01485     std::cout << std::endl;
01486   }
01487   std::cout << std::endl;
01488   #endif
01489
```

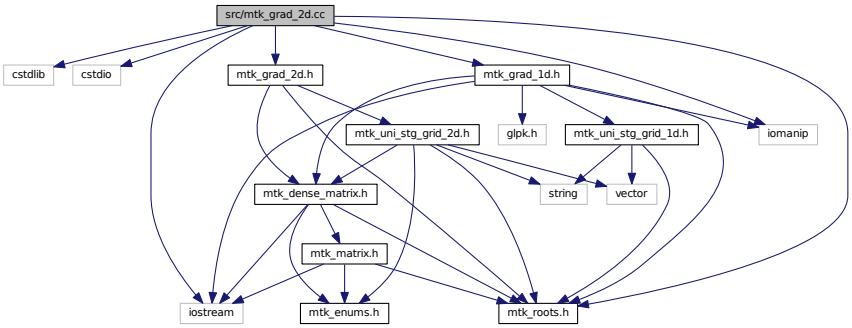
```
01490   delete[] lambda;
01491   lambda = nullptr;
01492
01493   delete[] alpha;
01494   alpha = nullptr;
01495
01496   return true;
01497 }
01498
01499 bool mtk::Grad1D::AssembleOperator(void) {
01500
01501   // The output array will have this form:
01502   // 1. The first entry of the array will contain the used order kk.
01503   // 2. The second entry of the array will contain the collection of
01504   // approximating coefficients for the interior of the grid.
01505   // 3. The third entry will contain a collection of weights.
01506   // 4. The next dim_null - 1 entries will contain the collections of
01507   // approximating coefficients for the west boundary of the grid.
01508
01509   gradient_length_ = 1 + order_accuracy_ + order_accuracy_ +
01510     num_bndy_approxs_*num_bndy_coeffs_;
01511
01512   #if MTK_DEBUG_LEVEL > 0
01513   std::cout << "gradient_length_ = " << gradient_length_ << std::endl;
01514   #endif
01515
01516   try {
01517     gradient_ = new mtk::Real[gradient_length_];
01518   } catch (std::bad_alloc &memory_allocation_exception) {
01519     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
01520       std::endl;
01521     std::cerr << memory_allocation_exception.what() << std::endl;
01522   }
01523   memset(gradient_, mtk::kZero, sizeof(gradient_[0])*gradient_length_);
01524
01526
01527   gradient_[0] = order_accuracy_;
01528
01531
01532   for (auto ii = 0; ii < order_accuracy_; ++ii) {
01533     gradient_[ii + 1] = coeffs_interior_[ii];
01534   }
01535
01537
01538   for (auto ii = 0; ii < order_accuracy_; ++ii) {
01539     gradient_[(order_accuracy_ + 1) + ii] = weights_cbs_[ii];
01540   }
01541
01544
01545   int offset{2*order_accuracy_ + 1};
01546
01547   int aux {}; // Auxiliary variable.
01548
01549   if (order_accuracy_ > mtk::kDefaultOrderAccuracy) {
01550     for (auto ii = 0; ii < num_bndy_approxs_ ; ii++) {
01551       for (auto jj = 0; jj < num_bndy_coeffs_; jj++) {
01552         gradient_[offset + aux] = mim_bndy_[jj*num_bndy_approxs_ + ii];
01553         aux++;
01554       }
01555     }
01556   } else {
01557     gradient_[offset + 0] = prem_apps_[0];
01558     gradient_[offset + 1] = prem_apps_[1];
01559     gradient_[offset + 2] = prem_apps_[2];
01560   }
01561
01562   #if MTK_DEBUG_LEVEL > 0
01563   std::cout << "1D " << order_accuracy_ << "-order grad built!" << std::endl;
01564   std::cout << std::endl;
01565   #endif
01566
01567   return true;
01568 }
```

17.69 src/mtk_grad_2d.cc File Reference

Implements the class Grad2D.

```
#include <cstdlib>
#include <cstdio>
#include <iostream>
#include <iomanip>
#include "mtk_roots.h"
#include "mtk_grad_1d.h"
#include "mtk_grad_2d.h"
```

Include dependency graph for mtk_grad_2d.cc:



17.69.1 Detailed Description

This class implements a 2D gradient operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm (C↩ BSA).

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_grad_2d.cc.

17.70 mtk_grad_2d.cc

```
00001
00011 /*
00012 Copyright (C) 2015, Computational Science Research Center, San Diego State
00013 University. All rights reserved.
00014
00015 Redistribution and use in source and binary forms, with or without modification,
00016 are permitted provided that the following conditions are met:
00017
00018 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00019 and a copy of the modified files should be reported once modifications are
00020 completed, unless these modifications are made through the project's GitHub
00021 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00022 should be developed and included in any deliverable.
00023
00024 2. Redistributions of source code must be done through direct
00025 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00026
```

```
00027 3. Redistributions in binary form must reproduce the above copyright notice,
00028 this list of conditions and the following disclaimer in the documentation and/or
00029 other materials provided with the distribution.
00030
00031 4. Usage of the binary form on proprietary applications shall require explicit
00032 prior written permission from the the copyright holders, and due credit should
00033 be given to the copyright holders.
00034
00035 5. Neither the name of the copyright holder nor the names of its contributors
00036 may be used to endorse or promote products derived from this software without
00037 specific prior written permission.
00038
00039 The copyright holders provide no reassurances that the source code provided does
00040 not infringe any patent, copyright, or any other intellectual property rights of
00041 third parties. The copyright holders disclaim any liability to any recipient for
00042 claims brought against recipient by any third party for infringement of that
00043 parties intellectual property rights.
00044
00045 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00046 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00047 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00048 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00049 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00050 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00051 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00052 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00053 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00054 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00055 */
00056
00057 #include <cstdlib>
00058 #include <cstdio>
00059
00060 #include <iostream>
00061 #include <iomanip>
00062
00063 #include "mtk_roots.h"
00064 #include "mtk_grad_1d.h"
00065 #include "mtk_grad_2d.h"
00066
00067 mtk::Grad2D::Grad2D():
00068   order_accuracy_(),
00069   mimetic_threshold_() {}
00070
00071 mtk::Grad2D::Grad2D(const Grad2D &grad):
00072   order_accuracy_(grad.order_accuracy_),
00073   mimetic_threshold_(grad.mimetic_threshold_) {}
00074
00075 mtk::Grad2D::~Grad2D() {}
00076
00077 bool mtk::Grad2D::ConstructGrad2D(const
      mtk::UniStgGrid2D &grid,
00078                                   int order_accuracy,
00079                                   mtk::Real mimetic_threshold) {
00080
00081   int num_cells_x = grid.num_cells_x();
00082   int num_cells_y = grid.num_cells_y();
00083
00084   int mx = num_cells_x + 1;  // Gx vertical dimension
00085   int nx = num_cells_x + 2;  // Gx horizontal dimension
00086   int my = num_cells_y + 1;  // Gy vertical dimension
00087   int ny = num_cells_y + 2;  // Gy horizontal dimension
00088
00089   mtk::Grad1D grad;
00090
00091   bool info = grad.ConstructGrad1D(order_accuracy, mimetic_threshold);
00092
00093   if (!info) {
00094     std::cerr << "Mimetic grad could not be built." << std::endl;
00095     return info;
00096   }
00097
00098   auto west = grid.west_bndy();
00099   auto east = grid.east_bndy();
00100   auto south = grid.south_bndy();
00101   auto north = grid.east_bndy();
00102
00103   mtk::UniStgGrid1D grid_x(west, east, num_cells_x);
00104   mtk::UniStgGrid1D grid_y(south, north, num_cells_y);
00105
00106   mtk::DenseMatrix Gx(grad.ReturnAsDenseMatrix(grid_x));
```

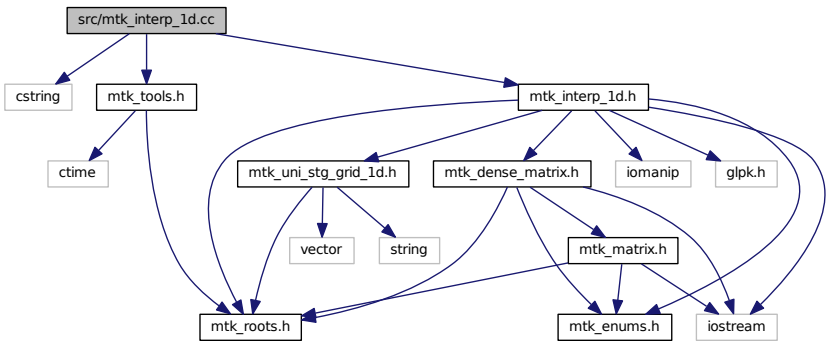
```
00107   mtk::DenseMatrix Gy(grad.ReturnAsDenseMatrix(grid_y));
00108
00109   bool padded{true};
00110   bool transpose{true};
00111
00112   mtk::DenseMatrix tix(num_cells_x, padded, transpose);
00113   mtk::DenseMatrix tiy(num_cells_y, padded, transpose);
00114
00115   mtk::DenseMatrix gxy(mtk::DenseMatrix::Kron(tiy, Gx));
00116   mtk::DenseMatrix gyx(mtk::DenseMatrix::Kron(Gy, tix));
00117
00118   #if MTK_DEBUG_LEVEL > 0
00119   std::cout << "Gx: " << mx << " by " << nx << std::endl;
00120   std::cout << "Transpose Iy: " << num_cells_y << " by " << ny  << std::endl;
00121   std::cout << "Gy: " << my << " by " << ny << std::endl;
00122   std::cout << "Transpose Ix: " << num_cells_x << " by " << nx  << std::endl;
00123   std::cout << "Grad 2D: " << mx*num_cells_y + my*num_cells_x << " by " <<
00124     nx*ny <<std::endl;
00125   #endif
00126
00127   mtk::DenseMatrix g2d(mx*num_cells_y + my*num_cells_x, nx*ny);
00128
00129   for(auto ii = 0; ii < nx*ny; ii++) {
00130     for(auto jj = 0; jj < mx*num_cells_y; jj++) {
00131       g2d.SetValue(jj,ii, gxy.GetValue(jj,ii));
00132     }
00133     for(auto kk = 0; kk < my*num_cells_x; kk++) {
00134       g2d.SetValue(kk + mx*num_cells_y, ii, gyx.GetValue(kk,ii));
00135     }
00136   }
00137
00138   gradient_ = g2d;
00139
00140   return info;
00141 }
00142
00143 mtk::DenseMatrix mtk::Grad2D::ReturnAsDenseMatrix() const {
00144
00145   return gradient_;
00146 }
```

17.71 src/mtk_interp_1d.cc File Reference

Includes the implementation of the class Interp1D.

```
#include <cstring>
#include "mtk_tools.h"
#include "mtk_interp_1d.h"
```

Include dependency graph for mtk_interp_1d.cc:



Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

Functions

- std::ostream & mtk::operator<< (std::ostream &stream, mtk::Interp1D &in)

17.71.1 Detailed Description

This class implements a 1D interpolation operator.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu
: Johnny Corbino - jcorbino at mail dot sdsu dot edu

Definition in file mtk_interp_1d.cc.

17.72 mtk_interp_1d.cc

```
00001
00012 /*
00013 Copyright (C) 2015, Computational Science Research Center, San Diego State
00014 University. All rights reserved.
00015
00016 Redistribution and use in source and binary forms, with or without modification,
00017 are permitted provided that the following conditions are met:
00018
00019 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00020 and a copy of the modified files should be reported once modifications are
00021 completed, unless these modifications are made through the project's GitHub
00022 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00023 should be developed and included in any deliverable.
00024
00025 2. Redistributions of source code must be done through direct
00026 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00027
00028 3. Redistributions in binary form must reproduce the above copyright notice,
00029 this list of conditions and the following disclaimer in the documentation and/or
00030 other materials provided with the distribution.
00031
00032 4. Usage of the binary form on proprietary applications shall require explicit
00033 prior written permission from the the copyright holders, and due credit should
00034 be given to the copyright holders.
00035
00036 5. Neither the name of the copyright holder nor the names of its contributors
00037 may be used to endorse or promote products derived from this software without
00038 specific prior written permission.
00039
00040 The copyright holders provide no reassurances that the source code provided does
00041 not infringe any patent, copyright, or any other intellectual property rights of
00042 third parties. The copyright holders disclaim any liability to any recipient for
00043 claims brought against recipient by any third party for infringement of that
00044 parties intellectual property rights.
00045
00046 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00047 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00048 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00049 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00050 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00051 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00052 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00053 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
```

```
00054 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00055 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00056 */
00057
00058 #include <cstring>
00059
00060 #include "mtk_tools.h"
00061
00062 #include "mtk_interp_1d.h"
00063
00064 namespace mtk {
00065
00066 std::ostream& operator <<(std::ostream &stream, mtk::Interp1D &in) {
00067
00069
00070   stream << "coeffs_interior_[1:" << in.order_accuracy_ << "] = ";
00071   for (auto ii = 0; ii < in.order_accuracy_; ++ii) {
00072     stream << std::setw(9) << in.coeffs_interior_[ii] << " ";
00073   }
00074   stream << std::endl;
00075
00076   return stream;
00077 }
00078 }
00079
00080 mtk::Interp1D::Interp1D():
00081   dir_interp_(mtk::SCALAR_TO_VECTOR),
00082   order_accuracy_(mtk::kDefaultOrderAccuracy),
00083   coeffs_interior_(nullptr) {}
00084
00085 mtk::Interp1D::Interp1D(const Interp1D &interp):
00086   dir_interp_(interp.dir_interp_),
00087   order_accuracy_(interp.order_accuracy_),
00088   coeffs_interior_(interp.coeffs_interior_) {}
00089
00090 mtk::Interp1D::~Interp1D() {
00091
00092   delete[] coeffs_interior_;
00093   coeffs_interior_ = nullptr;
00094 }
00095
00096 bool mtk::Interp1D::ConstructInterp1D(int order_accuracy,
      mtk::DirInterp dir) {
00097
00098   #if MTK_DEBUG_LEVEL > 0
00099   mtk::Tools::Prevent(order_accuracy < 2, __FILE__, __LINE__, __func__);
00100   mtk::Tools::Prevent((order_accuracy%2) != 0, __FILE__, __LINE__, __func__);
00101   mtk::Tools::Prevent(dir < mtk::SCALAR_TO_VECTOR &&
00102                       dir > mtk::VECTOR_TO_SCALAR,
00103                       __FILE__, __LINE__, __func__);
00104
00105   std::cout << "order_accuracy_ = " << order_accuracy << std::endl;
00106   #endif
00107
00108   order_accuracy_ = order_accuracy;
00109
00111
00112   try {
00113     coeffs_interior_ = new mtk::Real[order_accuracy_];
00114   } catch (std::bad_alloc &memory_allocation_exception) {
00115     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00116       std::endl;
00117     std::cerr << memory_allocation_exception.what() << std::endl;
00118   }
00119   memset(coeffs_interior_,
00120          mtk::kZero,
00121          sizeof(coeffs_interior_[0])*order_accuracy_);
00122
00123   for (int ii = 0; ii < order_accuracy_; ++ii) {
00124     coeffs_interior_[ii] = mtk::kOne;
00125   }
00126
00127   return true;
00128 }
00129
00130 mtk::Real *mtk::Interp1D::coeffs_interior() const {
00131
00132   return coeffs_interior_;
00133 }
00134
00135 mtk::DenseMatrix mtk::Interp1D::ReturnAsDenseMatrix(
```

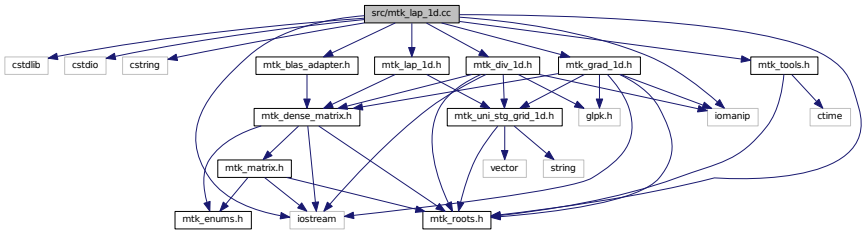
```
00136   const UniStgGrid1D &grid) const {
00137
00138   int nn{grid.num_cells_x()}; // Number of cells on the grid.
00139
00140   #if MTK_DEBUG_LEVEL > 0
00141   mtk::Tools::Prevent(nn <= 0, __FILE__, __LINE__, __func__);
00142   #endif
00143
00144   int gg_num_rows{};  // Number of rows.
00145   int gg_num_cols{};  // Number of columns.
00146
00147   if (dir_interp_ == mtk::SCALAR_TO_VECTOR) {
00148     gg_num_rows = nn + 1;
00149     gg_num_cols = nn + 2;
00150   } else {
00151     gg_num_rows = nn + 2;
00152     gg_num_cols = nn + 1;
00153   }
00154
00155   // Output matrix featuring sizes for gradient operators.
00156
00157   mtk::DenseMatrix out(gg_num_rows, gg_num_cols);
00158
00160
00161   out.SetValue(0, 0, mtk::kOne);
00162
00164
00165   for (auto ii = 1; ii < gg_num_rows - 1; ++ii) {
00166     for(auto jj = ii ; jj < order_accuracy_ + ii; ++jj) {
00167       out.SetValue(ii, jj, mtk::kOne/order_accuracy_);
00168     }
00169   }
00170
00172
00173   out.SetValue(gg_num_rows - 1, gg_num_cols - 1, mtk::kOne);
00174
00175   return out;
00176 }
```

17.73 src/mtk_lap_1d.cc File Reference

Includes the implementation of the class Lap1D.

```
#include <cstdlib>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <iomanip>
#include "mtk_roots.h"
#include "mtk_tools.h"
#include "mtk_blas_adapter.h"
#include "mtk_grad_1d.h"
#include "mtk_div_1d.h"
#include "mtk_lap_1d.h"
```

Include dependency graph for mtk_lap_1d.cc:



Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

Functions

- std::ostream & mtk::operator<< (std::ostream &stream, mtk::Lap1D &in)

17.73.1 Detailed Description

This class implements a 1D Laplacian operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm (CBSA).

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_lap_1d.cc.

17.74 mtk_lap_1d.cc

```
00001
00011 /*
00012 Copyright (C) 2015, Computational Science Research Center, San Diego State
00013 University. All rights reserved.
00014
00015 Redistribution and use in source and binary forms, with or without modification,
00016 are permitted provided that the following conditions are met:
00017
00018 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00019 and a copy of the modified files should be reported once modifications are
00020 completed, unless these modifications are made through the project's GitHub
00021 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00022 should be developed and included in any deliverable.
00023
00024 2. Redistributions of source code must be done through direct
00025 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00026
00027 3. Redistributions in binary form must reproduce the above copyright notice,
00028 this list of conditions and the following disclaimer in the documentation and/or
00029 other materials provided with the distribution.
00030
00031 4. Usage of the binary form on proprietary applications shall require explicit
```

```
00032 prior written permission from the the copyright holders, and due credit should
00033 be given to the copyright holders.
00034
00035 5. Neither the name of the copyright holder nor the names of its contributors
00036 may be used to endorse or promote products derived from this software without
00037 specific prior written permission.
00038
00039 The copyright holders provide no reassurances that the source code provided does
00040 not infringe any patent, copyright, or any other intellectual property rights of
00041 third parties. The copyright holders disclaim any liability to any recipient for
00042 claims brought against recipient by any third party for infringement of that
00043 parties intellectual property rights.
00044
00045 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00046 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00047 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00048 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00049 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00050 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00051 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00052 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00053 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00054 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00055 */
00056
00057 #include <cstdlib>
00058 #include <cstdio>
00059 #include <cstring>
00060
00061 #include <iostream>
00062 #include <iomanip>
00063
00064 #include "mtk_roots.h"
00065 #include "mtk_tools.h"
00066 #include "mtk_blas_adapter.h"
00067 #include "mtk_grad_1d.h"
00068 #include "mtk_div_1d.h"
00069 #include "mtk_lap_1d.h"
00070
00071 namespace mtk {
00072
00073 std::ostream& operator <<(std::ostream &stream, mtk::Lap1D &in) {
00074
00076
00077   stream << "laplacian_[0] = " << in.laplacian_[0] << std::endl << std::endl;
00078
00080
00081   stream << "laplacian_[1:" << 2*in.order_accuracy_ - 1 << "] = " <<
00082     std::endl << std::endl;
00083   for (auto ii = 1; ii <= (2*in.order_accuracy_ - 1); ++ii) {
00084     stream << std::setw(13) << in.laplacian_[ii] << " ";
00085   }
00086   stream << std::endl << std::endl;
00087
00089
00090   auto offset = 1 + (2*in.order_accuracy_ - 1);
00091
00092   stream << "laplacian_[" << offset << ":" << offset +
00093     (in.order_accuracy_ - 1)*(2*in.order_accuracy_) - 1 << "] = " <<
00094     std::endl << std::endl;
00095
00096   for (auto ii = 0; ii < in.order_accuracy_ - 1; ++ii) {
00097     for (auto jj = 0; jj < 2*in.order_accuracy_; ++jj) {
00098       stream << std::setw(13) <<
00099         in.laplacian_[offset + ii*(2*in.order_accuracy_) + jj];
00100     }
00101     stream << std::endl;
00102   }
00103
00104   return stream;
00105 }
00106 }
00107
00108 mtk::Lap1D::Lap1D():
00109   order_accuracy_(mtk::kDefaultOrderAccuracy),
00110   laplacian_length_(),
00111   mimetic_threshold_(mtk::kDefaultMimeticThreshold) {}
00112
00113 mtk::Lap1D::~Lap1D() {
00114
00115   delete [] laplacian_;
```

```
00116   laplacian_ = nullptr;
00117 }
00118
00119 bool mtk::Lap1D::ConstructLap1D(int order_accuracy,
00120                                 mtk::Real mimetic_threshold) {
00121
00122   #if MTK_DEBUG_LEVEL > 0
00123   mtk::Tools::Prevent(order_accuracy < 2, __FILE__, __LINE__, __func__);
00124   mtk::Tools::Prevent((order_accuracy%2) != 0, __FILE__, __LINE__, __func__);
00125   mtk::Tools::Prevent(mimetic_threshold <= mtk::kZero,
00126                       __FILE__, __LINE__, __func__);
00127
00128   if (order_accuracy >= mtk::kCriticalOrderAccuracyDiv) {
00129     std::cout << "WARNING: Numerical accuracy is high." << std::endl;
00130   }
00131
00132   std::cout << "order_accuracy_ = " << order_accuracy << std::endl;
00133   std::cout << "mimetic_threshold_ = " << mimetic_threshold << std::endl;
00134   #endif
00135
00136   order_accuracy_ = order_accuracy;
00137   mimetic_threshold_ = mimetic_threshold;
00138
00140
00141   mtk::Grad1D grad; // Mimetic gradient.
00142
00143   bool info = grad.ConstructGrad1D(order_accuracy_, mimetic_threshold_);
00144
00145   if (!info) {
00146     std::cerr << "Mimetic grad could not be built." << std::endl;
00147     return false;
00148   }
00149
00151
00152   mtk::Div1D div; // Mimetic divergence.
00153
00154   info = div.ConstructDiv1D(order_accuracy_, mimetic_threshold_);
00155
00156   if (!info) {
00157     std::cerr << "Mimetic div could not be built." << std::endl;
00158     return false;
00159   }
00160
00162
00163   // Since these are mimetic operator, we must multiply the matrices arising
00164   // from both the divergence and the Laplacian, in order to get the
00165   // approximating coefficients for the Laplacian operator.
00166
00167   // However, we must choose a grid that implied a step size of 1, so to get
00168   // the approximating coefficients, without being affected from the
00169   // normalization with respect to the grid.
00170
00171   // Also, the grid must be of the minimum size to support the requested order
00172   // of accuracy. We must please the divergence.
00173
00174   mtk::UniStgGrid1D aux(mtk::kZero,
00175                         (mtk::Real) 3*order_accuracy_ - 1,
00176                         3*order_accuracy_ - 1);
00177
00178   #if MTK_DEBUG_LEVEL > 0
00179   std::cout << "aux =" << std::endl;
00180   std::cout << aux << std::endl;
00181   std::cout <<"aux.delta_x() = " << aux.delta_x() << std::endl;
00182   std::cout << std::endl;
00183   #endif
00184
00185   mtk::DenseMatrix grad_m(grad.ReturnAsDenseMatrix(aux));
00186
00187   #if MTK_DEBUG_LEVEL > 0
00188   std::cout << "grad_m =" << std::endl;
00189   std::cout << grad_m << std::endl;
00190   #endif
00191
00192   mtk::DenseMatrix div_m(div.ReturnAsDenseMatrix(aux));
00193
00194   #if MTK_DEBUG_LEVEL > 0
00195   std::cout << "div_m =" << std::endl;
00196   std::cout << div_m << std::endl;
00197   #endif
00198
00202
```

```
00203   mtk::DenseMatrix lap; // Laplacian matrix to hold to computed coefficients.
00204
00205   lap = mtk::BLASAdapter::RealDenseMM(div_m, grad_m);
00206
00207   #if MTK_DEBUG_LEVEL > 0
00208   std::cout << "lap =" << std::endl;
00209   std::cout << lap << std::endl;
00210   #endif
00211
00213
00215
00216   // The output array will have this form:
00217   // 1. The first entry of the array will contain the used order kk.
00218   // 2. The second entry of the array will contain the collection of
00219   // approximating coefficients for the interior of the grid.
00220   // 3. The next entries will contain the collections of approximating
00221   // coefficients for the west boundary of the grid.
00222
00223   laplacian_length_ = 1 + (2*order_accuracy_ - 1) +
00224     (order_accuracy_ - 1)*(2*order_accuracy_);
00225
00226   #if MTK_DEBUG_LEVEL > 0
00227   std::cout << "laplacian_length_ = " << laplacian_length_ << std::endl;
00228   std::cout << std::endl;
00229   #endif
00230
00231   try {
00232     laplacian_ = new mtk::Real[laplacian_length_];
00233   } catch (std::bad_alloc &memory_allocation_exception) {
00234     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00235       std::endl;
00236     std::cerr << memory_allocation_exception.what() << std::endl;
00237   }
00238   memset(laplacian_, mtk::kZero, sizeof(laplacian_[0])*laplacian_length_);
00239
00241
00242   laplacian_[0] = order_accuracy_;
00243
00246
00247   for (auto ii = 0; ii < 2*order_accuracy_ - 1; ++ii) {
00248     laplacian_[ii + 1] = lap.GetValue(1 + (order_accuracy_ - 1), ii + 1);
00249   }
00250
00252
00253   auto offset = 1 + (2*order_accuracy_ - 1);
00254
00255   for (auto ii = 0; ii < order_accuracy_ - 1; ++ii) {
00256     for (auto jj = 0; jj < 2*order_accuracy_; ++jj) {
00257       laplacian_[offset + ii*(2*order_accuracy_) + jj] =
00258         lap.GetValue(1 + ii,jj);
00259     }
00260   }
00261
00262   return true;
00263 }
00264
00265 mtk::DenseMatrix mtk::Lap1D::ReturnAsDenseMatrix(
00266   const UniStgGrid1D &grid) const {
00267
00268   int nn{grid.num_cells_x()};  // Number of cells on the grid.
00269
00270   #if MTK_DEBUG_LEVEL > 0
00271   mtk::Tools::Prevent(nn <= 0, __FILE__, __LINE__, __func__);
00272   mtk::Tools::Prevent(nn < 3*order_accuracy_ - 1, __FILE__, __LINE__, __func__);
00273   #endif
00274
00275   mtk::DenseMatrix lap(nn + 2, nn + 2); // Laplacian matrix to be returned.
00276
00277   mtk::Real idx{mtk::kOne/(grid.delta_x()*grid.delta_x())}; // Inverse of
      dx^2.
00278
00280
00281   auto offset = (1 + 2*order_accuracy_ - 1);
00282
00283   for (auto ii = 0; ii < order_accuracy_ - 1; ++ii) {
00284     for (auto jj = 0; jj < 2*order_accuracy_; ++jj) {
00285       lap.SetValue(1 + ii,
00286                    jj,
00287                    idx*laplacian_[offset + ii*2*order_accuracy_ + jj]);
00288     }
00289   }
```

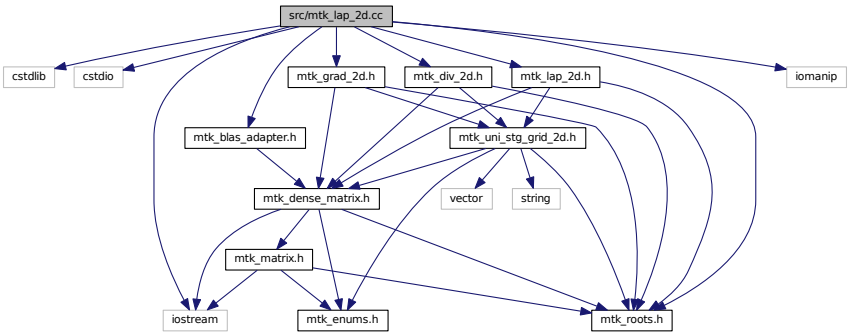
```
00290
00292
00293   offset = 1 + (order_accuracy_ - 1);
00294
00295   int kk{1};
00296   for (auto ii = order_accuracy_; ii <= nn - (order_accuracy_ - 1); ++ii) {
00297     int mm{1};
00298     for (auto jj = 0; jj < 2*order_accuracy_ - 1; ++jj) {
00299       lap.SetValue(ii, jj + kk, idx*laplacian_[mm]);
00300       mm = mm + 1;
00301     }
00302     kk = kk + 1;
00303   }
00304
00306
00307   offset = (1 + 2*order_accuracy_ - 1);
00308
00309   auto aux = order_accuracy_ + (nn - 2*(order_accuracy_ - 1));
00310
00311   auto ll = 1;
00312   auto rr = 1;
00313   for (auto ii = nn; ii > aux - 1; --ii) {
00314     auto cc = 0;
00315     for (auto jj = nn + 2 - 1; jj >= (nn + 2) - 2*order_accuracy_; --jj) {
00316       lap.SetValue(ii, jj, lap.GetValue(rr,cc));
00317       ++ll;
00318       ++cc;
00319     }
00320     rr++;
00321   }
00322
00329
00330   return lap;
00331 }
00332
00333 const mtk::Real* mtk::Lap1D::data(const UniStgGrid1D &grid) const {
00334
00335   mtk::DenseMatrix tmp;
00336
00337   tmp = ReturnAsDenseMatrix(grid);
00338
00339   return tmp.data();
00340 }
```

17.75 src/mtk_lap_2d.cc File Reference

Includes the implementation of the class Lap2D.

```
#include <cstdlib>
#include <cstdio>
#include <iostream>
#include <iomanip>
#include "mtk_roots.h"
#include "mtk_blas_adapter.h"
#include "mtk_grad_2d.h"
#include "mtk_div_2d.h"
#include "mtk_lap_2d.h"
```

Include dependency graph for mtk_lap_2d.cc:



17.75.1 Detailed Description

This class implements a 2D Laplacian operator, constructed using the Castillo-Blomgren-Sanchez (CBS) Algorithm (CBSA).

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_lap_2d.cc.

17.76 mtk_lap_2d.cc

```
00001
00011 /*
00012 Copyright (C) 2015, Computational Science Research Center, San Diego State
00013 University. All rights reserved.
00014
00015 Redistribution and use in source and binary forms, with or without modification,
00016 are permitted provided that the following conditions are met:
00017
00018 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00019 and a copy of the modified files should be reported once modifications are
00020 completed, unless these modifications are made through the project's GitHub
00021 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00022 should be developed and included in any deliverable.
00023
00024 2. Redistributions of source code must be done through direct
00025 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00026
00027 3. Redistributions in binary form must reproduce the above copyright notice,
00028 this list of conditions and the following disclaimer in the documentation and/or
00029 other materials provided with the distribution.
00030
00031 4. Usage of the binary form on proprietary applications shall require explicit
00032 prior written permission from the the copyright holders, and due credit should
00033 be given to the copyright holders.
00034
00035 5. Neither the name of the copyright holder nor the names of its contributors
00036 may be used to endorse or promote products derived from this software without
00037 specific prior written permission.
00038
00039 The copyright holders provide no reassurances that the source code provided does
00040 not infringe any patent, copyright, or any other intellectual property rights of
00041 third parties. The copyright holders disclaim any liability to any recipient for
00042 claims brought against recipient by any third party for infringement of that
```

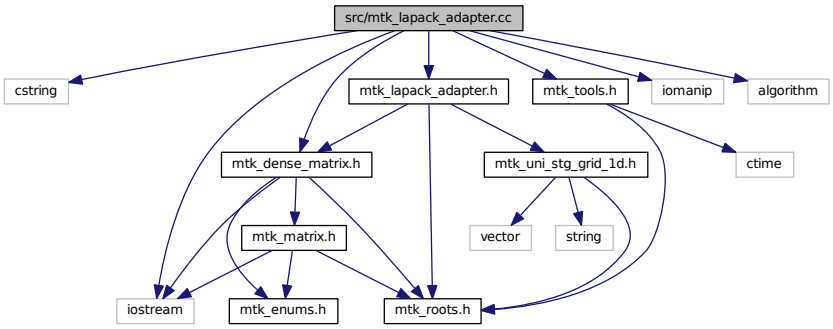
```
00043 parties intellectual property rights.
00044
00045 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00046 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00047 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00048 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00049 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00050 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00051 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00052 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00053 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00054 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00055 */
00056
00057 #include <cstdlib>
00058 #include <cstdio>
00059
00060 #include <iostream>
00061 #include <iomanip>
00062
00063 #include "mtk_roots.h"
00064 #include "mtk_blas_adapter.h"
00065 #include "mtk_grad_2d.h"
00066 #include "mtk_div_2d.h"
00067 #include "mtk_lap_2d.h"
00068
00069 mtk::Lap2D::Lap2D(): order_accuracy_(), mimetic_threshold_() {}
00070
00071 mtk::Lap2D::Lap2D(const Lap2D &lap):
00072   order_accuracy_(lap.order_accuracy_),
00073   mimetic_threshold_(lap.mimetic_threshold_) {}
00074
00075 mtk::Lap2D::~Lap2D() {}
00076
00077 bool mtk::Lap2D::ConstructLap2D(const
      mtk::UniStgGrid2D &grid,
00078                                 int order_accuracy,
00079                                 mtk::Real mimetic_threshold) {
00080
00081   mtk::Grad2D gg;
00082   mtk::Div2D dd;
00083
00084   bool info{gg.ConstructGrad2D(grid, order_accuracy, mimetic_threshold)};
00085
00086   if (!info) {
00087     std::cerr << "Mimetic lap could not be built." << std::endl;
00088     return info;
00089   }
00090
00091   info = dd.ConstructDiv2D(grid, order_accuracy, mimetic_threshold);
00092
00093   if (!info) {
00094     std::cerr << "Mimetic div could not be built." << std::endl;
00095     return info;
00096   }
00097
00098   mtk::DenseMatrix ggm(gg.ReturnAsDenseMatrix());
00099   mtk::DenseMatrix ddm(dd.ReturnAsDenseMatrix());
00100
00101   laplacian_ = mtk::BLASAdapter::RealDenseMM(ddm, ggm);
00102
00103   return info;
00104 }
00105
00106 mtk::DenseMatrix mtk::Lap2D::ReturnAsDenseMatrix() const {
00107
00108   return laplacian_;
00109 }
00110
00111 mtk::Real *mtk::Lap2D::data() const {
00112
00113   return laplacian_.data();
00114 }
```

17.77 src/mtk_lapack_adapter.cc File Reference

Adapter class for the LAPACK API.

```
#include <cstring>
#include <iostream>
#include <iomanip>
#include <algorithm>
#include "mtk_tools.h"
#include "mtk_dense_matrix.h"
#include "mtk_lapack_adapter.h"
```

Include dependency graph for mtk_lapack_adapter.cc:



Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

Functions

- void mtk::sgesv_ (int *n, int *nrhs, Real *a, int *lda, int *ipiv, Real *b, int *ldb, int *info)
- void mtk::sgels_ (char *trans, int *m, int *n, int *nrhs, Real *a, int *lda, Real *b, int *ldb, Real *work, int *lwork, int *info)

  Single-precision GEneral matrix Least Squares solver.
- void mtk::sgeqrf_ (int *m, int *n, Real *a, int *lda, Real *tau, Real *work, int *lwork, int *info)

  Single-precision GEneral matrix QR Factorization.
- void mtk::sormqr_ (char *side, char *trans, int *m, int *n, int *k, Real *a, int *lda, Real *tau, Real *c, int *ldc, Real *work, int *lwork, int *info)

  Single-precision Orthogonal Matrix from QR factorization.

17.77.1 Detailed Description

This class contains a collection of static classes, that posses direct access to the underlying structure of the matrices, thus allowing programmers to exploit some of the numerical methods implemented in the LAPACK.

The **LAPACK (Linear Algebra PACKage)** is written in Fortran 90 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems.

See also

http://www.netlib.org/lapack/

Todo Write documentation using LaTeX.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_lapack_adapter.cc.

17.78 mtk_lapack_adapter.cc

```
00001
00021 /*
00022 Copyright (C) 2015, Computational Science Research Center, San Diego State
00023 University. All rights reserved.
00024
00025 Redistribution and use in source and binary forms, with or without modification,
00026 are permitted provided that the following conditions are met:
00027
00028 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00029 and a copy of the modified files should be reported once modifications are
00030 completed, unless these modifications are made through the project's GitHub
00031 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00032 should be developed and included in any deliverable.
00033
00034 2. Redistributions of source code must be done through direct
00035 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00036
00037 3. Redistributions in binary form must reproduce the above copyright notice,
00038 this list of conditions and the following disclaimer in the documentation and/or
00039 other materials provided with the distribution.
00040
00041 4. Usage of the binary form on proprietary applications shall require explicit
00042 prior written permission from the the copyright holders, and due credit should
00043 be given to the copyright holders.
00044
00045 5. Neither the name of the copyright holder nor the names of its contributors
00046 may be used to endorse or promote products derived from this software without
00047 specific prior written permission.
00048
00049 The copyright holders provide no reassurances that the source code provided does
00050 not infringe any patent, copyright, or any other intellectual property rights of
00051 third parties. The copyright holders disclaim any liability to any recipient for
00052 claims brought against recipient by any third party for infringement of that
00053 parties intellectual property rights.
00054
00055 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00056 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00057 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00058 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00059 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00060 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00061 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00062 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00063 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00064 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00065 */
00066
00067 #include <cstring>
00068
00069 #include <iostream>
00070 #include <iomanip>
```

```
00071
00072 #include <algorithm>
00073
00074 #include "mtk_tools.h"
00075 #include "mtk_dense_matrix.h"
00076 #include "mtk_lapack_adapter.h"
00077
00078 namespace mtk {
00079
00080 extern "C" {
00081
00082 #ifdef MTK_PRECISION_DOUBLE
00083
00102 void dgesv_(int* n,
00103             int* nrhs,
00104             Real* a,
00105             int* lda,
00106             int* ipiv,
00107             Real* b,
00108             int* ldb,
00109             int* info);
00110 #else
00111
00130 void sgesv_(int* n,
00131             int* nrhs,
00132             Real* a,
00133             int* lda,
00134             int* ipiv,
00135             Real* b,
00136             int* ldb,
00137             int* info);
00138 #endif
00139
00140 #ifdef MTK_PRECISION_DOUBLE
00141
00184 void dgels_(char* trans,
00185             int* m,
00186             int* n,
00187             int* nrhs,
00188             Real* a,
00189             int* lda,
00190             Real* b,
00191             int* ldb,
00192             Real* work,
00193             int* lwork,
00194             int* info);
00195 #else
00196
00239 void sgels_(char* trans,
00240             int* m,
00241             int* n,
00242             int* nrhs,
00243             Real* a,
00244             int* lda,
00245             Real* b,
00246             int* ldb,
00247             Real* work,
00248             int* lwork,
00249             int* info);
00250 #endif
00251
00252 #ifdef MTK_PRECISION_DOUBLE
00253
00282 void dgeqrf_(int *m,
00283              int *n,
00284              Real *a,
00285              int *lda,
00286              Real *tau,
00287              Real *work,
00288              int *lwork,
00289              int *info);
00290 #else
00291
00320 void sgeqrf_(int *m,
00321              int *n,
00322              Real *a,
00323              int *lda,
00324              Real *tau,
00325              Real *work,
00326              int *lwork,
00327              int *info);
```

```
00328 #endif
00329
00330 #ifdef MTK_PRECISION_DOUBLE
00331
00365 void dormqr_(char *side,
00366              char *trans,
00367              int *m,
00368              int *n,
00369              int *k,
00370              Real *a,
00371              int *lda,
00372              Real *tau,
00373              Real *c,
00374              int *ldc,
00375              Real *work,
00376              int *lwork,
00377              int *info);
00378 #else
00379
00413 void sormqr_(char *side,
00414              char *trans,
00415              int *m,
00416              int *n,
00417              int *k,
00418              Real *a,
00419              int *lda,
00420              Real *tau,
00421              Real *c,
00422              int *ldc,
00423              Real *work,
00424              int *lwork,
00425              int *info);
00426 #endif
00427 }
00428 }
00429
00430 int mtk::LAPACKAdapter::SolveDenseSystem(
     mtk::DenseMatrix &mm,
00431                                          mtk::Real *rhs) {
00432
00433   #if MTK_DEBUG_LEVEL > 0
00434   mtk::Tools::Prevent(rhs == nullptr, __FILE__, __LINE__, __func__);
00435   #endif
00436
00437   int *ipiv{};                // Array for pivoting information.
00438   int nrhs{1};                // Number of right-hand sides.
00439   int info{};                 // Status of the solution.
00440   int mm_rank{mm.num_rows()}; // Rank of the matrix.
00441
00442   try {
00443     ipiv = new int[mm_rank];
00444   } catch (std::bad_alloc &memory_allocation_exception) {
00445     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00446       std::endl;
00447     std::cerr << memory_allocation_exception.what() << std::endl;
00448   }
00449   memset(ipiv, 0, sizeof(ipiv[0])*mm_rank);
00450
00451   int ldbb = mm_rank;
00452   int mm_ld = mm_rank;
00453
00454   #ifdef MTK_PRECISION_DOUBLE
00455   dgesv_(&mm_rank, &nrhs, mm.data(), &mm_ld, ipiv, rhs, &ldbb, &info);
00456   #else
00457   fgesv_(&mm_rank, &nrhs, mm.data(), &mm_ld, ipiv, rhs, &ldbb, &info);
00458   #endif
00459
00460   delete [] ipiv;
00461
00462   return info;
00463 }
00464
00465 int mtk::LAPACKAdapter::SolveDenseSystem(
     mtk::DenseMatrix &mm,
00466                                          mtk::DenseMatrix &bb) {
00467
00468   int nrhs{bb.num_rows()};  // Number of right-hand sides.
00469
00470   #if MTK_DEBUG_LEVEL > 0
00471   mtk::Tools::Prevent(nrhs <= 0, __FILE__, __LINE__, __func__);
00472   #endif
```

```
00473
00474   int *ipiv{};                // Array for pivoting information.
00475   int info{};                 // Status of the solution.
00476   int mm_rank{mm.num_rows()}; // Rank of the matrix.
00477
00478   try {
00479     ipiv = new int[mm_rank];
00480   } catch (std::bad_alloc &memory_allocation_exception) {
00481     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00482       std::endl;
00483     std::cerr << memory_allocation_exception.what() << std::endl;
00484   }
00485   memset(ipiv, 0, sizeof(ipiv[0])*mm_rank);
00486
00487   int ldbb = mm_rank;
00488   int mm_ld = mm_rank;
00489
00490   #ifdef MTK_PRECISION_DOUBLE
00491   dgesv_(&mm_rank, &nrhs, mm.data(), &mm_ld, ipiv, bb.data(), &ldbb, &info);
00492   #else
00493   fgesv_(&mm_rank, &nrhs, mm.data(), &mm_ld, ipiv, bb.data(), &ldbb, &info);
00494   #endif
00495
00496   delete [] ipiv;
00497
00498   // After output, the data in the matrix will be column-major ordered.
00499
00500   bb.SetOrdering(mtk::COL_MAJOR);
00501
00502   #if MTK_DEBUG_LEVEL > 0
00503   std::cout << "bb_col_maj_ord =" << std::endl;
00504   std::cout << bb << std::endl;
00505   #endif
00506
00507   bb.OrderRowMajor();
00508
00509   #if MTK_DEBUG_LEVEL > 0
00510   std::cout << "bb_row_maj_ord =" << std::endl;
00511   std::cout << bb << std::endl;
00512   #endif
00513
00514   return info;
00515 }
00516
00517 int mtk::LAPACKAdapter::SolveDenseSystem(
      mtk::DenseMatrix &mm,
00518                                          mtk::UniStgGrid1D &rhs) {
00519
00520   int nrhs{1};  // Number of right-hand sides.
00521
00522   int *ipiv{};                // Array for pivoting information.
00523   int info{};                 // Status of the solution.
00524   int mm_rank{mm.num_rows()}; // Rank of the matrix.
00525
00526   try {
00527     ipiv = new int[mm_rank];
00528   } catch (std::bad_alloc &memory_allocation_exception) {
00529     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00530       std::endl;
00531     std::cerr << memory_allocation_exception.what() << std::endl;
00532   }
00533   memset(ipiv, 0, sizeof(ipiv[0])*mm_rank);
00534
00535   int ldbb = mm_rank;
00536   int mm_ld = mm_rank;
00537
00538   mm.OrderColMajor();
00539
00540   #ifdef MTK_PRECISION_DOUBLE
00541   dgesv_(&mm_rank, &nrhs, mm.data(), &mm_ld, ipiv,
00542          rhs.discrete_field_u(), &ldbb, &info);
00543   #else
00544   fgesv_(&mm_rank, &nrhs, mm.data(), &mm_ld, ipiv,
00545          rhs.discrete_field_u(), &ldbb, &info);
00546   #endif
00547
00548   mm.OrderRowMajor();
00549
00550   delete [] ipiv;
00551
00552   return info;
```

```
00553 }
00554
00555 mtk::DenseMatrix mtk::LAPACKAdapter::QRFactorDenseMatrix
      (mtk::DenseMatrix &aa) {
00556
00557   mtk::Real *work{}; // Working array.
00558   mtk::Real *tau{};  // Array for the Householder scalars.
00559
00560   // Prepare to factorize: allocate and inquire for the value of lwork.
00561   try {
00562     work = new mtk::Real[1];
00563   } catch (std::bad_alloc &memory_allocation_exception) {
00564     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00565       std::endl;
00566     std::cerr << memory_allocation_exception.what() << std::endl;
00567   }
00568   memset(work, mtk::kZero, sizeof(aa.data()[0])*1);
00569
00570   int lwork{-1};
00571   int info{};
00572
00573   int aa_num_cols = aa.num_cols();
00574   int aaT_num_rows = aa.num_cols();
00575   int aaT_num_cols = aa.num_rows();
00576
00577   #if MTK_DEBUG_LEVEL > 0
00578   std::cout << "Input matrix BEFORE QR factorization:" << std::endl;
00579   std::cout << aa << std::endl;
00580   #endif
00581
00582   #ifdef MTK_PRECISION_DOUBLE
00583   dgeqrf_(&aaT_num_rows, &aaT_num_cols, aa.data(), &aaT_num_rows,
00584           tau,
00585           work, &lwork, &info);
00586   #else
00587   fgeqrf_(&aaT_num_rows, &aaT_num_cols, aa.data(), &aaT_num_rows,
00588           tau,
00589           work, &lwork, &info);
00590   #endif
00591
00592   #if MTK_DEBUG_LEVEL > 0
00593   if (info == 0) {
00594     lwork = (int) work[0];
00595   } else {
00596     std::cerr << "Could not get value for lwork on line " << __LINE__ - 5 <<
00597       std::endl;
00598     std::cerr << "Exiting..." << std::endl;
00599   }
00600   #endif
00601
00602   #if MTK_DEBUG_LEVEL>0
00603   std::cout << "lwork = " << std::endl << std::setw(12) << lwork << std::endl
00604     << std::endl;
00605   #endif
00606
00607   delete [] work;
00608   work = nullptr;
00609
00610   // Once we know lwork, we can actually invoke the factorization:
00611   try {
00612     work = new mtk::Real [lwork];
00613   } catch (std::bad_alloc &memory_allocation_exception) {
00614     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00615       std::endl;
00616     std::cerr << memory_allocation_exception.what() << std::endl;
00617   }
00618   memset(work, mtk::kZero, sizeof(work[0])*lwork);
00619
00620   int ltau = std::min(aaT_num_rows,aaT_num_cols);
00621
00622   try {
00623     tau = new mtk::Real [ltau];
00624   } catch (std::bad_alloc &memory_allocation_exception) {
00625     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00626       std::endl;
00627     std::cerr << memory_allocation_exception.what() << std::endl;
00628   }
00629   memset(tau, mtk::kZero, sizeof(0.0)*ltau);
00630
00631   #ifdef MTK_PRECISION_DOUBLE
00632   dgeqrf_(&aaT_num_rows, &aaT_num_cols, aa.data(), &aaT_num_rows,
```

```
00633           tau, work, &lwork, &info);
00634   #else
00635   fgeqrf_(&aaT_num_rows, &aaT_num_cols, aa.data(), &aaT_num_rows,
00636           tau, work, &lwork, &info);
00637   #endif
00638
00639   if (!info) {
00640     #if MTK_DEBUG_LEVEL > 0
00641     std::cout << "QR factorization completed!" << std::endl << std::endl;
00642     #endif
00643   } else {
00644     std::cerr << "Error solving system! info = " << info << std::endl;
00645     std::cerr << "Exiting..." << std::endl;
00646   }
00647
00648   #if MTK_DEBUG_LEVEL > 0
00649   std::cout << "Input matrix AFTER QR factorization:" << std::endl;
00650   std::cout << aa << std::endl;
00651   #endif
00652
00653   // We now generate the real matrix Q with orthonormal columns. This has to
00654   // be done separately since the actual output of dgeqrf_ (AA_) represents
00655   // the orthogonal matrix Q as a product of min(aa_num_rows,aa_num_cols)
00656   // elementary Householder reflectors. Notice that we must re-inquire the new
00657   // value for lwork that is used.
00658
00659   bool padded{false};
00660
00661   bool transpose{false};
00662
00663   mtk::DenseMatrix QQ_(aa.num_cols(),padded,transpose);
00664
00665   #if MTK_DEBUG_LEVEL > 0
00666   std::cout << "Initialized QQ_T: " << std::endl;
00667   std::cout << QQ_ << std::endl;
00668   #endif
00669
00670   // Assemble the QQ_ matrix:
00671   lwork = -1;
00672
00673   delete[] work;
00674   work = nullptr;
00675
00676   try {
00677     work = new mtk::Real[1];
00678   } catch (std::bad_alloc &memory_allocation_exception) {
00679     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00680       std::endl;
00681     std::cerr << memory_allocation_exception.what() <<
00682       std::endl;
00683   }
00684   memset(work, mtk::kZero, sizeof(work[0])*1);
00685
00686   char side_{'L'};
00687   char trans_{'N'};
00688
00689   int aux = QQ_.num_rows();
00690
00691   #ifdef MTK_PRECISION_DOUBLE
00692   dormqr_(&side_, &trans_,
00693           &aa_num_cols, &aa_num_cols, &ltau, aa.data(), &aaT_num_rows, tau,
00694           QQ_.data(), &aux, work, &lwork, &info);
00695   #else
00696   formqr_(&side_, &trans_,
00697           &aa_num_cols, &aa_num_cols, &ltau, aa.data(), &aaT_num_rows, tau,
00698           QQ_.data(), &aux, work, &lwork, &info);
00699   #endif
00700
00701   #if MTK_DEBUG_LEVEL > 0
00702   if (info == 0) {
00703     lwork = (int) work[0];
00704   } else {
00705     std::cerr << "Could not get lwork on line " << __LINE__ - 5 << std::endl;
00706     std::cerr << "Exiting..." << std::endl;
00707   }
00708   #endif
00709
00710   #if MTK_DEBUG_LEVEL > 0
00711   std::cout << "lwork = " << std::endl << std::setw(12) << lwork <<
00712     std::endl << std::endl;
00713   #endif
```

```
00714
00715   delete[] work;
00716   work = nullptr;
00717
00718   try {
00719     work = new mtk::Real[lwork];
00720   } catch (std::bad_alloc &memory_allocation_exception) {
00721     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
00722       std::endl;
00723     std::cerr << memory_allocation_exception.what() << std::endl;
00724   }
00725   memset(work, mtk::kZero, sizeof(work[0])*lwork);
00726
00727   #ifdef MTK_PRECISION_DOUBLE
00728   dormqr_(&side_, &trans_,
00729           &aa_num_cols, &aa_num_cols, &ltau, aa.data(), &aaT_num_rows, tau,
00730           QQ_.data(), &aux, work, &lwork, &info);
00731   #else
00732   formqr_(&side_, &trans_,
00733           &aa_num_cols, &aa_num_cols, &ltau, aa.data(), &aaT_num_rows, tau,
00734           QQ_.data(), &aux, work, &lwork, &info);
00735   #endif
00736
00737   if (!info) {
00738     #if MTK_DEBUG_LEVEL>0
00739     std::cout << "Q matrix successfully assembled!" << std::endl << std::endl;
00740     #endif
00741   } else {
00742     std::cerr << "Something went wrong solving system! info = " << info <<
00743       std::endl;
00744     std::cerr << "Exiting..." << std::endl;
00745   }
00746
00747   delete[] work;
00748   work = nullptr;
00749
00750   delete[] tau;
00751   tau = nullptr;
00752
00753   return QQ_;
00754 }
00755
00756 int mtk::LAPACKAdapter::SolveRectangularDenseSystem(const
      mtk::DenseMatrix &aa,
00757                                                     mtk::Real *ob_,
00758                                                     int ob_ld_) {
00759
00760   // We first invoke the solver to query for the value of lwork. For this,
00761   // we must at least allocate enough space to allow access to WORK(1), or
00762   // work[0]:
00763
00764   // If LWORK = -1, then a workspace query is assumed; the routine only
00765   // calculates the optimal size of the WORK array, returns this value as
00766   // the first entry of the WORK array, and no error message related to
00767   // LWORK is issued by XERBLA.
00768
00769   mtk::Real *work{}; // Work array.
00770
00771   try {
00772     work = new mtk::Real[1];
00773   } catch (std::bad_alloc &memory_allocation_exception) {
00774     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 << std::endl;
00775     std::cerr << memory_allocation_exception.what() << std::endl;
00776   }
00777   memset(work, mtk::kZero, sizeof(work[0])*1);
00778
00779   char trans_{'N'};
00780   int nrhs_{1};
00781   int info{0};
00782   int lwork{-1};
00783
00784   int AA_num_rows_  = aa.num_cols();
00785   int AA_num_cols_  = aa.num_rows();
00786   int AA_ld_ = std::max(1,aa.num_cols());
00787
00788   #ifdef MTK_PRECISION_DOUBLE
00789   dgels_(&trans_, &AA_num_rows_, &AA_num_cols_, &nrhs_, aa.data(), &AA_ld_,
00790          ob_, &ob_ld_,
00791          work, &lwork, &info);
00792   #else
00793   sgels_(&trans_, &AA_num_rows_, &AA_num_cols_, &nrhs_, aa.data(), &AA_ld_,
```

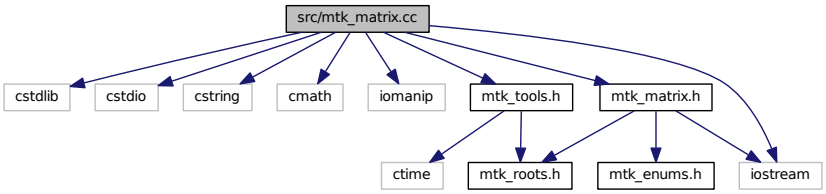
```
00794           ob_, &ob_ld_,
00795           work, &lwork, &info);
00796   #endif
00797
00798   if (info == 0) {
00799     lwork = (int) work[0];
00800   } else {
00801     std::cerr << "Could not get value for lwork on line " << __LINE__ - 2 <<
00802       std::endl;
00803     std::cerr << "Exiting..." << std::endl;
00804     return info;
00805   }
00806
00807   #if MTK_DEBUG_LEVEL > 0
00808   std::cout << "lwork = " << std::endl << std::setw(12)<< lwork <<
00809     std::endl << std::endl;
00810   #endif
00811
00812   // We then use lwork's new value to create the work array:
00813   delete[] work;
00814   work = nullptr;
00815
00816   try {
00817     work = new mtk::Real[lwork];
00818   } catch (std::bad_alloc &memory_allocation_exception) {
00819     std::cerr << "Memory allocation exception on line " << __LINE__ - 3 << std::endl;
00820     std::cerr << memory_allocation_exception.what() << std::endl;
00821   }
00822   memset(work, 0.0, sizeof(work[0])*lwork);
00823
00824   // We now invoke the solver again:
00825   #ifdef MTK_PRECISION_DOUBLE
00826   dgels_(&trans_, &AA_num_rows_, &AA_num_cols_, &nrhs_, aa.data(), &AA_ld_,
00827          ob_, &ob_ld_,
00828          work, &lwork, &info);
00829   #else
00830   sgels_(&trans_, &AA_num_rows_, &AA_num_cols_, &nrhs_, aa.data(), &AA_ld_,
00831          ob_, &ob_ld_,
00832          work, &lwork, &info);
00833   #endif
00834
00835   delete [] work;
00836   work = nullptr;
00837
00838   return info;
00839 }
```

17.79 src/mtk_matrix.cc File Reference

Implementing the representation of a matrix in the MTK.

```
#include <cstdlib>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <iomanip>
#include <iostream>
#include "mtk_tools.h"
#include "mtk_matrix.h"
```

Include dependency graph for mtk_matrix.cc:



17.79.1 Detailed Description

Implementation of the representation for the matrices implemented in the MTK.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_matrix.cc.

17.80 mtk_matrix.cc

```
00001
00010 /*
00011 Copyright (C) 2015, Computational Science Research Center, San Diego State
00012 University. All rights reserved.
00013
00014 Redistribution and use in source and binary forms, with or without modification,
00015 are permitted provided that the following conditions are met:
00016
00017 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00018 and a copy of the modified files should be reported once modifications are
00019 completed, unless these modifications are made through the project's GitHub
00020 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00021 should be developed and included in any deliverable.
00022
00023 2. Redistributions of source code must be done through direct
00024 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00025
00026 3. Redistributions in binary form must reproduce the above copyright notice,
00027 this list of conditions and the following disclaimer in the documentation and/or
00028 other materials provided with the distribution.
00029
00030 4. Usage of the binary form on proprietary applications shall require explicit
00031 prior written permission from the the copyright holders, and due credit should
00032 be given to the copyright holders.
00033
00034 5. Neither the name of the copyright holder nor the names of its contributors
00035 may be used to endorse or promote products derived from this software without
00036 specific prior written permission.
00037
00038 The copyright holders provide no reassurances that the source code provided does
00039 not infringe any patent, copyright, or any other intellectual property rights of
00040 third parties. The copyright holders disclaim any liability to any recipient for
00041 claims brought against recipient by any third party for infringement of that
00042 parties intellectual property rights.
00043
00044 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00045 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00046 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00047 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00048 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
```

```
00049 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00050 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00051 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00052 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00053 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00054 */
00055
00056 #include <cstdlib>
00057 #include <cstdio>
00058 #include <cstring>
00059 #include <cmath>
00060
00061 #include <iomanip>
00062 #include <iostream>
00063
00064 #include "mtk_tools.h"
00065 #include "mtk_matrix.h"
00066
00067 mtk::Matrix::Matrix():
00068   storage_(mtk::DENSE),
00069   ordering_(mtk::ROW_MAJOR),
00070   num_rows_(),
00071   num_cols_(),
00072   num_values_(),
00073   ld_(),
00074   num_zero_(),
00075   num_non_zero_(),
00076   num_null_(),
00077   num_non_null_(),
00078   kl_(),
00079   ku_(),
00080   bandwidth_(),
00081   abs_density_(),
00082   rel_density_(),
00083   abs_sparsity_(),
00084   rel_sparsity_() {}
00085
00086 mtk::Matrix::Matrix(const Matrix &in):
00087   storage_(in.storage_),
00088   ordering_(in.ordering_),
00089   num_rows_(in.num_rows_),
00090   num_cols_(in.num_cols_),
00091   num_values_(in.num_values_),
00092   ld_(in.ld_),
00093   num_zero_(in.num_zero_),
00094   num_non_zero_(in.num_non_zero_),
00095   num_null_(in.num_null_),
00096   num_non_null_(in.num_non_null_),
00097   kl_(in.kl_),
00098   ku_(in.ku_),
00099   bandwidth_(in.bandwidth_),
00100   abs_density_(in.abs_density_),
00101   rel_density_(in.rel_density_),
00102   abs_sparsity_(in.abs_sparsity_),
00103   rel_sparsity_(in.rel_sparsity_) {}
00104
00105 mtk::Matrix::~Matrix() noexcept {}
00106
00107 mtk::MatrixStorage mtk::Matrix::storage() const noexcept {
00108
00109   return storage_;
00110 }
00111
00112 mtk::MatrixOrdering mtk::Matrix::ordering() const noexcept {
00113
00114   return ordering_;
00115 }
00116
00117 int mtk::Matrix::num_rows() const noexcept {
00118
00119   return num_rows_;
00120 }
00121
00122 int mtk::Matrix::num_cols() const noexcept {
00123
00124   return num_cols_;
00125 }
00126
00127 int mtk::Matrix::num_values() const noexcept {
00128
00129   return num_values_;
```

```
00130 }
00131
00132 int mtk::Matrix::ld() const noexcept {
00133
00134   return ld_;
00135 }
00136
00137 int mtk::Matrix::num_zero() const noexcept {
00138
00139   return num_zero_;
00140 }
00141
00142 int mtk::Matrix::num_non_zero() const noexcept {
00143
00144   return num_non_zero_;
00145 }
00146
00147 int mtk::Matrix::num_null() const noexcept {
00148
00149   return num_null_;
00150 }
00151
00152 int mtk::Matrix::num_non_null() const noexcept {
00153
00154   return num_non_null_;
00155 }
00156
00157 int mtk::Matrix::kl() const noexcept {
00158
00159   return kl_;
00160 }
00161
00162 int mtk::Matrix::ku() const noexcept {
00163
00164   return ku_;
00165 }
00166
00167 int mtk::Matrix::bandwidth() const noexcept {
00168
00169   return bandwidth_;
00170 }
00171
00172 mtk::Real mtk::Matrix::rel_density() const noexcept {
00173
00174   return rel_density_;
00175 }
00176
00177 mtk::Real mtk::Matrix::abs_sparsity() const noexcept {
00178
00179   return abs_sparsity_;
00180 }
00181
00182 mtk::Real mtk::Matrix::rel_sparsity() const noexcept {
00183
00184   return rel_sparsity_;
00185 }
00186
00187 void mtk::Matrix::set_storage(const mtk::MatrixStorage &ss)
     noexcept {
00188
00189   #if MTK_DEBUG_LEVEL > 0
00190   mtk::Tools::Prevent(!(ss == mtk::DENSE ||
00191                         ss == mtk::BANDED ||
00192                         ss == mtk::CRS),
00193                       __FILE__, __LINE__, __func__);
00194   #endif
00195
00196   storage_ = ss;
00197 }
00198
00199 void mtk::Matrix::set_ordering(const
     mtk::MatrixOrdering &oo) noexcept {
00200
00201   #if MTK_DEBUG_LEVEL > 0
00202   mtk::Tools::Prevent(!(oo == mtk::ROW_MAJOR || oo ==
     mtk::COL_MAJOR),
00203                       __FILE__, __LINE__, __func__);
00204   #endif
00205
00206   ordering_ = oo;
00207
```

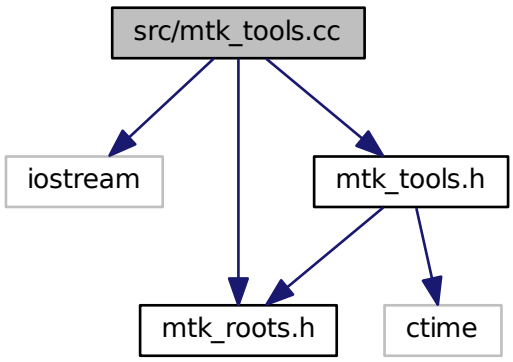
```
00208   ld_ = (ordering_ == mtk::ROW_MAJOR)?
00209     std::max(1,num_cols_): std::max(1,num_rows_);
00210 }
00211
00212 void mtk::Matrix::set_num_rows(const int &in) noexcept {
00213
00214   #if MTK_DEBUG_LEVEL > 0
00215   mtk::Tools::Prevent(in < 1, __FILE__, __LINE__, __func__);
00216   #endif
00217
00218   num_rows_ = in;
00219   num_values_ = num_rows_*num_cols_;
00220   ld_ = (ordering_ == mtk::ROW_MAJOR)?
00221     std::max(1,num_cols_): std::max(1,num_rows_);
00222 }
00223
00224 void mtk::Matrix::set_num_cols(const int &in) noexcept {
00225
00226   #if MTK_DEBUG_LEVEL > 0
00227   mtk::Tools::Prevent(in < 1, __FILE__, __LINE__, __func__);
00228   #endif
00229
00230   num_cols_ = in;
00231   num_values_ = num_rows_*num_cols_;
00232   ld_ = (ordering_ == mtk::ROW_MAJOR)?
00233     std::max(1,num_cols_): std::max(1,num_rows_);
00234 }
00235
00236 void mtk::Matrix::set_num_zero(const int &in) noexcept {
00237
00238   #if MTK_DEBUG_LEVEL > 0
00239   mtk::Tools::Prevent(in < 0, __FILE__, __LINE__, __func__);
00240   #endif
00241
00242   num_zero_ = in;
00243   num_non_zero_ = num_values_ - num_zero_;
00244
00246   rel_density_ = (mtk::Real) num_non_zero_/num_values_;
00247   rel_sparsity_ = 1.0 - rel_density_;
00248 }
00249
00250 void mtk::Matrix::set_num_null(const int &in) noexcept {
00251
00252   #if MTK_DEBUG_LEVEL > 0
00253   mtk::Tools::Prevent(in < 0, __FILE__, __LINE__, __func__);
00254   #endif
00255
00256   num_null_ = in;
00257   num_non_null_ = num_values_ - num_null_;
00258
00260   abs_density_ = (mtk::Real) num_non_null_/num_values_;
00261   abs_sparsity_ = 1.0 - abs_density_;
00262 }
00263
00264 void mtk::Matrix::IncreaseNumZero() noexcept {
00265
00267
00268   num_zero_++;
00269   num_non_zero_ = num_values_ - num_zero_;
00270   rel_density_ = (mtk::Real) num_non_zero_/num_values_;
00271   rel_sparsity_ = 1.0 - rel_density_;
00272 }
00273
00274 void mtk::Matrix::IncreaseNumNull() noexcept {
00275
00277
00278   num_null_++;
00279   num_non_null_ = num_values_ - num_null_;
00280   abs_density_ = (mtk::Real) num_non_null_/num_values_;
00281   abs_sparsity_ = 1.0 - abs_density_;
00282 }
```

17.81 src/mtk_tools.cc File Reference

Implements a execution tool manager class.

```
#include <iostream>
#include "mtk_roots.h"
#include "mtk_tools.h"
```

Include dependency graph for mtk_tools.cc:



17.81.1 Detailed Description

Basic tools to ensure execution correctness.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_tools.cc.

17.82 mtk_tools.cc

```
00001
00010 /*
00011 Copyright (C) 2015, Computational Science Research Center, San Diego State
00012 University. All rights reserved.
00013
00014 Redistribution and use in source and binary forms, with or without modification,
00015 are permitted provided that the following conditions are met:
00016
00017 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00018 and a copy of the modified files should be reported once modifications are
00019 completed, unless these modifications are made through the project's GitHub
00020 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00021 should be developed and included in any deliverable.
00022
00023 2. Redistributions of source code must be done through direct
00024 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00025
00026 3. Redistributions in binary form must reproduce the above copyright notice,
00027 this list of conditions and the following disclaimer in the documentation and/or
00028 other materials provided with the distribution.
00029
00030 4. Usage of the binary form on proprietary applications shall require explicit
00031 prior written permission from the the copyright holders, and due credit should
```

```
00032 be given to the copyright holders.
00033
00034 5. Neither the name of the copyright holder nor the names of its contributors
00035 may be used to endorse or promote products derived from this software without
00036 specific prior written permission.
00037
00038 The copyright holders provide no reassurances that the source code provided does
00039 not infringe any patent, copyright, or any other intellectual property rights of
00040 third parties. The copyright holders disclaim any liability to any recipient for
00041 claims brought against recipient by any third party for infringement of that
00042 parties intellectual property rights.
00043
00044 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00045 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00046 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00047 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00048 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00049 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00050 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00051 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00052 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00053 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00054 */
00055
00056 #include <iostream>
00057
00058 #include "mtk_roots.h"
00059 #include "mtk_tools.h"
00060
00061 void mtk::Tools::Prevent(const bool condition,
00062                          const char *const fname,
00063                          int lineno,
00064                          const char *const fxname) noexcept {
00065
00067   #if MTK_DEBUG_LEVEL > 0
00068   if (lineno < 1) {
00069     std::cerr << __FILE__ << ": " << "Incorrect parameter at line " <<
00070     __LINE__ - 2 << " (" << __func__ << ")" << std::endl;
00072     exit(EXIT_FAILURE);
00073   }
00074   #endif
00075
00076   if (condition) {
00077     std::cerr << fname << ": " << "Incorrect parameter at line " <<
00078     lineno << " (" << fxname << ")" << std::endl;
00079     exit(EXIT_FAILURE);
00080   }
00081 }
00082
00084
00085 int mtk::Tools::test_number_;  // Used to control the correctness of the test.
00086
00087 mtk::Real mtk::Tools::duration_;  // Duration of the current test.
00088
00089 clock_t mtk::Tools::begin_time_;  // Used to time tests.
00090
00091 void mtk::Tools::BeginUnitTestNo(const int &nn) noexcept {
00092
00093   #if MTK_DEBUG_LEVEL > 0
00094   mtk::Tools::Prevent(nn <= 0, __FILE__, __LINE__, __func__);
00095   #endif
00096
00097   test_number_ = nn;
00098
00099   #if MTK_DEBUG_LEVEL > 0
00100   std::cout << "Beginning test " << nn << "." << std::endl;
00101   #endif
00102   begin_time_ = clock();
00103 }
00104
00105 void mtk::Tools::EndUnitTestNo(const int &nn) noexcept {
00106
00107   #if MTK_DEBUG_LEVEL > 0
00108   mtk::Tools::Prevent(test_number_ != nn, __FILE__, __LINE__, __func__);
00109   #endif
00110
00111   duration_ = mtk::Real(clock() - begin_time_)/CLOCKS_PER_SEC;
00112 }
00113
00114 void mtk::Tools::Assert(const bool &condition) noexcept {
```

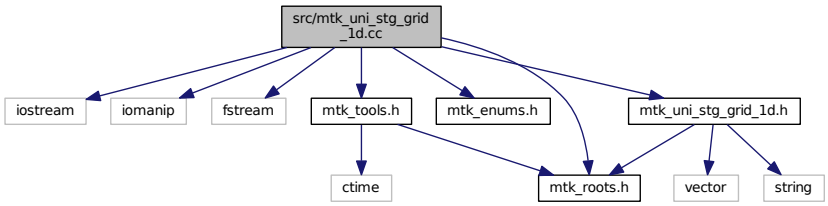
```
00115
00116   if (condition) {
00117     std::cout << "Test " << test_number_ << ": PASSED in " << duration_ <<
00118       " s." << std::endl;
00119   } else {
00120     std::cout << "Test " << test_number_ << ": FAILED in " << duration_ <<
00121       " s." << std::endl;
00122   }
00123 }
```

17.83 src/mtk_uni_stg_grid_1d.cc File Reference

Implementation of an 1D uniform staggered grid.

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include "mtk_roots.h"
#include "mtk_enums.h"
#include "mtk_tools.h"
#include "mtk_uni_stg_grid_1d.h"
```

Include dependency graph for mtk_uni_stg_grid_1d.cc:



Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

Functions

- std::ostream & mtk::operator<< (std::ostream &stream, mtk::UniStgGrid1D &in)

17.83.1 Detailed Description

Implementation of an 1D uniform staggered grid.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_uni_stg_grid_1d.cc.

17.84 mtk_uni_stg_grid_1d.cc

```
00001
00010 /*
00011 Copyright (C) 2015, Computational Science Research Center, San Diego State
00012 University. All rights reserved.
00013
00014 Redistribution and use in source and binary forms, with or without modification,
00015 are permitted provided that the following conditions are met:
00016
00017 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00018 and a copy of the modified files should be reported once modifications are
00019 completed, unless these modifications are made through the project's GitHub
00020 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00021 should be developed and included in any deliverable.
00022
00023 2. Redistributions of source code must be done through direct
00024 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00025
00026 3. Redistributions in binary form must reproduce the above copyright notice,
00027 this list of conditions and the following disclaimer in the documentation and/or
00028 other materials provided with the distribution.
00029
00030 4. Usage of the binary form on proprietary applications shall require explicit
00031 prior written permission from the the copyright holders, and due credit should
00032 be given to the copyright holders.
00033
00034 5. Neither the name of the copyright holder nor the names of its contributors
00035 may be used to endorse or promote products derived from this software without
00036 specific prior written permission.
00037
00038 The copyright holders provide no reassurances that the source code provided does
00039 not infringe any patent, copyright, or any other intellectual property rights of
00040 third parties. The copyright holders disclaim any liability to any recipient for
00041 claims brought against recipient by any third party for infringement of that
00042 parties intellectual property rights.
00043
00044 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00045 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00046 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00047 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00048 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00049 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00050 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00051 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00052 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00053 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00054 */
00055
00056 #include <iostream>
00057 #include <iomanip>
00058 #include <fstream>
00059
00060 #include "mtk_roots.h"
00061 #include "mtk_enums.h"
00062 #include "mtk_tools.h"
00063
00064 #include "mtk_uni_stg_grid_1d.h"
00065
00066 namespace mtk {
00067
00068 std::ostream& operator <<(std::ostream &stream, mtk::UniStgGrid1D &in) {
00069
00070   stream << '[' << in.west_bndy_x_ << ':' << in.num_cells_x_ << ':' <<
00071   in.east_bndy_x_ << "] = " << std::endl << std::endl;
00072
00074
00075   stream << "x:";
00076   for (unsigned int ii = 0; ii < in.discrete_domain_x_.size(); ++ii) {
00077     stream << std::setw(10) << in.discrete_domain_x_[ii];
00078   }
00079   stream << std::endl;
00080
00082
00083   if (in.nature_ == mtk::SCALAR) {
00084     stream << "u:";
00085   }
00086   else {
00087     stream << "v:";
00088   }
```

```
00089   for (unsigned int ii = 0; ii < in.discrete_field_u_.size(); ++ii) {
00090     stream << std::setw(10) << in.discrete_field_u_[ii];
00091   }
00092
00093   stream << std::endl;
00094
00095   return stream;
00096 }
00097 }
00098
00099 mtk::UniStgGrid1D::UniStgGrid1D():
00100     nature_(),
00101     discrete_domain_x_(),
00102     discrete_field_u_(),
00103     west_bndy_x_(),
00104     east_bndy_x_(),
00105     num_cells_x_(),
00106     delta_x_() {}
00107
00108 mtk::UniStgGrid1D::UniStgGrid1D(const
      UniStgGrid1D &grid):
00109     nature_(grid.nature_),
00110     west_bndy_x_(grid.west_bndy_x_),
00111     east_bndy_x_(grid.east_bndy_x_),
00112     num_cells_x_(grid.num_cells_x_),
00113     delta_x_(grid.delta_x_) {
00114
00115     std::copy(grid.discrete_domain_x_.begin(),
00116               grid.discrete_domain_x_.begin() + grid.
      discrete_domain_x_.size(),
00117               discrete_domain_x_.begin());
00118
00119     std::copy(grid.discrete_field_u_.begin(),
00120               grid.discrete_field_u_.begin() + grid.
      discrete_field_u_.size(),
00121               discrete_field_u_.begin());
00122 }
00123
00124 mtk::UniStgGrid1D::UniStgGrid1D(const Real &west_bndy_x,
00125                                 const Real &east_bndy_x,
00126                                 const int &num_cells_x,
00127                                 const mtk::FieldNature &nature) {
00128
00129   #if MTK_DEBUG_LEVEL > 0
00130   mtk::Tools::Prevent(west_bndy_x < mtk::kZero, __FILE__, __LINE__, __func__);
00131   mtk::Tools::Prevent(east_bndy_x < mtk::kZero, __FILE__, __LINE__, __func__);
00132   mtk::Tools::Prevent(east_bndy_x <= west_bndy_x, __FILE__, __LINE__, __func__);
00133   mtk::Tools::Prevent(num_cells_x < 0, __FILE__, __LINE__, __func__);
00134   #endif
00135
00136   nature_ = nature;
00137   west_bndy_x_ = west_bndy_x;
00138   east_bndy_x_ = east_bndy_x;
00139   num_cells_x_ = num_cells_x;
00140
00141   delta_x_ = (east_bndy_x - west_bndy_x)/((mtk::Real) num_cells_x);
00142 }
00143
00144 mtk::UniStgGrid1D::~UniStgGrid1D() {}
00145
00146 mtk::Real mtk::UniStgGrid1D::west_bndy_x() const {
00147
00148   return west_bndy_x_;
00149 }
00150
00151 mtk::Real mtk::UniStgGrid1D::east_bndy_x() const {
00152
00153   return east_bndy_x_;
00154 }
00155
00156 mtk::Real mtk::UniStgGrid1D::delta_x() const {
00157
00158   return delta_x_;
00159 }
00160
00161 const mtk::Real *mtk::UniStgGrid1D::discrete_domain_x() const
      {
00162
00163   return discrete_domain_x_.data();
00164 }
00165
```

```
00166 mtk::Real *mtk::UniStgGrid1D::discrete_field_u() {
00167
00168   return discrete_field_u_.data();
00169 }
00170
00171 int mtk::UniStgGrid1D::num_cells_x() const {
00172
00173   return num_cells_x_;
00174 }
00175
00176 void mtk::UniStgGrid1D::BindScalarField(
00177     mtk::Real (*ScalarField)(mtk::Real xx)) {
00178
00179   #if MTK_DEBUG_LEVEL > 0
00180   mtk::Tools::Prevent(nature_ == mtk::VECTOR, __FILE__, __LINE__, __func__);
00181   #endif
00182
00184
00185   discrete_domain_x_.reserve(num_cells_x_ + 2);
00186
00187   discrete_domain_x_.push_back(west_bndy_x_);
00188   #ifdef MTK_PRECISION_DOUBLE
00189   auto first_center = west_bndy_x_ + delta_x_/2.0;
00190   #else
00191   auto first_center = west_bndy_x_ + delta_x_/2.0f;
00192   #endif
00193   discrete_domain_x_.push_back(first_center);
00194   for (auto ii = 1; ii < num_cells_x_; ++ii) {
00195     discrete_domain_x_.push_back(first_center + ii*delta_x_);
00196   }
00197   discrete_domain_x_.push_back(east_bndy_x_);
00198
00200
00201   discrete_field_u_.reserve(num_cells_x_ + 2);
00202
00203   discrete_field_u_.push_back(ScalarField(west_bndy_x_));
00204
00205   discrete_field_u_.push_back(ScalarField(first_center));
00206   for (auto ii = 1; ii < num_cells_x_; ++ii) {
00207     discrete_field_u_.push_back(ScalarField(first_center + ii*delta_x_));
00208   }
00209   discrete_field_u_.push_back(ScalarField(east_bndy_x_));
00210 }
00211
00212 void mtk::UniStgGrid1D::BindVectorField(
00213     mtk::Real (*VectorField)(mtk::Real xx)) {
00214
00215   #if MTK_DEBUG_LEVEL > 0
00216   mtk::Tools::Prevent(nature_ == mtk::SCALAR, __FILE__, __LINE__, __func__);
00217   #endif
00218
00220
00221   discrete_domain_x_.reserve(num_cells_x_ + 1);
00222
00223   discrete_domain_x_.push_back(west_bndy_x_);
00224   for (auto ii = 1; ii < num_cells_x_; ++ii) {
00225     discrete_domain_x_.push_back(west_bndy_x_ + ii*delta_x_);
00226   }
00227   discrete_domain_x_.push_back(east_bndy_x_);
00228
00230
00231   discrete_field_u_.reserve(num_cells_x_ + 1);
00232
00233   discrete_field_u_.push_back(VectorField(west_bndy_x_));
00234   for (auto ii = 1; ii < num_cells_x_; ++ii) {
00235     discrete_field_u_.push_back(VectorField(west_bndy_x_ + ii*delta_x_));
00236   }
00237   discrete_field_u_.push_back(VectorField(east_bndy_x_));
00238 }
00239
00240 bool mtk::UniStgGrid1D::WriteToFile(std::string filename,
00241                                     std::string space_name,
00242                                     std::string field_name) const {
00243
00244   std::ofstream output_dat_file;  // Output file.
00245
00246   output_dat_file.open(filename);
00247
00248   if (!output_dat_file.is_open()) {
00249     return false;
00250   }
```

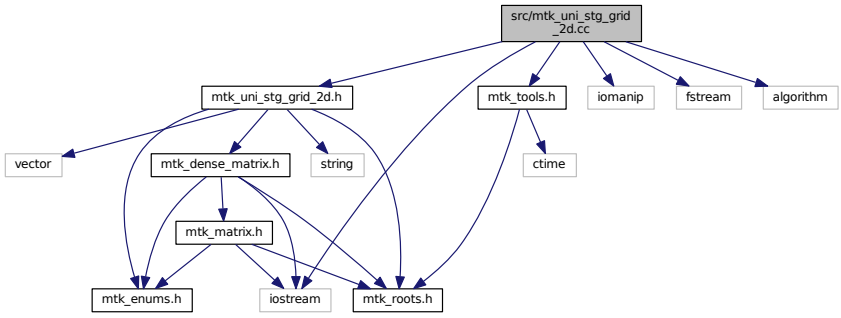
```
00251
00252   output_dat_file << "# " << space_name <<  ' ' << field_name << std::endl;
00253   for (unsigned int ii = 0; ii < discrete_domain_x_.size(); ++ii) {
00254     output_dat_file << discrete_domain_x_[ii] << ' ' << discrete_field_u_[ii] <<
00255       std::endl;
00256   }
00257
00258   output_dat_file.close();
00259
00260   return true;
00261 }
```

17.85 src/mtk_uni_stg_grid_2d.cc File Reference

Implementation of a 2D uniform staggered grid.

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <algorithm>
#include "mtk_tools.h"
#include "mtk_uni_stg_grid_2d.h"
```

Include dependency graph for mtk_uni_stg_grid_2d.cc:



Namespaces

- mtk

  Mimetic Methods Toolkit namespace.

Functions

- std::ostream & mtk::operator<< (std::ostream &stream, mtk::UniStgGrid2D &in)

17.85.1 Detailed Description

Implementation of a 2D uniform staggered grid.

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_uni_stg_grid_2d.cc.

17.86 mtk_uni_stg_grid_2d.cc

```
00001
00010 /*
00011 Copyright (C) 2015, Computational Science Research Center, San Diego State
00012 University. All rights reserved.
00013
00014 Redistribution and use in source and binary forms, with or without modification,
00015 are permitted provided that the following conditions are met:
00016
00017 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00018 and a copy of the modified files should be reported once modifications are
00019 completed, unless these modifications are made through the project's GitHub
00020 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00021 should be developed and included in any deliverable.
00022
00023 2. Redistributions of source code must be done through direct
00024 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00025
00026 3. Redistributions in binary form must reproduce the above copyright notice,
00027 this list of conditions and the following disclaimer in the documentation and/or
00028 other materials provided with the distribution.
00029
00030 4. Usage of the binary form on proprietary applications shall require explicit
00031 prior written permission from the the copyright holders, and due credit should
00032 be given to the copyright holders.
00033
00034 5. Neither the name of the copyright holder nor the names of its contributors
00035 may be used to endorse or promote products derived from this software without
00036 specific prior written permission.
00037
00038 The copyright holders provide no reassurances that the source code provided does
00039 not infringe any patent, copyright, or any other intellectual property rights of
00040 third parties. The copyright holders disclaim any liability to any recipient for
00041 claims brought against recipient by any third party for infringement of that
00042 parties intellectual property rights.
00043
00044 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00045 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00046 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00047 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00048 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00049 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00050 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00051 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00052 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00053 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00054 */
00055
00056 #include <iostream>
00057 #include <iomanip>
00058 #include <fstream>
00059
00060 #include <algorithm>
00061
00062 #include "mtk_tools.h"
00063 #include "mtk_uni_stg_grid_2d.h"
00064
00065 namespace mtk {
00066
00067 std::ostream& operator <<(std::ostream &stream, mtk::UniStgGrid2D &in) {
00068
00069   stream << '[' << in.west_bndy_ << ':' << in.num_cells_x_ << ':' <<
00070   in.east_bndy_ << "] x ";
00071
00072   stream << '[' << in.south_bndy_ << ':' << in.num_cells_y_ << ':' <<
00073   in.north_bndy_ << "] = " << std::endl << std::endl;
00074
00076
00077   stream << "x:";
```

```
00078   for (unsigned int ii = 0; ii < in.discrete_domain_x_.size(); ++ii) {
00079     stream << std::setw(10) << in.discrete_domain_x_[ii];
00080   }
00081   stream << std::endl;
00082
00083   stream << "y:";
00084   for (unsigned int ii = 0; ii < in.discrete_domain_y_.size(); ++ii) {
00085     stream << std::setw(10) << in.discrete_domain_y_[ii];
00086   }
00087   stream << std::endl;
00088
00090
00091   if (in.nature_ == mtk::SCALAR) {
00092     stream << "u:" << std::endl;
00093     if (in.discrete_field_.size() > 0) {
00094       for (int ii = 0; ii < in.num_cells_x_ + 2; ++ii) {
00095         for (int jj = 0; jj < in.num_cells_y_ + 2; ++jj) {
00096           stream << std::setw(10) << in.discrete_field_[ii*in.
      num_cells_y_ + jj];
00097         }
00098         stream << std::endl;
00099       }
00100     }
00101   } else {
00102
00103     int mm{in.num_cells_x_};
00104     int nn{in.num_cells_y_};
00105     int p_offset{nn*(mm + 1) - 1};
00106
00107     stream << "p(x,y):" << std::endl;
00108     for (int ii = 0; ii < nn; ++ii) {
00109       for (int jj = 0; jj < mm + 1; ++jj) {
00110         stream << std::setw(10) << in.discrete_field_[ii*(mm + 1) + jj];
00111       }
00112       stream << std::endl;
00113     }
00114     stream << std::endl;
00115
00116     stream << "q(x,y):" << std::endl;
00117     for (int ii = 0; ii < nn + 1; ++ii) {
00118       for (int jj = 0; jj < mm; ++jj) {
00119         stream << std::setw(10) <<
00120           in.discrete_field_[p_offset + ii*mm + jj];
00121       }
00122       stream << std::endl;
00123     }
00124     stream << std::endl;
00125   }
00126
00127   return stream;
00128 }
00129 }
00130
00131 mtk::UniStgGrid2D::UniStgGrid2D():
00132     discrete_domain_x_(),
00133     discrete_domain_y_(),
00134     discrete_field_(),
00135     nature_(),
00136     west_bndy_(),
00137     east_bndy_(),
00138     num_cells_x_(),
00139     delta_x_(),
00140     south_bndy_(),
00141     north_bndy_(),
00142     num_cells_y_(),
00143     delta_y_() {}
00144
00145 mtk::UniStgGrid2D::UniStgGrid2D(const
      UniStgGrid2D &grid):
00146     nature_(grid.nature_),
00147     west_bndy_(grid.west_bndy_),
00148     east_bndy_(grid.east_bndy_),
00149     num_cells_x_(grid.num_cells_x_),
00150     delta_x_(grid.delta_x_),
00151     south_bndy_(grid.south_bndy_),
00152     north_bndy_(grid.north_bndy_),
00153     num_cells_y_(grid.num_cells_y_),
00154     delta_y_(grid.delta_y_) {
00155
00156     std::copy(grid.discrete_domain_x_.begin(),
00157               grid.discrete_domain_x_.begin() + grid.
```

```
    discrete_domain_x_.size(),
00158               discrete_domain_x_.begin());
00159
00160     std::copy(grid.discrete_domain_y_.begin(),
00161               grid.discrete_domain_y_.begin() + grid.
    discrete_domain_y_.size(),
00162               discrete_domain_y_.begin());
00163
00164     std::copy(grid.discrete_field_.begin(),
00165               grid.discrete_field_.begin() + grid.discrete_field_.size(),
00166               discrete_field_.begin());
00167 }
00168
00169 mtk::UniStgGrid2D::UniStgGrid2D(const Real &west_bndy,
00170                                 const Real &east_bndy,
00171                                 const int &num_cells_x,
00172                                 const Real &south_bndy,
00173                                 const Real &north_bndy,
00174                                 const int &num_cells_y,
00175                                 const mtk::FieldNature &nature) {
00176
00177   #if MTK_DEBUG_LEVEL > 0
00178   mtk::Tools::Prevent(west_bndy < mtk::kZero, __FILE__, __LINE__, __func__);
00179   mtk::Tools::Prevent(east_bndy < mtk::kZero, __FILE__, __LINE__, __func__);
00180   mtk::Tools::Prevent(east_bndy <= west_bndy, __FILE__, __LINE__, __func__);
00181   mtk::Tools::Prevent(num_cells_x < 0, __FILE__, __LINE__, __func__);
00182   mtk::Tools::Prevent(south_bndy < mtk::kZero, __FILE__, __LINE__, __func__);
00183   mtk::Tools::Prevent(north_bndy < mtk::kZero, __FILE__, __LINE__, __func__);
00184   mtk::Tools::Prevent(north_bndy <= south_bndy,
00185                       __FILE__, __LINE__, __func__);
00186   mtk::Tools::Prevent(num_cells_y < 0, __FILE__, __LINE__, __func__);
00187   #endif
00188
00189   nature_ = nature;
00190
00191   west_bndy_ = west_bndy;
00192   east_bndy_ = east_bndy;
00193   num_cells_x_ = num_cells_x;
00194
00195   south_bndy_ = south_bndy;
00196   north_bndy_ = north_bndy;
00197   num_cells_y_ = num_cells_y;
00198
00199   delta_x_ = (east_bndy_ - west_bndy_)/((mtk::Real) num_cells_x);
00200   delta_y_ = (north_bndy_ - south_bndy_)/((mtk::Real) num_cells_y);
00201 }
00202
00203 mtk::UniStgGrid2D::~UniStgGrid2D() {}
00204
00205 mtk::FieldNature mtk::UniStgGrid2D::nature() const {
00206
00207   return nature_;
00208 }
00209
00210 mtk::Real mtk::UniStgGrid2D::west_bndy() const {
00211
00212   return west_bndy_;
00213 }
00214
00215 mtk::Real mtk::UniStgGrid2D::east_bndy() const {
00216
00217   return east_bndy_;
00218 }
00219
00220 int mtk::UniStgGrid2D::num_cells_x() const {
00221
00222   return num_cells_x_;
00223 }
00224
00225 mtk::Real mtk::UniStgGrid2D::delta_x() const {
00226
00227   return delta_x_;
00228 }
00229
00230 const mtk::Real* mtk::UniStgGrid2D::discrete_domain_x() const
    {
00231
00232   return discrete_domain_x_.data();
00233 }
00234
00235 mtk::Real mtk::UniStgGrid2D::south_bndy() const {
```

```
00236
00237   return south_bndy_;
00238 }
00239
00240 mtk::Real mtk::UniStgGrid2D::north_bndy() const {
00241
00242   return north_bndy_;
00243 }
00244
00245 int mtk::UniStgGrid2D::num_cells_y() const {
00246
00247   return num_cells_y_;
00248 }
00249
00250 mtk::Real mtk::UniStgGrid2D::delta_y() const {
00251
00252   return delta_y_;
00253 }
00254
00255 bool mtk::UniStgGrid2D::Bound() const {
00256
00257   return discrete_field_.size() != 0;
00258 }
00259
00260 const mtk::Real* mtk::UniStgGrid2D::discrete_domain_y() const
      {
00261
00262   return discrete_domain_y_.data();
00263 }
00264
00265 mtk::Real* mtk::UniStgGrid2D::discrete_field() {
00266
00267   return discrete_field_.data();
00268 }
00269
00270 void mtk::UniStgGrid2D::BindScalarField(Real (*ScalarField)(
      Real xx, Real yy)) {
00271
00272   #if MTK_DEBUG_LEVEL > 0
00273   mtk::Tools::Prevent(nature_ != mtk::SCALAR, __FILE__, __LINE__, __func__);
00274   #endif
00275
00277
00278   discrete_domain_x_.reserve(num_cells_x_ + 2);
00279
00280   discrete_domain_x_.push_back(west_bndy_);
00281   #ifdef MTK_PRECISION_DOUBLE
00282   auto first_center = west_bndy_ + delta_x_/2.0;
00283   #else
00284   auto first_center = west_bndy_ + delta_x_/2.0f;
00285   #endif
00286   discrete_domain_x_.push_back(first_center);
00287   for (auto ii = 1; ii < num_cells_x_; ++ii) {
00288     discrete_domain_x_.push_back(first_center + ii*delta_x_);
00289   }
00290   discrete_domain_x_.push_back(east_bndy_);
00291
00293
00294   discrete_domain_y_.reserve(num_cells_y_ + 2);
00295
00296   discrete_domain_y_.push_back(south_bndy_);
00297   #ifdef MTK_PRECISION_DOUBLE
00298   first_center = south_bndy_ + delta_x_/2.0;
00299   #else
00300   first_center = south_bndy_ + delta_x_/2.0f;
00301   #endif
00302   discrete_domain_y_.push_back(first_center);
00303   for (auto ii = 1; ii < num_cells_y_; ++ii) {
00304     discrete_domain_y_.push_back(first_center + ii*delta_y_);
00305   }
00306   discrete_domain_y_.push_back(north_bndy_);
00307
00309
00310   discrete_field_.reserve((num_cells_x_ + 2)*(num_cells_y_ + 2));
00311
00312   for (int ii = 0; ii < num_cells_y_ + 2; ++ii) {
00313     for (int jj = 0; jj < num_cells_x_ + 2; ++jj) {
00314       #if MTK_DEBUG_LEVEL > 0
00315       std::cout << "Pushing value for x = " << discrete_domain_x_[jj] <<
00316         " y = " << discrete_domain_y_[ii] << std::endl;
00317       #endif
```

```
00318       discrete_field_.push_back(ScalarField(discrete_domain_x_[jj],
00319                                             discrete_domain_y_[ii]));
00320     }
00321   }
00322 }
00323
00324 void mtk::UniStgGrid2D::BindVectorFieldPComponent(
00325   mtk::Real (*VectorField)(mtk::Real xx, mtk::Real yy)) {
00326
00327   int mm{num_cells_x_};
00328   int nn{num_cells_y_};
00329
00330   int total{nn*(mm + 1) + mm*(nn + 1)};
00331
00332   #ifdef MTK_PRECISION_DOUBLE
00333   double half_delta_x{delta_x_/2.0};
00334   double half_delta_y{delta_y_/2.0};
00335   #else
00336   float half_delta_x{delta_x_/2.0f};
00337   float half_delta_y{delta_y_/2.0f};
00338   #endif
00339
00341
00342   // We need every data point of the discrete domain; i.e. we need all the
00343   // nodes and all the centers. There are mm centers for the x direction, and
00344   // nn centers for the y direction. Since there is one node per center, that
00345   // amounts to 2*mm. If we finally consider the final boundary node, it
00346   // amounts to a total of 2*mm + 1 for the x direction. Analogously, for the
00347   // y direction, this amounts to 2*nn + 1.
00348
00349   discrete_domain_x_.reserve(2*mm + 1);
00350
00351   discrete_domain_x_.push_back(west_bndy_);
00352   for (int ii = 1; ii < (2*mm + 1); ++ii) {
00353     discrete_domain_x_.push_back(west_bndy_ + ii*half_delta_x);
00354   }
00355
00357
00358   discrete_domain_y_.reserve(2*nn + 1);
00359
00360   discrete_domain_y_.push_back(south_bndy_);
00361   for (int ii = 1; ii < (2*nn + 1); ++ii) {
00362     discrete_domain_y_.push_back(south_bndy_ + ii*half_delta_y);
00363   }
00364
00366
00367   discrete_field_.reserve(total);
00368
00369   // For each y-center.
00370   for (int ii = 1; ii < 2*nn + 1; ii += 2) {
00371
00372     // Bind all of the x-nodes for this y-center.
00373     for (int jj = 0; jj < 2*mm + 1; jj += 2) {
00374       discrete_field_.push_back(VectorField(discrete_domain_x_[jj],
00375                                             discrete_domain_y_[ii]));
00376
00377       #if MTK_DEBUG_LEVEL > 0
00378       std::cout << "Binding v at x = " << discrete_domain_x_[jj] << " y = " <<
00379         discrete_domain_y_[ii] << " = " <<
00380         VectorField(discrete_domain_x_[jj],discrete_domain_y_[ii]) << std::endl;
00381       #endif
00382     }
00383   }
00384   #if MTK_DEBUG_LEVEL > 0
00385   std::cout << std::endl;
00386   #endif
00387 }
00388
00389 void mtk::UniStgGrid2D::BindVectorFieldQComponent(
00390   mtk::Real (*VectorField)(mtk::Real xx, mtk::Real yy)) {
00391
00392   int mm{num_cells_x_};
00393   int nn{num_cells_y_};
00394
00396
00397   // For each y-node.
00398   for (int ii = 0; ii < 2*nn + 1; ii += 2) {
00399
00400     // Bind all of the x-center for this y-node.
00401     for (int jj = 1; jj < 2*mm + 1; jj += 2) {
00402       discrete_field_.push_back(VectorField(discrete_domain_x_[jj],
```

```
00403                                           discrete_domain_y_[ii]));
00404
00405       #if MTK_DEBUG_LEVEL > 0
00406       std::cout << "Binding v at x = " << discrete_domain_x_[jj] << " y = " <<
00407         discrete_domain_y_[ii] << " = " <<
00408         VectorField(discrete_domain_x_[jj],discrete_domain_y_[ii]) << std::endl;
00409       #endif
00410     }
00411   }
00412   #if MTK_DEBUG_LEVEL > 0
00413   std::cout << std::endl;
00414   #endif
00415 }
00416
00417 void mtk::UniStgGrid2D::BindVectorField(
00418   Real (*VectorFieldPComponent)(Real xx,Real yy),
00419   Real (*VectorFieldQComponent)(Real xx,Real yy)) {
00420
00421   #if MTK_DEBUG_LEVEL > 0
00422   mtk::Tools::Prevent(nature_ != mtk::VECTOR, __FILE__, __LINE__, __func__);
00423   #endif
00424
00425   BindVectorFieldPComponent(VectorFieldPComponent);
00426   BindVectorFieldQComponent(VectorFieldQComponent);
00427 }
00428
00429 bool mtk::UniStgGrid2D::WriteToFile(std::string filename,
00430                                     std::string space_name_x,
00431                                     std::string space_name_y,
00432                                     std::string field_name) const {
00433
00434   std::ofstream output_dat_file;  // Output file.
00435
00436   output_dat_file.open(filename);
00437
00438   if (!output_dat_file.is_open()) {
00439     return false;
00440   }
00441
00442   if (nature_ == mtk::SCALAR) {
00443     output_dat_file << "# " << space_name_x <<  ' ' << space_name_y << ' ' <<
00444       field_name << std::endl;
00445
00446     int idx{};
00447     for (unsigned int ii = 0; ii < discrete_domain_y_.size(); ++ii) {
00448       for (unsigned int jj = 0; jj < discrete_domain_x_.size(); ++jj) {
00449         output_dat_file << discrete_domain_x_[jj] << ' ' <<
00450                            discrete_domain_y_[ii] << ' ' <<
00451                            discrete_field_[idx] <<
00452                           std::endl;
00453         idx++;
00454       }
00455       output_dat_file << std::endl;
00456     }
00457   } else {
00458     output_dat_file << "# " << space_name_x <<  ' ' << space_name_y << ' ' <<
00459       field_name << std::endl;
00460
00461     output_dat_file << "# Horizontal component:" << std::endl;
00462
00463     int mm{num_cells_x_};
00464     int nn{num_cells_y_};
00465
00467
00468     // For each y-center.
00469     int idx{};
00470     for (int ii = 1; ii < 2*nn + 1; ii += 2) {
00471       // Bind all of the x-nodes for this y-center.
00472       for (int jj = 0; jj < 2*mm + 1; jj += 2) {
00473
00474         output_dat_file << discrete_domain_x_[jj] << ' ' <<
00475           discrete_domain_y_[ii] << ' ' << discrete_field_[idx] << ' ' <<
00476           mtk::kZero << std::endl;
00477
00478         ++idx;
00479       }
00480     }
00481
00483     int p_offset{nn*(mm + 1) - 1};
00484     idx = 0;
00485     output_dat_file << "# Vertical component:" << std::endl;
```

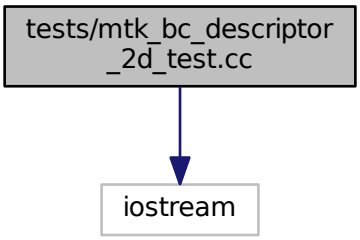
```
00486     // For each y-node.
00487     for (int ii = 0; ii < 2*nn + 1; ii += 2) {
00488       // Bind all of the x-center for this y-node.
00489       for (int jj = 1; jj < 2*mm + 1; jj += 2) {
00490
00491         output_dat_file << discrete_domain_x_[jj] << ' ' <<
00492           discrete_domain_y_[ii] << ' ' << mtk::kZero << ' ' <<
00493           discrete_field_[p_offset + idx] << std::endl;
00494
00495         ++idx;
00496       }
00497     }
00498   }
00499
00500   output_dat_file.close();
00501
00502   return true;
00503 }
```

17.87 tests/mtk_bc_descriptor_2d_test.cc File Reference

Test file for the mtk::BCDescriptor2D class.

`#include <iostream>`

Include dependency graph for mtk_bc_descriptor_2d_test.cc:



Functions

- int main ()

17.87.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_bc_descriptor_2d_test.cc.

17.87.2 Function Documentation

17.87.2.1 int main ( )

Definition at line 196 of file mtk_bc_descriptor_2d_test.cc.

17.88 mtk_bc_descriptor_2d_test.cc

```
00001
00008 /*
00009 Copyright (C) 2015, Computational Science Research Center, San Diego State
00010 University. All rights reserved.
00011
00012 Redistribution and use in source and binary forms, with or without modification,
00013 are permitted provided that the following conditions are met:
00014
00015 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00016 and a copy of the modified files should be reported once modifications are
00017 completed, unless these modifications are made through the project's GitHub
00018 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00019 should be developed and included in any deliverable.
00020
00021 2. Redistributions of source code must be done through direct
00022 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00023
00024 3. Redistributions in binary form must reproduce the above copyright notice,
00025 this list of conditions and the following disclaimer in the documentation and/or
00026 other materials provided with the distribution.
00027
00028 4. Usage of the binary form on proprietary applications shall require explicit
00029 prior written permission from the the copyright holders, and due credit should
00030 be given to the copyright holders.
00031
00032 5. Neither the name of the copyright holder nor the names of its contributors
00033 may be used to endorse or promote products derived from this software without
00034 specific prior written permission.
00035
00036 The copyright holders provide no reassurances that the source code provided does
00037 not infringe any patent, copyright, or any other intellectual property rights of
00038 third parties. The copyright holders disclaim any liability to any recipient for
00039 claims brought against recipient by any third party for infringement of that
00040 parties intellectual property rights.
00041
00042 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00043 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00044 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00045 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00046 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00047 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00048 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00049 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00050 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00051 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00052 */
00053
00054 #if __cplusplus == 201103L
00055
00056 #include <cmath>
00057 #include <ctime>
00058
00059 #include <iostream>
00060
00061 #include "mtk.h"
00062
00063 void TestDefaultConstructorGetters() {
00064
00065   mtk::Tools::BeginUnitTestNo(1);
00066
00067   mtk::BCDescriptor2D bcd;
00068
00069   bool assertion{true};
00070
00071   assertion = assertion && bcd.highest_order_diff_west() == -1;
00072   assertion = assertion && bcd.highest_order_diff_east() == -1;
00073   assertion = assertion && bcd.highest_order_diff_south() == -1;
00074   assertion = assertion && bcd.highest_order_diff_north() == -1;
00075
00076   mtk::Tools::EndUnitTestNo(1);
```

```
00077   mtk::Tools::Assert(assertion);
00078 }
00079
00080 mtk::Real cc(const mtk::Real &xx, const mtk::Real &yy) {
00081
00082   return mtk::kOne;
00083 }
00084
00085 void TestPushBackImposeOnLaplacianMatrix() {
00086
00087   mtk::Tools::BeginUnitTestNo(2);
00088
00089   mtk::BCDescriptor2D bcd;
00090
00091   bool assertion{true};
00092
00093   bcd.PushBackWestCoeff(cc);
00094   bcd.PushBackEastCoeff(cc);
00095   bcd.PushBackSouthCoeff(cc);
00096   bcd.PushBackNorthCoeff(cc);
00097
00098   assertion = assertion && bcd.highest_order_diff_west() == 0;
00099   assertion = assertion && bcd.highest_order_diff_east() == 0;
00100   assertion = assertion && bcd.highest_order_diff_south() == 0;
00101   assertion = assertion && bcd.highest_order_diff_north() == 0;
00102
00103   mtk::Real aa = 0.0;
00104   mtk::Real bb = 1.0;
00105   mtk::Real cc = 0.0;
00106   mtk::Real dd = 1.0;
00107
00108   int nn = 5;
00109   int mm = 5;
00110
00111   mtk::UniStgGrid2D llg(aa, bb, nn, cc, dd, mm);
00112
00113   mtk::Lap2D ll;
00114
00115   assertion = ll.ConstructLap2D(llg);
00116
00117   if (!assertion) {
00118     std::cerr << "Mimetic lap (2nd order) could not be built." << std::endl;
00119   }
00120
00121   mtk::DenseMatrix llm(ll.ReturnAsDenseMatrix());
00122
00123   assertion = assertion && (llm.num_rows() != 0);
00124
00125   bcd.ImposeOnLaplacianMatrix(llg, llm);
00126
00127   assertion = assertion && llm.WriteToFile("mtk_bc_descriptor_2d_test_02.dat");
00128
00129   mtk::Tools::EndUnitTestNo(2);
00130   mtk::Tools::Assert(assertion);
00131 }
00132
00133 mtk::Real ScalarField(mtk::Real xx, mtk::Real yy) {
00134
00135   mtk::Real aux{-(1.0/2.0)*xx*xx - (1.0/2.0)*yy*yy};
00136
00137   return xx*yy*exp(aux);
00138 }
00139
00140 mtk::Real HomogeneousDiricheletBC(mtk::Real xx, mtk::Real yy) {
00141
00142   return mtk::kZero;
00143 }
00144
00145 void TestImposeOnGrid() {
00146
00147   mtk::Tools::BeginUnitTestNo(3);
00148
00149   mtk::Real aa = 0.0;
00150   mtk::Real bb = 1.0;
00151   mtk::Real cc = 0.0;
00152   mtk::Real dd = 1.0;
00153
00154   int nn = 5;
00155   int mm = 5;
00156
00157   mtk::UniStgGrid2D gg(aa, bb, nn, cc, dd, mm);
```

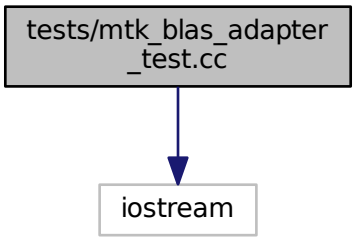
```
00158
00159   gg.BindScalarField(ScalarField);
00160
00161   mtk::BCDescriptor2D desc;
00162
00163   desc.set_west_condition(HomogeneousDiricheletBC);
00164   desc.set_east_condition(HomogeneousDiricheletBC);
00165   desc.set_south_condition(HomogeneousDiricheletBC);
00166   desc.set_north_condition(HomogeneousDiricheletBC);
00167
00168   desc.ImposeOnGrid(gg);
00169
00170   bool assertion{gg.WriteToFile("mtk_bc_descriptor_2d_test_03.dat",
00171                                 "x",
00172                                 "y",
00173                                 "u(x,y)")};
00174
00175   if(!assertion) {
00176     std::cerr << "Error writing to file." << std::endl;
00177   }
00178
00179   mtk::Tools::EndUnitTestNo(3);
00180   mtk::Tools::Assert(assertion);
00181 }
00182
00183 int main () {
00184
00185   std::cout << "Testing mtk::BCDescriptor2D class." << std::endl;
00186
00187   TestDefaultConstructorGetters();
00188   TestPushBackImposeOnLaplacianMatrix();
00189   TestImposeOnGrid();
00190 }
00191
00192 #else
00193 #include <iostream>
00194 using std::cout;
00195 using std::endl;
00196 int main () {
00197   cout << "This code HAS to be compiled with support for C++11." << endl;
00198   cout << "Exiting..." << endl;
00199 }
00200 #endif
```

17.89 tests/mtk_blas_adapter_test.cc File Reference

Test file for the mtk::BLASAdapter class.

`#include <iostream>`

Include dependency graph for mtk_blas_adapter_test.cc:



Functions

- int main ()

17.89.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_blas_adapter_test.cc.

17.89.2 Function Documentation

17.89.2.1 int main ( )

Definition at line 109 of file mtk_blas_adapter_test.cc.

17.90 mtk_blas_adapter_test.cc

```
00001
00008 /*
00009 Copyright (C) 2015, Computational Science Research Center, San Diego State
00010 University. All rights reserved.
00011
00012 Redistribution and use in source and binary forms, with or without modification,
00013 are permitted provided that the following conditions are met:
00014
00015 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00016 and a copy of the modified files should be reported once modifications are
00017 completed, unless these modifications are made through the project's GitHub
00018 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00019 should be developed and included in any deliverable.
00020
00021 2. Redistributions of source code must be done through direct
00022 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00023
00024 3. Redistributions in binary form must reproduce the above copyright notice,
00025 this list of conditions and the following disclaimer in the documentation and/or
00026 other materials provided with the distribution.
00027
00028 4. Usage of the binary form on proprietary applications shall require explicit
00029 prior written permission from the the copyright holders, and due credit should
00030 be given to the copyright holders.
00031
00032 5. Neither the name of the copyright holder nor the names of its contributors
00033 may be used to endorse or promote products derived from this software without
00034 specific prior written permission.
00035
00036 The copyright holders provide no reassurances that the source code provided does
00037 not infringe any patent, copyright, or any other intellectual property rights of
00038 third parties. The copyright holders disclaim any liability to any recipient for
00039 claims brought against recipient by any third party for infringement of that
00040 parties intellectual property rights.
00041
00042 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00043 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00044 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00045 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00046 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00047 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00048 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00049 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00050 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00051 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00052 */
00053
```

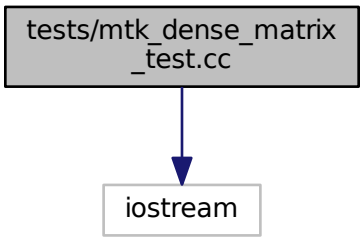
```
00054 #if __cplusplus == 201103L
00055
00056 #include <iostream>
00057
00058 #include "mtk.h"
00059
00060 void TestRealDenseMM() {
00061
00062   mtk::Tools::BeginUnitTestNo(1);
00063
00064   int rr = 2;
00065   int cc = 3;
00066
00067   mtk::DenseMatrix aa(rr,cc);
00068
00069   aa.SetValue(0,0,1.0);
00070   aa.SetValue(0,1,2.0);
00071   aa.SetValue(0,2,3.0);
00072   aa.SetValue(1,0,4.0);
00073   aa.SetValue(1,1,5.0);
00074   aa.SetValue(1,2,6.0);
00075
00076   mtk::DenseMatrix bb(cc,rr);
00077
00078   bb.SetValue(0,0,7.0);
00079   bb.SetValue(0,1,8.0);
00080   bb.SetValue(1,0,9.0);
00081   bb.SetValue(1,1,10.0);
00082   bb.SetValue(2,0,11.0);
00083   bb.SetValue(2,1,12.0);
00084
00085   mtk::DenseMatrix pp = mtk::BLASAdapter::RealDenseMM(aa,bb);
00086
00087   mtk::DenseMatrix ff(rr,rr);
00088
00089   ff.SetValue(0,0,58.0);
00090   ff.SetValue(0,1,64.00);
00091   ff.SetValue(1,0,139.0);
00092   ff.SetValue(1,1,154.0);
00093
00094   mtk::Tools::EndUnitTestNo(1);
00095   mtk::Tools::Assert(pp == ff);
00096 }
00097
00098 int main () {
00099
00100   std::cout << "Testing mtk::BLASAdapter class." << std::endl;
00101
00102   TestRealDenseMM();
00103 }
00104
00105 #else
00106 #include <iostream>
00107 using std::cout;
00108 using std::endl;
00109 int main () {
00110   cout << "This code HAS to be compiled with support for C++11." << endl;
00111   cout << "Exiting..." << endl;
00112 }
00113 #endif
```

17.91 tests/mtk_dense_matrix_test.cc File Reference

Test file for the mtk::DenseMatrix class.

```
#include <iostream>
```

Include dependency graph for mtk_dense_matrix_test.cc:



Functions

- int main ()

17.91.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_dense_matrix_test.cc.

17.91.2 Function Documentation

17.91.2.1 int main ( )

Definition at line 330 of file mtk_dense_matrix_test.cc.

17.92 mtk_dense_matrix_test.cc

```
00001
00008 /*
00009 Copyright (C) 2015, Computational Science Research Center, San Diego State
00010 University. All rights reserved.
00011
00012 Redistribution and use in source and binary forms, with or without modification,
00013 are permitted provided that the following conditions are met:
00014
00015 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00016 and a copy of the modified files should be reported once modifications are
00017 completed, unless these modifications are made through the project's GitHub
00018 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00019 should be developed and included in any deliverable.
00020
00021 2. Redistributions of source code must be done through direct
00022 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00023
```

```
00024 3. Redistributions in binary form must reproduce the above copyright notice,
00025 this list of conditions and the following disclaimer in the documentation and/or
00026 other materials provided with the distribution.
00027
00028 4. Usage of the binary form on proprietary applications shall require explicit
00029 prior written permission from the the copyright holders, and due credit should
00030 be given to the copyright holders.
00031
00032 5. Neither the name of the copyright holder nor the names of its contributors
00033 may be used to endorse or promote products derived from this software without
00034 specific prior written permission.
00035
00036 The copyright holders provide no reassurances that the source code provided does
00037 not infringe any patent, copyright, or any other intellectual property rights of
00038 third parties. The copyright holders disclaim any liability to any recipient for
00039 claims brought against recipient by any third party for infringement of that
00040 parties intellectual property rights.
00041
00042 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00043 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00044 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00045 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00046 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00047 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00048 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00049 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00050 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00051 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00052 */
00053
00054 #if __cplusplus == 201103L
00055
00056 #include <iostream>
00057 #include <ctime>
00058
00059 #include "mtk.h"
00060
00061 void TestDefaultConstructor() {
00062
00063   mtk::Tools::BeginUnitTestNo(1);
00064
00065   mtk::DenseMatrix m1;
00066
00067   mtk::Tools::EndUnitTestNo(1);
00068   mtk::Tools::Assert(m1.data() == nullptr);
00069 }
00070
00071 void TestConstructorWithNumRowsNumCols() {
00072
00073   mtk::Tools::BeginUnitTestNo(2);
00074
00075   int rr = 4;
00076   int cc = 7;
00077
00078   mtk::DenseMatrix m2(rr,cc);
00079
00080   mtk::Tools::EndUnitTestNo(2);
00081
00082   bool assertion =
00083     m2.data() != nullptr && m2.num_rows() == rr && m2.num_cols() == cc;
00084
00085   mtk::Tools::Assert(assertion);
00086 }
00087
00088 void TestConstructAsIdentity() {
00089
00090   mtk::Tools::BeginUnitTestNo(3);
00091
00092   int rank = 5;
00093   bool padded = true;
00094   bool transpose = false;
00095
00096   mtk::DenseMatrix m3(rank,padded,transpose);
00097
00098   mtk::DenseMatrix rr(rank + 2,rank);
00099
00100   for (int ii = 0; ii < rank; ++ii) {
00101     rr.SetValue(ii + 1, ii, mtk::kOne);
00102   }
00103
00104   mtk::Tools::EndUnitTestNo(3);
```

```
00105   mtk::Tools::Assert(m3 == rr);
00106 }
00107
00108   void TestConstructAsVandermonde() {
00109
00110   mtk::Tools::BeginUnitTestNo(4);
00111
00112   int rank = 5;
00113   bool padded = false;
00114   bool transpose = false;
00115
00116   mtk::DenseMatrix m4(rank,padded,transpose);
00117
00118   mtk::DenseMatrix rr(rank,rank);
00119
00120   for (int ii = 0; ii < rank; ++ii) {
00121     rr.SetValue(ii, ii, mtk::kOne);
00122   }
00123
00124   mtk::Tools::EndUnitTestNo(4);
00125   mtk::Tools::Assert(m4 == rr);
00126 }
00127
00128 void TestSetValueGetValue() {
00129
00130   mtk::Tools::BeginUnitTestNo(5);
00131
00132   int rr = 4;
00133   int cc = 7;
00134
00135   mtk::DenseMatrix m5(rr,cc);
00136
00137   for (auto ii = 0; ii < rr; ++ii) {
00138     for (auto jj = 0; jj < cc; ++jj) {
00139       m5.SetValue(ii,jj,(mtk::Real) ii + jj);
00140     }
00141   }
00142
00143   mtk::Real *vals = m5.data();
00144
00145   bool assertion{true};
00146
00147   for (auto ii = 0; ii < rr && assertion; ++ii) {
00148     for (auto jj = 0; jj < cc  && assertion; ++jj) {
00149       assertion = assertion && m5.GetValue(ii,jj) == vals[ii*cc + jj];
00150     }
00151   }
00152
00153   mtk::Tools::EndUnitTestNo(5);
00154   mtk::Tools::Assert(assertion);
00155 }
00156
00157 void TestConstructAsVandermondeTranspose() {
00158
00159   mtk::Tools::BeginUnitTestNo(6);
00160
00161   bool transpose = false;
00162   int generator_length = 3;
00163   int progression_length = 4;
00164
00165   mtk::Real generator[] = {-0.5, 0.5, 1.5};
00166
00167   mtk::DenseMatrix m6(generator,generator_length,progression_length,transpose);
00168
00169   transpose = true;
00170
00171   mtk::DenseMatrix m7(generator,generator_length,progression_length,transpose);
00172   mtk::DenseMatrix rr(progression_length, generator_length);
00173
00174   rr.SetValue(0, 0, 1.0);
00175   rr.SetValue(0, 1, 1.0);
00176   rr.SetValue(0, 2, 1.0);
00177
00178   rr.SetValue(1, 0, -0.5);
00179   rr.SetValue(1, 1, 0.5);
00180   rr.SetValue(1, 2, 1.5);
00181
00182   rr.SetValue(2, 0, 0.25);
00183   rr.SetValue(2, 1, 0.25);
00184   rr.SetValue(2, 2, 2.25);
00185
```

```
00186   rr.SetValue(3, 0, -0.125);
00187   rr.SetValue(3, 1, 0.125);
00188   rr.SetValue(3, 2, 3.375);
00189
00190   mtk::Tools::EndUnitTestNo(6);
00191   mtk::Tools::Assert(m7 == rr);
00192 }
00193
00194 void TestKron() {
00195
00196   mtk::Tools::BeginUnitTestNo(7);
00197
00198   bool padded = false;
00199   bool transpose = false;
00200   int lots_of_rows = 2;
00201   int lots_of_cols = 5;
00202   mtk::DenseMatrix m8(lots_of_rows,padded,transpose);
00203
00204   mtk::DenseMatrix m9(lots_of_rows,lots_of_cols);
00205
00206   for (auto ii = 0; ii < lots_of_rows; ++ii) {
00207     for (auto jj = 0; jj < lots_of_cols; ++jj) {
00208       m9.SetValue(ii,jj,(mtk::Real) ii*lots_of_cols + jj + 1);
00209     }
00210   }
00211
00212   mtk::DenseMatrix m10 = mtk::DenseMatrix::Kron(m8,m9);
00213
00214   mtk::DenseMatrix rr(lots_of_rows*lots_of_rows, lots_of_rows*lots_of_cols);
00215
00216   rr.SetValue(0,0,1.0);
00217   rr.SetValue(0,1,2.0);
00218   rr.SetValue(0,2,3.0);
00219   rr.SetValue(0,3,4.0);
00220   rr.SetValue(0,4,5.0);
00221   rr.SetValue(0,5,0.0);
00222   rr.SetValue(0,6,0.0);
00223   rr.SetValue(0,7,0.0);
00224   rr.SetValue(0,8,0.0);
00225   rr.SetValue(0,9,0.0);
00226
00227   rr.SetValue(1,0,6.0);
00228   rr.SetValue(1,1,7.0);
00229   rr.SetValue(1,2,8.0);
00230   rr.SetValue(1,3,9.0);
00231   rr.SetValue(1,4,10.0);
00232   rr.SetValue(1,5,0.0);
00233   rr.SetValue(1,6,0.0);
00234   rr.SetValue(1,7,0.0);
00235   rr.SetValue(1,8,0.0);
00236   rr.SetValue(1,9,0.0);
00237
00238   rr.SetValue(2,0,0.0);
00239   rr.SetValue(2,1,0.0);
00240   rr.SetValue(2,2,0.0);
00241   rr.SetValue(2,3,0.0);
00242   rr.SetValue(2,4,0.0);
00243   rr.SetValue(2,5,1.0);
00244   rr.SetValue(2,6,2.0);
00245   rr.SetValue(2,7,3.0);
00246   rr.SetValue(2,8,4.0);
00247   rr.SetValue(2,9,5.0);
00248
00249   rr.SetValue(3,0,0.0);
00250   rr.SetValue(3,1,0.0);
00251   rr.SetValue(3,2,0.0);
00252   rr.SetValue(3,3,0.0);
00253   rr.SetValue(3,4,0.0);
00254   rr.SetValue(3,5,6.0);
00255   rr.SetValue(3,6,7.0);
00256   rr.SetValue(3,7,8.0);
00257   rr.SetValue(3,8,9.0);
00258   rr.SetValue(3,9,10.0);
00259
00260   mtk::Tools::EndUnitTestNo(7);
00261   mtk::Tools::Assert(m10 == rr);
00262 }
00263
00264 void TestConstructWithNumRowsNumColsAssignmentOperator() {
00265
00266   mtk::Tools::BeginUnitTestNo(8);
```

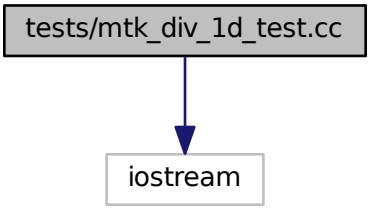
```
00267
00268   int lots_of_rows = 4;
00269   int lots_of_cols = 3;
00270   mtk::DenseMatrix m11(lots_of_rows,lots_of_cols);
00271
00272   for (auto ii = 0; ii < lots_of_rows; ++ii) {
00273     for (auto jj = 0; jj < lots_of_cols; ++jj) {
00274       m11.SetValue(ii,jj,(mtk::Real) ii*lots_of_cols + jj + 1);
00275     }
00276   }
00277
00278   m11.Transpose();
00279
00280   mtk::DenseMatrix m12;
00281
00282   m12 = m11;
00283
00284   mtk::Tools::EndUnitTestNo(8);
00285   mtk::Tools::Assert(m11 == m12);
00286 }
00287
00288 void TestConstructAsVandermondeTransposeAssignmentOperator() {
00289
00290   mtk::Tools::BeginUnitTestNo(9);
00291
00292   bool transpose = false;
00293   int gg_l = 3;
00294   int progression_length = 4;
00295   mtk::Real gg[] = {-0.5, 0.5, 1.5};
00296
00297   mtk::DenseMatrix m13(gg, gg_l ,progression_length, transpose);
00298
00299   mtk::DenseMatrix m14;
00300
00301   m14 = m13;
00302
00303   m13.Transpose();
00304
00305   m14 = m13;
00306
00307   mtk::Tools::EndUnitTestNo(9);
00308   mtk::Tools::Assert(m13 == m14);
00309 }
00310
00311 int main () {
00312
00313   std::cout << "Testing mtk::DenseMatrix class." << std::endl;
00314
00315   TestDefaultConstructor();
00316   TestConstructorWithNumRowsNumCols();
00317   TestConstructAsIdentity();
00318   TestConstructAsVandermonde();
00319   TestSetValueGetValue();
00320   TestConstructAsVandermondeTranspose();
00321   TestKron();
00322   TestConstructWithNumRowsNumColsAssignmentOperator();
00323   TestConstructAsVandermondeTransposeAssignmentOperator();
00324 }
00325
00326 #else
00327 #include <iostream>
00328 using std::cout;
00329 using std::endl;
00330 int main () {
00331   cout << "This code HAS to be compiled with support for C++11." << endl;
00332   cout << "Exiting..." << endl;
00333 }
00334 #endif
```

17.93 tests/mtk_div_1d_test.cc File Reference

Testing the mimetic 1D divergence, constructed with the CBS algorithm.

```
#include <iostream>
```

Include dependency graph for mtk_div_1d_test.cc:



Functions

- int main ()

17.93.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_div_1d_test.cc.

17.93.2 Function Documentation

17.93.2.1 int main ( )

Definition at line 288 of file mtk_div_1d_test.cc.

17.94 mtk_div_1d_test.cc

```
00001
00008 /*
00009 Copyright (C) 2015, Computational Science Research Center, San Diego State
00010 University. All rights reserved.
00011
00012 Redistribution and use in source and binary forms, with or without modification,
00013 are permitted provided that the following conditions are met:
00014
00015 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00016 and a copy of the modified files should be reported once modifications are
00017 completed, unless these modifications are made through the project's GitHub
00018 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00019 should be developed and included in any deliverable.
00020
00021 2. Redistributions of source code must be done through direct
00022 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00023
00024 3. Redistributions in binary form must reproduce the above copyright notice,
```

```
00025 this list of conditions and the following disclaimer in the documentation and/or
00026 other materials provided with the distribution.
00027
00028 4. Usage of the binary form on proprietary applications shall require explicit
00029 prior written permission from the the copyright holders, and due credit should
00030 be given to the copyright holders.
00031
00032 5. Neither the name of the copyright holder nor the names of its contributors
00033 may be used to endorse or promote products derived from this software without
00034 specific prior written permission.
00035
00036 The copyright holders provide no reassurances that the source code provided does
00037 not infringe any patent, copyright, or any other intellectual property rights of
00038 third parties. The copyright holders disclaim any liability to any recipient for
00039 claims brought against recipient by any third party for infringement of that
00040 parties intellectual property rights.
00041
00042 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00043 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00044 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00045 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00046 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00047 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00048 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00049 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00050 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00051 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00052 */
00053
00054 #if __cplusplus == 201103L
00055
00056 #include <iostream>
00057
00058 #include "mtk.h"
00059
00060 void TestDefaultConstructorFactoryMethodDefault() {
00061
00062   mtk::Tools::BeginUnitTestNo(1);
00063
00064   mtk::Div1D div2;
00065
00066   bool assertion = div2.ConstructDiv1D();
00067
00068   if (!assertion) {
00069     std::cerr << "Mimetic div (2nd order) could not be built." << std::endl;
00070   }
00071
00072   mtk::Tools::EndUnitTestNo(1);
00073   mtk::Tools::Assert(assertion);
00074 }
00075
00076 void TestDefaultConstructorFactoryMethodFourthOrder() {
00077
00078   mtk::Tools::BeginUnitTestNo(2);
00079
00080   mtk::Div1D div4;
00081
00082   bool assertion = div4.ConstructDiv1D(4);
00083
00084   if (!assertion) {
00085     std::cerr << "Mimetic div (4th order) could not be built." << std::endl;
00086   }
00087
00088   mtk::Tools::EndUnitTestNo(2);
00089   mtk::Tools::Assert(assertion);
00090 }
00091
00092 void TestDefaultConstructorFactoryMethodSixthOrder() {
00093
00094   mtk::Tools::BeginUnitTestNo(3);
00095
00096   mtk::Div1D div6;
00097
00098   bool assertion = div6.ConstructDiv1D(6);
00099
00100   if (!assertion) {
00101     std::cerr << "Mimetic div (6th order) could not be built." << std::endl;
00102   }
00103
00104   mtk::Tools::EndUnitTestNo(3);
00105   mtk::Tools::Assert(assertion);
```

```
00106 }
00107
00108 void TestDefaultConstructorFactoryMethodEightOrderDefThreshold() {
00109
00110   mtk::Tools::BeginUnitTestNo(4);
00111
00112   mtk::Div1D div8;
00113
00114   bool assertion = div8.ConstructDiv1D(8);
00115
00116   if (!assertion) {
00117     std::cerr << "Mimetic div (8th order) could not be built." << std::endl;
00118   }
00119
00120   mtk::Tools::EndUnitTestNo(4);
00121   mtk::Tools::Assert(assertion);
00122 }
00123
00124 void TestDefaultConstructorFactoryMethodTenthOrderDefThreshold() {
00125
00126   mtk::Tools::BeginUnitTestNo(5);
00127
00128   mtk::Div1D div10;
00129
00130   bool assertion = div10.ConstructDiv1D(10);
00131
00132   if (!assertion) {
00133     std::cerr << "Mimetic div (10th order) could not be built." << std::endl;
00134   }
00135
00136   mtk::Tools::EndUnitTestNo(5);
00137   mtk::Tools::Assert(assertion);
00138 }
00139
00140 void TestDefaultConstructorFactoryMethodTwelfthOrderDefThreshold() {
00141
00142   mtk::Tools::BeginUnitTestNo(6);
00143
00144   mtk::Div1D div12;
00145
00146   bool assertion = div12.ConstructDiv1D(12);
00147
00148   if (!assertion) {
00149     std::cerr << "Mimetic div (12th order) could not be built." << std::endl;
00150   }
00151
00152   mtk::Tools::EndUnitTestNo(6);
00153   mtk::Tools::Assert(assertion);
00154 }
00155
00156 void TestDefaultConstructorFactoryMethodFourteenthOrderDefThreshold() {
00157
00158   mtk::Tools::BeginUnitTestNo(7);
00159
00160   mtk::Div1D div14;
00161
00162   bool assertion = div14.ConstructDiv1D(14);
00163
00164   if (!assertion) {
00165     std::cerr << "Mimetic div (14th order) could not be built." << std::endl;
00166   }
00167
00168   mtk::Tools::EndUnitTestNo(7);
00169   mtk::Tools::Assert(assertion);
00170 }
00171
00172 void TestSecondOrderReturnAsDenseMatrixWithGrid() {
00173
00174   mtk::Tools::BeginUnitTestNo(8);
00175
00176   mtk::Div1D div2;
00177
00178   bool assertion = div2.ConstructDiv1D();
00179
00180   if (!assertion) {
00181     std::cerr << "Mimetic div (2nd order) could not be built." << std::endl;
00182   }
00183
00184   mtk::UniStgGrid1D grid(0.0, 1.0, 5);
00185
00186   mtk::DenseMatrix div2m(div2.ReturnAsDenseMatrix(grid));
```

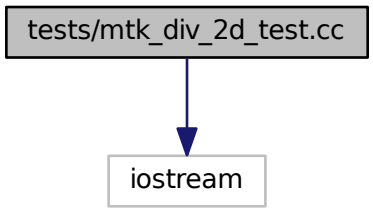
```
00187
00188   int rr{7};
00189   int cc{6};
00190
00191   mtk::DenseMatrix ref(rr, cc);
00192
00193   // Row 2.
00194   ref.SetValue(1,0,-5.0);
00195   ref.SetValue(1,1,5.0);
00196   ref.SetValue(1,2,0.0);
00197   ref.SetValue(1,3,0.0);
00198   ref.SetValue(1,4,0.0);
00199   ref.SetValue(1,5,0.0);
00200   ref.SetValue(1,6,0.0);
00201
00202   // Row 3.
00203   ref.SetValue(2,0,0.0);
00204   ref.SetValue(2,1,-5.0);
00205   ref.SetValue(2,2,5.0);
00206   ref.SetValue(2,3,0.0);
00207   ref.SetValue(2,4,0.0);
00208   ref.SetValue(2,5,0.0);
00209   ref.SetValue(2,6,0.0);
00210
00211   // Row 4.
00212   ref.SetValue(3,0,0.0);
00213   ref.SetValue(3,1,0.0);
00214   ref.SetValue(3,2,-5.0);
00215   ref.SetValue(3,3,5.0);
00216   ref.SetValue(3,4,0.0);
00217   ref.SetValue(3,5,0.0);
00218   ref.SetValue(3,6,0.0);
00219
00220   // Row 5.
00221   ref.SetValue(4,0,0.0);
00222   ref.SetValue(4,1,0.0);
00223   ref.SetValue(4,2,0.0);
00224   ref.SetValue(4,3,-5.0);
00225   ref.SetValue(4,4,5.0);
00226   ref.SetValue(4,5,0.0);
00227   ref.SetValue(4,6,0.0);
00228
00229   // Row 6.
00230   ref.SetValue(5,0,0.0);
00231   ref.SetValue(5,1,0.0);
00232   ref.SetValue(5,2,0.0);
00233   ref.SetValue(5,3,0.0);
00234   ref.SetValue(5,4,-5.0);
00235   ref.SetValue(5,5,5.0);
00236   ref.SetValue(5,6,0.0);
00237
00238   assertion = assertion && (div2m == ref);
00239
00240   mtk::Tools::EndUnitTestNo(8);
00241   mtk::Tools::Assert(assertion);
00242 }
00243
00244 void TestFourthOrderReturnAsDenseMatrixWithGrid() {
00245
00246   mtk::Tools::BeginUnitTestNo(9);
00247
00248   mtk::Div1D div4;
00249
00250   bool assertion = div4.ConstructDiv1D(4);
00251
00252   if (!assertion) {
00253     std::cerr << "Mimetic div (4th order) could not be built." << std::endl;
00254   }
00255
00256   std::cout << div4 << std::endl;
00257
00258   mtk::UniStgGrid1D grid(0.0, 1.0, 11);
00259
00260   std::cout << grid << std::endl;
00261
00262   mtk::DenseMatrix div4m(div4.ReturnAsDenseMatrix(grid));
00263
00264   std::cout << div4m << std::endl;
00265
00266   mtk::Tools::EndUnitTestNo(9);
00267 }
```

```
00268
00269 int main () {
00270
00271   std::cout << "Testing mtk::Div1D class." << std::endl;
00272
00273   TestDefaultConstructorFactoryMethodDefault();
00274   TestDefaultConstructorFactoryMethodFourthOrder();
00275   TestDefaultConstructorFactoryMethodSixthOrder();
00276   TestDefaultConstructorFactoryMethodEightOrderDefThreshold();
00277   TestDefaultConstructorFactoryMethodTenthOrderDefThreshold();
00278   TestDefaultConstructorFactoryMethodTwelfthOrderDefThreshold();
00279   TestDefaultConstructorFactoryMethodFourteenthOrderDefThreshold();
00280   TestSecondOrderReturnAsDenseMatrixWithGrid();
00281   TestFourthOrderReturnAsDenseMatrixWithGrid();
00282 }
00283
00284 #else
00285 #include <iostream>
00286 using std::cout;
00287 using std::endl;
00288 int main () {
00289   cout << "This code HAS to be compiled with support for C++11." << endl;
00290   cout << "Exiting..." << endl;
00291 }
00292 #endif
```

17.95 tests/mtk_div_2d_test.cc File Reference

Test file for the mtk::Div2D class.

`#include <iostream>`
Include dependency graph for mtk_div_2d_test.cc:



Functions

- int main ()

17.95.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_div_2d_test.cc.

17.95.2 Function Documentation

17.95.2.1 int main ( )

Definition at line 139 of file mtk_div_2d_test.cc.

17.96 mtk_div_2d_test.cc

```
00001
00008 /*
00009 Copyright (C) 2015, Computational Science Research Center, San Diego State
00010 University. All rights reserved.
00011
00012 Redistribution and use in source and binary forms, with or without modification,
00013 are permitted provided that the following conditions are met:
00014
00015 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00016 and a copy of the modified files should be reported once modifications are
00017 completed, unless these modifications are made through the project's GitHub
00018 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00019 should be developed and included in any deliverable.
00020
00021 2. Redistributions of source code must be done through direct
00022 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00023
00024 3. Redistributions in binary form must reproduce the above copyright notice,
00025 this list of conditions and the following disclaimer in the documentation and/or
00026 other materials provided with the distribution.
00027
00028 4. Usage of the binary form on proprietary applications shall require explicit
00029 prior written permission from the the copyright holders, and due credit should
00030 be given to the copyright holders.
00031
00032 5. Neither the name of the copyright holder nor the names of its contributors
00033 may be used to endorse or promote products derived from this software without
00034 specific prior written permission.
00035
00036 The copyright holders provide no reassurances that the source code provided does
00037 not infringe any patent, copyright, or any other intellectual property rights of
00038 third parties. The copyright holders disclaim any liability to any recipient for
00039 claims brought against recipient by any third party for infringement of that
00040 parties intellectual property rights.
00041
00042 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00043 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00044 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00045 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00046 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00047 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00048 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00049 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00050 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00051 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00052 */
00053
00054 #if __cplusplus == 201103L
00055
00056 #include <cmath>
00057 #include <ctime>
00058
00059 #include <iostream>
00060
00061 #include "mtk.h"
00062
00063 void TestDefaultConstructorFactory() {
00064
00065   mtk::Tools::BeginUnitTestNo(1);
00066
00067   mtk::Div2D dd;
00068
00069   mtk::Real aa = 0.0;
00070   mtk::Real bb = 1.0;
00071   mtk::Real cc = 0.0;
00072   mtk::Real ee = 1.0;
00073
```

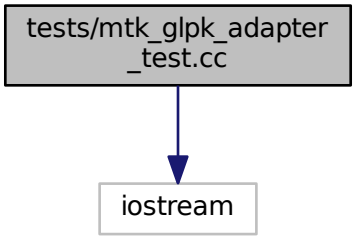
```
00074   int nn = 5;
00075   int mm = 5;
00076
00077   mtk::UniStgGrid2D ddg(aa, bb, nn, cc, ee, mm);
00078
00079   bool assertion = dd.ConstructDiv2D(ddg);
00080
00081   if (!assertion) {
00082     std::cerr << "Mimetic div (2nd order) could not be built." << std::endl;
00083   }
00084
00085   mtk::Tools::EndUnitTestNo(1);
00086   mtk::Tools::Assert(assertion);
00087 }
00088
00089 void TestReturnAsDenseMatrixWriteToFile() {
00090
00091   mtk::Tools::BeginUnitTestNo(2);
00092
00093   mtk::Div2D dd;
00094
00095   mtk::Real aa = 0.0;
00096   mtk::Real bb = 1.0;
00097   mtk::Real cc = 0.0;
00098   mtk::Real ee = 1.0;
00099
00100   int nn = 5;
00101   int mm = 5;
00102
00103   mtk::UniStgGrid2D ddg(aa, bb, nn, cc, ee, mm);
00104
00105   bool assertion = dd.ConstructDiv2D(ddg);
00106
00107   if (!assertion) {
00108     std::cerr << "Mimetic div (2nd order) could not be built." << std::endl;
00109   }
00110
00111   mtk::DenseMatrix ddm(dd.ReturnAsDenseMatrix());
00112
00113   assertion = assertion && (ddm.num_rows() != mtk::kZero);
00114
00115   std::cout << ddm << std::endl;
00116
00117   assertion = assertion && ddm.WriteToFile("mtk_div_2d_test_02.dat");
00118
00119   if(!assertion) {
00120     std::cerr << "Error writing to file." << std::endl;
00121   }
00122
00123   mtk::Tools::EndUnitTestNo(2);
00124   mtk::Tools::Assert(assertion);
00125 }
00126
00127 int main () {
00128
00129   std::cout << "Testing mtk::Div2D class." << std::endl;
00130
00131   TestDefaultConstructorFactory();
00132   TestReturnAsDenseMatrixWriteToFile();
00133 }
00134
00135 #else
00136 #include <iostream>
00137 using std::cout;
00138 using std::endl;
00139 int main () {
00140   cout << "This code HAS to be compiled with support for C++11." << endl;
00141   cout << "Exiting..." << endl;
00142 }
00143 #endif
```

17.97 tests/mtk_glpk_adapter_test.cc File Reference

Test file for the mtk::GLPKAdapter class.

```
#include <iostream>
```

Include dependency graph for mtk_glpk_adapter_test.cc:



Functions

- int main ()

17.97.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Todo Test the mtk::GLPKAdapter class.

Definition in file mtk_glpk_adapter_test.cc.

17.97.2 Function Documentation

17.97.2.1 int main ( )

Definition at line 81 of file mtk_glpk_adapter_test.cc.

17.98 mtk_glpk_adapter_test.cc

```
00001
00010 /*
00011 Copyright (C) 2015, Computational Science Research Center, San Diego State
00012 University. All rights reserved.
00013
00014 Redistribution and use in source and binary forms, with or without modification,
00015 are permitted provided that the following conditions are met:
00016
00017 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00018 and a copy of the modified files should be reported once modifications are
00019 completed, unless these modifications are made through the project's GitHub
00020 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00021 should be developed and included in any deliverable.
```

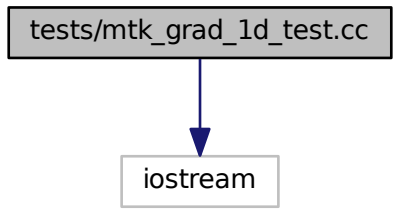
```
00022
00023 2. Redistributions of source code must be done through direct
00024 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00025
00026 3. Redistributions in binary form must reproduce the above copyright notice,
00027 this list of conditions and the following disclaimer in the documentation and/or
00028 other materials provided with the distribution.
00029
00030 4. Usage of the binary form on proprietary applications shall require explicit
00031 prior written permission from the the copyright holders, and due credit should
00032 be given to the copyright holders.
00033
00034 5. Neither the name of the copyright holder nor the names of its contributors
00035 may be used to endorse or promote products derived from this software without
00036 specific prior written permission.
00037
00038 The copyright holders provide no reassurances that the source code provided does
00039 not infringe any patent, copyright, or any other intellectual property rights of
00040 third parties. The copyright holders disclaim any liability to any recipient for
00041 claims brought against recipient by any third party for infringement of that
00042 parties intellectual property rights.
00043
00044 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00045 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00046 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00047 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00048 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00049 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00050 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00051 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00052 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00053 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00054 */
00055
00056 #if __cplusplus == 201103L
00057
00058 #include <iostream>
00059 #include <ctime>
00060
00061 #include "mtk.h"
00062
00063 void Test1() {
00064
00065   mtk::Tools::BeginUnitTestNo(1);
00066
00067   mtk::Tools::EndUnitTestNo(1);
00068 }
00069
00070 int main () {
00071
00072   std::cout << "Testing mtk::GLPKAdapter class." << std::endl;
00073
00074   Test1();
00075 }
00076
00077 #else
00078 #include <iostream>
00079 using std::cout;
00080 using std::endl;
00081 int main () {
00082   cout << "This code HAS to be compiled with support for C++11." << endl;
00083   cout << "Exiting..." << endl;
00084 }
00085 #endif
```

17.99 tests/mtk_grad_1d_test.cc File Reference

Testing the mimetic 1D gradient, constructed with the CBS algorithm.

```
#include <iostream>
```

Include dependency graph for mtk_grad_1d_test.cc:



Functions

- int main ()

17.99.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_grad_1d_test.cc.

17.99.2 Function Documentation

17.99.2.1 int main ( )

Definition at line 296 of file mtk_grad_1d_test.cc.

17.100 mtk_grad_1d_test.cc

```
00001
00008 /*
00009 Copyright (C) 2015, Computational Science Research Center, San Diego State
00010 University. All rights reserved.
00011
00012 Redistribution and use in source and binary forms, with or without modification,
00013 are permitted provided that the following conditions are met:
00014
00015 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00016 and a copy of the modified files should be reported once modifications are
00017 completed, unless these modifications are made through the project's GitHub
00018 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00019 should be developed and included in any deliverable.
00020
00021 2. Redistributions of source code must be done through direct
00022 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00023
00024 3. Redistributions in binary form must reproduce the above copyright notice,
```

```
00025 this list of conditions and the following disclaimer in the documentation and/or
00026 other materials provided with the distribution.
00027
00028 4. Usage of the binary form on proprietary applications shall require explicit
00029 prior written permission from the the copyright holders, and due credit should
00030 be given to the copyright holders.
00031
00032 5. Neither the name of the copyright holder nor the names of its contributors
00033 may be used to endorse or promote products derived from this software without
00034 specific prior written permission.
00035
00036 The copyright holders provide no reassurances that the source code provided does
00037 not infringe any patent, copyright, or any other intellectual property rights of
00038 third parties. The copyright holders disclaim any liability to any recipient for
00039 claims brought against recipient by any third party for infringement of that
00040 parties intellectual property rights.
00041
00042 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00043 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00044 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00045 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00046 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00047 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00048 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00049 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00050 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00051 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00052 */
00053
00054 #if __cplusplus == 201103L
00055
00056 #include <iostream>
00057
00058 #include "mtk.h"
00059
00060 void TestDefaultConstructorFactoryMethodDefault() {
00061
00062   mtk::Tools::BeginUnitTestNo(1);
00063
00064   mtk::Grad1D grad2;
00065
00066   bool assertion = grad2.ConstructGrad1D();
00067
00068   if (!assertion) {
00069     std::cerr << "Mimetic grad (2nd order) could not be built." << std::endl;
00070
00071   }
00072
00073   std::cout << grad2 << std::endl;
00074
00075   mtk::Tools::EndUnitTestNo(1);
00076   mtk::Tools::Assert(assertion);
00077 }
00078
00079 void TestDefaultConstructorFactoryMethodFourthOrder() {
00080
00081   mtk::Tools::BeginUnitTestNo(2);
00082
00083   mtk::Grad1D grad4;
00084
00085   bool assertion = grad4.ConstructGrad1D(4);
00086
00087   if (!assertion) {
00088     std::cerr << "Mimetic grad (4th order) could not be built." << std::endl;
00089   }
00090
00091   std::cout << grad4 << std::endl;
00092
00093   mtk::Tools::EndUnitTestNo(2);
00094   mtk::Tools::Assert(assertion);
00095 }
00096
00097 void TestDefaultConstructorFactoryMethodSixthOrder() {
00098
00099   mtk::Tools::BeginUnitTestNo(3);
00100
00101   mtk::Grad1D grad6;
00102
00103   bool assertion = grad6.ConstructGrad1D(6);
00104
00105   if (!assertion) {
```

```
00106     std::cerr << "Mimetic grad (6th order) could not be built." << std::endl;
00107   }
00108
00109   std::cout << grad6 << std::endl;
00110
00111   mtk::Tools::EndUnitTestNo(3);
00112   mtk::Tools::Assert(assertion);
00113 }
00114
00115 void TestDefaultConstructorFactoryMethodEightOrderDefThreshold() {
00116
00117   mtk::Tools::BeginUnitTestNo(4);
00118
00119   mtk::Grad1D grad8;
00120
00121   bool assertion = grad8.ConstructGrad1D(8);
00122
00123   if (!assertion) {
00124     std::cerr << "Mimetic grad (8th order) could not be built." << std::endl;
00125   }
00126
00127   std::cout << grad8 << std::endl;
00128
00129   mtk::Tools::EndUnitTestNo(4);
00130   mtk::Tools::Assert(assertion);
00131 }
00132
00133 void TestDefaultConstructorFactoryMethodTenthOrderDefThreshold() {
00134
00135   mtk::Tools::BeginUnitTestNo(5);
00136
00137   mtk::Grad1D grad10;
00138
00139   bool assertion = grad10.ConstructGrad1D(10);
00140
00141   if (!assertion) {
00142     std::cerr << "Mimetic grad (10th order) could not be built." << std::endl;
00143   }
00144
00145   std::cout << grad10 << std::endl;
00146
00147   mtk::Tools::EndUnitTestNo(5);
00148   mtk::Tools::Assert(assertion);
00149 }
00150
00151 void TestReturnAsDenseMatrixWithGrid() {
00152
00153   mtk::Tools::BeginUnitTestNo(6);
00154
00155   mtk::Grad1D grad2;
00156
00157   bool assertion = grad2.ConstructGrad1D();
00158
00159   if (!assertion) {
00160     std::cerr << "Mimetic grad (2nd order) could not be built." << std::endl;
00161   }
00162
00163   mtk::UniStgGrid1D grid(0.0, 1.0, 5);
00164
00165   mtk::DenseMatrix grad2m(grad2.ReturnAsDenseMatrix(grid));
00166
00167   int rr{6};
00168   int cc{7};
00169
00170   mtk::DenseMatrix ref(rr, cc);
00171
00172   // Row 1.
00173   ref.SetValue(0,0,-13.3333);
00174   ref.SetValue(0,1,15);
00175   ref.SetValue(0,2,-1.66667);
00176   ref.SetValue(0,3,0.0);
00177   ref.SetValue(0,4,0.0);
00178   ref.SetValue(0,5,0.0);
00179   ref.SetValue(0,6,0.0);
00180
00181   // Row 2.
00182   ref.SetValue(1,0,0.0);
00183   ref.SetValue(1,1,-5.0);
00184   ref.SetValue(1,2,5.0);
00185   ref.SetValue(1,3,0.0);
00186   ref.SetValue(1,4,0.0);
```

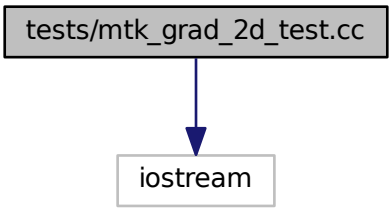
```
00187   ref.SetValue(1,5,0.0);
00188   ref.SetValue(1,6,0.0);
00189
00190   // Row 3.
00191   ref.SetValue(2,0,0.0);
00192   ref.SetValue(2,1,0.0);
00193   ref.SetValue(2,2,-5.0);
00194   ref.SetValue(2,3,5.0);
00195   ref.SetValue(2,4,0.0);
00196   ref.SetValue(2,5,0.0);
00197   ref.SetValue(2,6,0.0);
00198
00199   // Row 4.
00200   ref.SetValue(3,0,0.0);
00201   ref.SetValue(3,1,0.0);
00202   ref.SetValue(3,2,0.0);
00203   ref.SetValue(3,3,-5.0);
00204   ref.SetValue(3,4,5.0);
00205   ref.SetValue(3,5,0.0);
00206   ref.SetValue(3,6,0.0);
00207
00208   // Row 5.
00209   ref.SetValue(4,0,0.0);
00210   ref.SetValue(4,1,0.0);
00211   ref.SetValue(4,2,0.0);
00212   ref.SetValue(4,3,0.0);
00213   ref.SetValue(4,4,-5.0);
00214   ref.SetValue(4,5,5.0);
00215   ref.SetValue(4,6,0.0);
00216
00217   // Row 6.
00218   ref.SetValue(5,0,0.0);
00219   ref.SetValue(5,1,0.0);
00220   ref.SetValue(5,2,0.0);
00221   ref.SetValue(5,3,0.0);
00222   ref.SetValue(5,4,1.66667);
00223   ref.SetValue(5,5,-15.0);
00224   ref.SetValue(5,6,13.3333);
00225
00226   mtk::Tools::EndUnitTestNo(6);
00227   mtk::Tools::Assert(grad2m == ref);
00228 }
00229
00230 void TestReturnAsDimensionlessDenseMatrix() {
00231
00232   mtk::Tools::BeginUnitTestNo(7);
00233
00234   mtk::Grad1D grad4;
00235
00236   bool assertion = grad4.ConstructGrad1D(4);
00237
00238   if (!assertion) {
00239     std::cerr << "Mimetic grad (4th order) could not be built." << std::endl;
00240   }
00241
00242   mtk::DenseMatrix grad4m(grad4.ReturnAsDimensionlessDenseMatrix
      (10));
00243
00244   std::cout << grad4m << std::endl;
00245
00246   mtk::Tools::EndUnitTestNo(7);
00247   mtk::Tools::Assert(assertion);
00248 }
00249
00250 void TestWriteToFile() {
00251
00252   mtk::Tools::BeginUnitTestNo(8);
00253
00254   mtk::Grad1D grad2;
00255
00256   bool assertion = grad2.ConstructGrad1D();
00257
00258   if (!assertion) {
00259     std::cerr << "Mimetic grad (2nd order) could not be built." << std::endl;
00260   }
00261
00262   mtk::UniStgGrid1D grid(0.0, 1.0, 50);
00263
00264   mtk::DenseMatrix grad2m(grad2.ReturnAsDenseMatrix(grid));
00265
00266   std::cout << grad2m << std::endl;
```

```
00267
00268   assertion = assertion && grad2m.WriteToFile("mtk_grad_1d_test_08.dat");
00269
00270   if(!assertion) {
00271     std::cerr << "Error writing to file." << std::endl;
00272   }
00273
00274   mtk::Tools::EndUnitTestNo(8);
00275   mtk::Tools::Assert(assertion);
00276 }
00277
00278 int main () {
00279
00280   std::cout << "Testing mtk::Grad1D class." << std::endl;
00281
00282   TestDefaultConstructorFactoryMethodDefault();
00283   TestDefaultConstructorFactoryMethodFourthOrder();
00284   TestDefaultConstructorFactoryMethodSixthOrder();
00285   TestDefaultConstructorFactoryMethodEightOrderDefThreshold();
00286   TestDefaultConstructorFactoryMethodTenthOrderDefThreshold();
00287   TestReturnAsDenseMatrixWithGrid();
00288   TestReturnAsDimensionlessDenseMatrix();
00289   TestWriteToFile();
00290 }
00291
00292 #else
00293 #include <iostream>
00294 using std::cout;
00295 using std::endl;
00296 int main () {
00297   cout << "This code HAS to be compiled with support for C++11." << endl;
00298   cout << "Exiting..." << endl;
00299 }
00300 #endif
```

17.101 tests/mtk_grad_2d_test.cc File Reference

Test file for the mtk::Grad2D class.

`#include <iostream>`
Include dependency graph for mtk_grad_2d_test.cc:



Functions

- int main ()

17.101.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_grad_2d_test.cc.

17.101.2 Function Documentation

17.101.2.1 int main ( )

Definition at line 139 of file mtk_grad_2d_test.cc.

17.102 mtk_grad_2d_test.cc

```
00001
00008 /*
00009 Copyright (C) 2015, Computational Science Research Center, San Diego State
00010 University. All rights reserved.
00011
00012 Redistribution and use in source and binary forms, with or without modification,
00013 are permitted provided that the following conditions are met:
00014
00015 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00016 and a copy of the modified files should be reported once modifications are
00017 completed, unless these modifications are made through the project's GitHub
00018 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00019 should be developed and included in any deliverable.
00020
00021 2. Redistributions of source code must be done through direct
00022 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00023
00024 3. Redistributions in binary form must reproduce the above copyright notice,
00025 this list of conditions and the following disclaimer in the documentation and/or
00026 other materials provided with the distribution.
00027
00028 4. Usage of the binary form on proprietary applications shall require explicit
00029 prior written permission from the the copyright holders, and due credit should
00030 be given to the copyright holders.
00031
00032 5. Neither the name of the copyright holder nor the names of its contributors
00033 may be used to endorse or promote products derived from this software without
00034 specific prior written permission.
00035
00036 The copyright holders provide no reassurances that the source code provided does
00037 not infringe any patent, copyright, or any other intellectual property rights of
00038 third parties. The copyright holders disclaim any liability to any recipient for
00039 claims brought against recipient by any third party for infringement of that
00040 parties intellectual property rights.
00041
00042 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00043 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00044 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00045 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00046 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00047 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00048 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00049 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00050 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00051 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00052 */
00053
00054 #if __cplusplus == 201103L
00055
00056 #include <cmath>
00057 #include <ctime>
00058
00059 #include <iostream>
00060
00061 #include "mtk.h"
00062
00063 void TestDefaultConstructorFactory() {
```

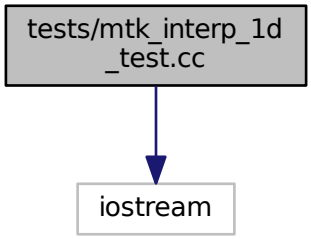
```
00064
00065   mtk::Tools::BeginUnitTestNo(1);
00066
00067   mtk::Grad2D gg;
00068
00069   mtk::Real aa = 0.0;
00070   mtk::Real bb = 1.0;
00071   mtk::Real cc = 0.0;
00072   mtk::Real dd = 1.0;
00073
00074   int nn = 5;
00075   int mm = 5;
00076
00077   mtk::UniStgGrid2D ggg(aa, bb, nn, cc, dd, mm, mtk::VECTOR);
00078
00079   bool assertion = gg.ConstructGrad2D(ggg);
00080
00081   if (!assertion) {
00082     std::cerr << "Mimetic grad (2nd order) could not be built." << std::endl;
00083   }
00084
00085   mtk::Tools::EndUnitTestNo(1);
00086   mtk::Tools::Assert(assertion);
00087 }
00088
00089 void TestReturnAsDenseMatrixWriteToFile() {
00090
00091   mtk::Tools::BeginUnitTestNo(2);
00092
00093   mtk::Grad2D gg;
00094
00095   mtk::Real aa = 0.0;
00096   mtk::Real bb = 1.0;
00097   mtk::Real cc = 0.0;
00098   mtk::Real dd = 1.0;
00099
00100   int nn = 5;
00101   int mm = 5;
00102
00103   mtk::UniStgGrid2D ggg(aa, bb, nn, cc, dd, mm, mtk::VECTOR);
00104
00105   bool assertion = gg.ConstructGrad2D(ggg);
00106
00107   if (!assertion) {
00108     std::cerr << "Mimetic grad (2nd order) could not be built." << std::endl;
00109   }
00110
00111   mtk::DenseMatrix ggm(gg.ReturnAsDenseMatrix());
00112
00113   assertion = assertion && (ggm.num_rows() != mtk::kZero);
00114
00115   std::cout << ggm << std::endl;
00116
00117   assertion = assertion && ggm.WriteToFile("mtk_grad_2d_test_02.dat");
00118
00119   if(!assertion) {
00120     std::cerr << "Error writing to file." << std::endl;
00121   }
00122
00123   mtk::Tools::EndUnitTestNo(2);
00124   mtk::Tools::Assert(assertion);
00125 }
00126
00127 int main () {
00128
00129   std::cout << "Testing mtk::Grad2D class." << std::endl;
00130
00131   TestDefaultConstructorFactory();
00132   TestReturnAsDenseMatrixWriteToFile();
00133 }
00134
00135 #else
00136 #include <iostream>
00137 using std::cout;
00138 using std::endl;
00139 int main () {
00140   cout << "This code HAS to be compiled with support for C++11." << endl;
00141   cout << "Exiting..." << endl;
00142 }
00143 #endif
```

17.103 tests/mtk_interp_1d_test.cc File Reference

Testing the 1D interpolation.

```
#include <iostream>
```

Include dependency graph for mtk_interp_1d_test.cc:



Functions

- int main ()

17.103.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu
: Johnny Corbino - jcorbino at mail dot sdsu dot edu

Definition in file mtk_interp_1d_test.cc.

17.103.2 Function Documentation

17.103.2.1 int main ( )

Definition at line 113 of file mtk_interp_1d_test.cc.

17.104 mtk_interp_1d_test.cc

```
00001
00010 /*
00011 Copyright (C) 2015, Computational Science Research Center, San Diego State
00012 University. All rights reserved.
00013
00014 Redistribution and use in source and binary forms, with or without modification,
00015 are permitted provided that the following conditions are met:
00016
00017 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00018 and a copy of the modified files should be reported once modifications are
```

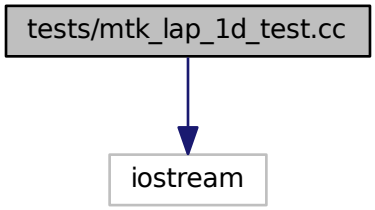
```
00019 completed, unless these modifications are made through the project's GitHub
00020 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00021 should be developed and included in any deliverable.
00022
00023 2. Redistributions of source code must be done through direct
00024 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00025
00026 3. Redistributions in binary form must reproduce the above copyright notice,
00027 this list of conditions and the following disclaimer in the documentation and/or
00028 other materials provided with the distribution.
00029
00030 4. Usage of the binary form on proprietary applications shall require explicit
00031 prior written permission from the the copyright holders, and due credit should
00032 be given to the copyright holders.
00033
00034 5. Neither the name of the copyright holder nor the names of its contributors
00035 may be used to endorse or promote products derived from this software without
00036 specific prior written permission.
00037
00038 The copyright holders provide no reassurances that the source code provided does
00039 not infringe any patent, copyright, or any other intellectual property rights of
00040 third parties. The copyright holders disclaim any liability to any recipient for
00041 claims brought against recipient by any third party for infringement of that
00042 parties intellectual property rights.
00043
00044 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00045 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00046 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00047 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00048 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00049 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00050 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00051 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00052 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00053 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00054 */
00055
00056 #if __cplusplus == 201103L
00057
00058 #include <iostream>
00059
00060 #include "mtk.h"
00061
00062 void TestDefaultConstructorFactoryMethodDefault() {
00063
00064   mtk::Tools::BeginUnitTestNo(1);
00065
00066   mtk::Interp1D inter;
00067
00068   bool assertion = inter.ConstructInterp1D();
00069
00070   if (!assertion) {
00071     std::cerr << "Mimetic interp could not be built." << std::endl;
00072   }
00073
00074   mtk::Tools::EndUnitTestNo(1);
00075   mtk::Tools::Assert(assertion);
00076 }
00077
00078 void TestReturnAsDenseMatrixWithGrid() {
00079
00080   mtk::Tools::BeginUnitTestNo(2);
00081
00082   mtk::Interp1D inter;
00083
00084   bool assertion = inter.ConstructInterp1D();
00085
00086   if (!assertion) {
00087     std::cerr << "Mimetic grad (2nd order) could not be built." << std::endl;
00088   }
00089
00090   mtk::UniStgGrid1D grid(0.0, 1.0, 5);
00091
00092   mtk::DenseMatrix interpm(inter.ReturnAsDenseMatrix(grid));
00093
00094   assertion =
00095     assertion && interpm.GetValue(0,0) == 1.0 && interpm.GetValue(5,6) == 1.0;
00096
00097   mtk::Tools::EndUnitTestNo(2);
00098   mtk::Tools::Assert(assertion);
00099 }
```

```
00100
00101 int main () {
00102
00103   std::cout << "Testing mtk::Interp1D class." << std::endl;
00104
00105   TestDefaultConstructorFactoryMethodDefault();
00106   TestReturnAsDenseMatrixWithGrid();
00107 }
00108
00109 #else
00110 #include <iostream>
00111 using std::cout;
00112 using std::endl;
00113 int main () {
00114   cout << "This code HAS to be compiled with support for C++11." << endl;
00115   cout << "Exiting..." << endl;
00116 }
00117 #endif
```

17.105 tests/mtk_lap_1d_test.cc File Reference

Testing the 1D Laplacian operator.

`#include <iostream>`
Include dependency graph for mtk_lap_1d_test.cc:



Functions

- int main ()

17.105.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu
: Johnny Corbino - jcorbino at mail dot sdsu dot edu

Definition in file mtk_lap_1d_test.cc.

17.105.2 Function Documentation

17.105.2.1 int main ( )

Definition at line 193 of file mtk_lap_1d_test.cc.

17.106 mtk_lap_1d_test.cc

```
00001
00010 /*
00011 Copyright (C) 2015, Computational Science Research Center, San Diego State
00012 University. All rights reserved.
00013
00014 Redistribution and use in source and binary forms, with or without modification,
00015 are permitted provided that the following conditions are met:
00016
00017 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00018 and a copy of the modified files should be reported once modifications are
00019 completed, unless these modifications are made through the project's GitHub
00020 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00021 should be developed and included in any deliverable.
00022
00023 2. Redistributions of source code must be done through direct
00024 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00025
00026 3. Redistributions in binary form must reproduce the above copyright notice,
00027 this list of conditions and the following disclaimer in the documentation and/or
00028 other materials provided with the distribution.
00029
00030 4. Usage of the binary form on proprietary applications shall require explicit
00031 prior written permission from the the copyright holders, and due credit should
00032 be given to the copyright holders.
00033
00034 5. Neither the name of the copyright holder nor the names of its contributors
00035 may be used to endorse or promote products derived from this software without
00036 specific prior written permission.
00037
00038 The copyright holders provide no reassurances that the source code provided does
00039 not infringe any patent, copyright, or any other intellectual property rights of
00040 third parties. The copyright holders disclaim any liability to any recipient for
00041 claims brought against recipient by any third party for infringement of that
00042 parties intellectual property rights.
00043
00044 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00045 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00046 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00047 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00048 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00049 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00050 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00051 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00052 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00053 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00054 */
00055
00056 #if __cplusplus == 201103L
00057
00058 #include <iostream>
00059
00060 #include "mtk.h"
00061
00062 void TestDefaultConstructorFactoryMethodDefault() {
00063
00064   mtk::Tools::BeginUnitTestNo(1);
00065
00066   mtk::Lap1D lap2;
00067
00068   bool assertion = lap2.ConstructLap1D();
00069
00070   if (!assertion) {
00071     std::cerr << "Mimetic lap (2nd order) could not be built." << std::endl;
00072   }
00073
00074   mtk::Tools::EndUnitTestNo(1);
00075   mtk::Tools::Assert(assertion);
00076 }
00077
00078 void TestDefaultConstructorFactoryMethodFourthOrder() {
```

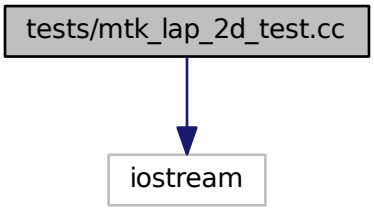
```
00079
00080   mtk::Tools::BeginUnitTestNo(2);
00081
00082   mtk::Lap1D lap4;
00083
00084   bool assertion = lap4.ConstructLap1D(4);
00085
00086   if (!assertion) {
00087     std::cerr << "Mimetic lap (4th order) could not be built." << std::endl;
00088   }
00089
00090   mtk::Tools::EndUnitTestNo(2);
00091   mtk::Tools::Assert(assertion);
00092 }
00093
00094 void TestDefaultConstructorFactoryMethodSixthOrder() {
00095
00096   mtk::Tools::BeginUnitTestNo(3);
00097
00098   mtk::Lap1D lap6;
00099
00100   bool assertion = lap6.ConstructLap1D(6);
00101
00102   if (!assertion) {
00103     std::cerr << "Mimetic lap (6th order) could not be built." << std::endl;
00104   }
00105
00106   mtk::Tools::EndUnitTestNo(3);
00107   mtk::Tools::Assert(assertion);
00108 }
00109
00110 void TestDefaultConstructorFactoryMethodEightOrderDefThreshold() {
00111
00112   mtk::Tools::BeginUnitTestNo(4);
00113
00114   mtk::Lap1D lap8;
00115
00116   bool assertion = lap8.ConstructLap1D(8);
00117
00118   if (!assertion) {
00119     std::cerr << "Mimetic lap (8th order) could not be built." << std::endl;
00120   }
00121
00122   mtk::Tools::EndUnitTestNo(4);
00123 }
00124
00125 void TestDefaultConstructorFactoryMethodTenthOrderDefThreshold() {
00126
00127   mtk::Tools::BeginUnitTestNo(5);
00128
00129   mtk::Lap1D lap10;
00130
00131   bool assertion = lap10.ConstructLap1D(10);
00132
00133   if (!assertion) {
00134     std::cerr << "Mimetic lap (10th order) could not be built." << std::endl;
00135   }
00136
00137   mtk::Tools::EndUnitTestNo(5);
00138 }
00139
00140 void TestDefaultConstructorFactoryMethodTwelfthOrderDefThreshold() {
00141
00142   mtk::Tools::BeginUnitTestNo(6);
00143
00144   mtk::Lap1D lap12;
00145
00146   bool assertion = lap12.ConstructLap1D(12);
00147
00148   if (!assertion) {
00149     std::cerr << "Mimetic lap (12th order) could not be built." << std::endl;
00150   }
00151
00152   mtk::Tools::EndUnitTestNo(6);
00153 }
00154
00155 void TestReturnAsDenseMatrix() {
00156
00157   mtk::Tools::BeginUnitTestNo(8);
00158
00159   mtk::Lap1D lap4;
```

```
00160
00161   bool assertion = lap4.ConstructLap1D(4);
00162
00163   if (!assertion) {
00164     std::cerr << "Mimetic lap (4th order) could not be built." << std::endl;
00165   }
00166
00167   mtk::UniStgGrid1D aux(0.0, 1.0, 11);
00168
00169   mtk::DenseMatrix lap4_m(lap4.ReturnAsDenseMatrix(aux));
00170
00171   assertion = assertion &&
00172       abs(lap4_m.GetValue(1, 0) - 385.133) < mtk::kDefaultTolerance &&
00173       abs(lap4_m.GetValue(11, 12) - 385.133) < mtk::kDefaultTolerance;
00174   mtk::Tools::EndUnitTestNo(8);
00175   mtk::Tools::Assert(assertion);
00176 }
00177
00178 int main () {
00179
00180   std::cout << "Testing MTK 1D Laplacian" << std::endl;
00181
00182   TestDefaultConstructorFactoryMethodDefault();
00183   TestDefaultConstructorFactoryMethodFourthOrder();
00184   TestDefaultConstructorFactoryMethodSixthOrder();
00185   TestDefaultConstructorFactoryMethodEightOrderDefThreshold();
00186   TestDefaultConstructorFactoryMethodTenthOrderDefThreshold();
00187   TestDefaultConstructorFactoryMethodTwelfthOrderDefThreshold();
00188   TestReturnAsDenseMatrix();
00189 }
00190
00191 #else
00192 #include <iostream>
00193 int main () {
00194   std::cout << "This code HAS to be compiled to support C++11." << std::endl;
00195   std::cout << "Exiting..." << std::endl;
00196 }
00197 #endif
```

17.107 tests/mtk_lap_2d_test.cc File Reference

Test file for the mtk::Lap2D class.

`#include <iostream>`
Include dependency graph for mtk_lap_2d_test.cc:



Functions

- int main ()

17.107.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_lap_2d_test.cc.

17.107.2 Function Documentation

17.107.2.1 int main ( )

Definition at line 139 of file mtk_lap_2d_test.cc.

17.108 mtk_lap_2d_test.cc

```
00001
00008 /*
00009 Copyright (C) 2015, Computational Science Research Center, San Diego State
00010 University. All rights reserved.
00011
00012 Redistribution and use in source and binary forms, with or without modification,
00013 are permitted provided that the following conditions are met:
00014
00015 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00016 and a copy of the modified files should be reported once modifications are
00017 completed, unless these modifications are made through the project's GitHub
00018 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00019 should be developed and included in any deliverable.
00020
00021 2. Redistributions of source code must be done through direct
00022 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00023
00024 3. Redistributions in binary form must reproduce the above copyright notice,
00025 this list of conditions and the following disclaimer in the documentation and/or
00026 other materials provided with the distribution.
00027
00028 4. Usage of the binary form on proprietary applications shall require explicit
00029 prior written permission from the the copyright holders, and due credit should
00030 be given to the copyright holders.
00031
00032 5. Neither the name of the copyright holder nor the names of its contributors
00033 may be used to endorse or promote products derived from this software without
00034 specific prior written permission.
00035
00036 The copyright holders provide no reassurances that the source code provided does
00037 not infringe any patent, copyright, or any other intellectual property rights of
00038 third parties. The copyright holders disclaim any liability to any recipient for
00039 claims brought against recipient by any third party for infringement of that
00040 parties intellectual property rights.
00041
00042 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00043 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00044 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00045 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00046 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00047 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00048 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00049 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00050 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00051 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00052 */
00053
00054 #if __cplusplus == 201103L
00055
00056 #include <cmath>
00057 #include <ctime>
00058
00059 #include <iostream>
00060
```

```
00061 #include "mtk.h"
00062
00063 void TestDefaultConstructorFactory() {
00064
00065   mtk::Tools::BeginUnitTestNo(1);
00066
00067   mtk::Lap2D ll;
00068
00069   mtk::Real aa = 0.0;
00070   mtk::Real bb = 1.0;
00071   mtk::Real cc = 0.0;
00072   mtk::Real dd = 1.0;
00073
00074   int nn = 5;
00075   int mm = 5;
00076
00077   mtk::UniStgGrid2D llg(aa, bb, nn, cc, dd, mm);
00078
00079   bool assertion = ll.ConstructLap2D(llg);
00080
00081   if (!assertion) {
00082     std::cerr << "Mimetic lap (2nd order) could not be built." << std::endl;
00083   }
00084
00085   mtk::Tools::EndUnitTestNo(1);
00086   mtk::Tools::Assert(assertion);
00087 }
00088
00089 void TestReturnAsDenseMatrixWriteToFile() {
00090
00091   mtk::Tools::BeginUnitTestNo(2);
00092
00093   mtk::Lap2D ll;
00094
00095   mtk::Real aa = 0.0;
00096   mtk::Real bb = 1.0;
00097   mtk::Real cc = 0.0;
00098   mtk::Real dd = 1.0;
00099
00100   int nn = 5;
00101   int mm = 5;
00102
00103   mtk::UniStgGrid2D llg(aa, bb, nn, cc, dd, mm);
00104
00105   bool assertion = ll.ConstructLap2D(llg);
00106
00107   if (!assertion) {
00108     std::cerr << "Mimetic lap (2nd order) could not be built." << std::endl;
00109   }
00110
00111   mtk::DenseMatrix llm(ll.ReturnAsDenseMatrix());
00112
00113   assertion = assertion && (llm.num_rows() != 0);
00114
00115   std::cout << llm << std::endl;
00116
00117   assertion = assertion && llm.WriteToFile("mtk_lap_2d_test_02.dat");
00118
00119   if(!assertion) {
00120     std::cerr << "Error writing to file." << std::endl;
00121   }
00122
00123   mtk::Tools::EndUnitTestNo(2);
00124   mtk::Tools::Assert(assertion);
00125 }
00126
00127 int main () {
00128
00129   std::cout << "Testing mtk::Lap2D class." << std::endl;
00130
00131   TestDefaultConstructorFactory();
00132   TestReturnAsDenseMatrixWriteToFile();
00133 }
00134
00135 #else
00136 #include <iostream>
00137 using std::cout;
00138 using std::endl;
00139 int main () {
00140   cout << "This code HAS to be compiled with support for C++11." << endl;
00141   cout << "Exiting..." << endl;
```

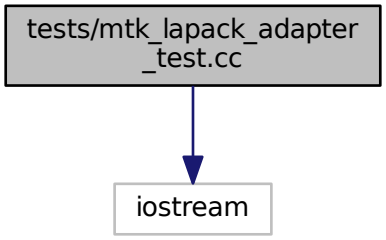
```
00142 }
00143 #endif
```

17.109 tests/mtk_lapack_adapter_test.cc File Reference

Test file for the mtk::LAPACKAdapter class.

```
#include <iostream>
```

Include dependency graph for mtk_lapack_adapter_test.cc:



Functions

- int main ()

17.109.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Todo Test the mtk::LAPACKAdapter class.

Definition in file mtk_lapack_adapter_test.cc.

17.109.2 Function Documentation

17.109.2.1 int main ( )

Definition at line 81 of file mtk_lapack_adapter_test.cc.

17.110 mtk_lapack_adapter_test.cc

```
00001
00010 /*
```

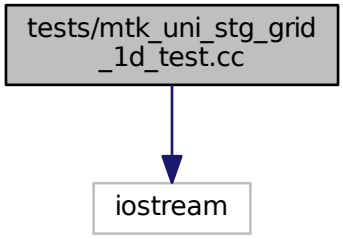
```
00011 Copyright (C) 2015, Computational Science Research Center, San Diego State
00012 University. All rights reserved.
00013
00014 Redistribution and use in source and binary forms, with or without modification,
00015 are permitted provided that the following conditions are met:
00016
00017 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00018 and a copy of the modified files should be reported once modifications are
00019 completed, unless these modifications are made through the project's GitHub
00020 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00021 should be developed and included in any deliverable.
00022
00023 2. Redistributions of source code must be done through direct
00024 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00025
00026 3. Redistributions in binary form must reproduce the above copyright notice,
00027 this list of conditions and the following disclaimer in the documentation and/or
00028 other materials provided with the distribution.
00029
00030 4. Usage of the binary form on proprietary applications shall require explicit
00031 prior written permission from the the copyright holders, and due credit should
00032 be given to the copyright holders.
00033
00034 5. Neither the name of the copyright holder nor the names of its contributors
00035 may be used to endorse or promote products derived from this software without
00036 specific prior written permission.
00037
00038 The copyright holders provide no reassurances that the source code provided does
00039 not infringe any patent, copyright, or any other intellectual property rights of
00040 third parties. The copyright holders disclaim any liability to any recipient for
00041 claims brought against recipient by any third party for infringement of that
00042 parties intellectual property rights.
00043
00044 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00045 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00046 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00047 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00048 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00049 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00050 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00051 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00052 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00053 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00054 */
00055
00056 #if __cplusplus == 201103L
00057
00058 #include <iostream>
00059 #include <ctime>
00060
00061 #include "mtk.h"
00062
00063 void Test1() {
00064
00065   mtk::Tools::BeginUnitTestNo(1);
00066
00067   mtk::Tools::EndUnitTestNo(1);
00068 }
00069
00070 int main () {
00071
00072   std::cout << "Testing mtk::LAPACKAdapter class." << std::endl;
00073
00074   Test1();
00075 }
00076
00077 #else
00078 #include <iostream>
00079 using std::cout;
00080 using std::endl;
00081 int main () {
00082   cout << "This code HAS to be compiled with support for C++11." << endl;
00083   cout << "Exiting..." << endl;
00084 }
00085 #endif
```

17.111 tests/mtk_uni_stg_grid_1d_test.cc File Reference

Test file for the mtk::UniStgGrid1D class.

```
#include <iostream>
```

Include dependency graph for mtk_uni_stg_grid_1d_test.cc:



Functions

- int main ()

17.111.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_uni_stg_grid_1d_test.cc.

17.111.2 Function Documentation

17.111.2.1 int main ( )

Definition at line 172 of file mtk_uni_stg_grid_1d_test.cc.

17.112 mtk_uni_stg_grid_1d_test.cc

```
00001
00008 /*
00009 Copyright (C) 2015, Computational Science Research Center, San Diego State
00010 University. All rights reserved.
00011
00012 Redistribution and use in source and binary forms, with or without modification,
00013 are permitted provided that the following conditions are met:
00014
00015 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00016 and a copy of the modified files should be reported once modifications are
00017 completed, unless these modifications are made through the project's GitHub
```

```
00018 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00019 should be developed and included in any deliverable.
00020
00021 2. Redistributions of source code must be done through direct
00022 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00023
00024 3. Redistributions in binary form must reproduce the above copyright notice,
00025 this list of conditions and the following disclaimer in the documentation and/or
00026 other materials provided with the distribution.
00027
00028 4. Usage of the binary form on proprietary applications shall require explicit
00029 prior written permission from the the copyright holders, and due credit should
00030 be given to the copyright holders.
00031
00032 5. Neither the name of the copyright holder nor the names of its contributors
00033 may be used to endorse or promote products derived from this software without
00034 specific prior written permission.
00035
00036 The copyright holders provide no reassurances that the source code provided does
00037 not infringe any patent, copyright, or any other intellectual property rights of
00038 third parties. The copyright holders disclaim any liability to any recipient for
00039 claims brought against recipient by any third party for infringement of that
00040 parties intellectual property rights.
00041
00042 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00043 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00044 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00045 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00046 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00047 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00048 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00049 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00050 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00051 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00052 */
00053
00054 #if __cplusplus == 201103L
00055
00056 #include <iostream>
00057 #include <ctime>
00058
00059 #include "mtk.h"
00060
00061 void TestDefaultConstructor() {
00062
00063   mtk::Tools::BeginUnitTestNo(1);
00064
00065   mtk::UniStgGrid1D gg;
00066
00067   mtk::Tools::EndUnitTestNo(1);
00068   mtk::Tools::Assert(gg.delta_x() == mtk::kZero);
00069 }
00070
00071 mtk::Real ScalarField(mtk::Real xx) {
00072
00073   return 2.0*xx;
00074 }
00075
00076 void TestConstructWithWestBndyEastBndyNumCellsOStreamOperatorBindScalarField() {
00077
00078   mtk::Tools::BeginUnitTestNo(2);
00079
00080   mtk::Real aa = 0.0;
00081   mtk::Real bb = 1.0;
00082
00083   int nn = 5;
00084
00085   mtk::UniStgGrid1D gg(aa, bb, nn);
00086
00087   gg.BindScalarField(ScalarField);
00088
00089   std::cout << gg << std::endl;
00090
00091   mtk::Tools::EndUnitTestNo(2);
00092   mtk::Tools::Assert(gg.delta_x() == 0.2 && gg.
      num_cells_x() == 5);
00093 }
00094
00095 void TestBindScalarFieldWriteToFile() {
00096
00097   mtk::Tools::BeginUnitTestNo(3);
```

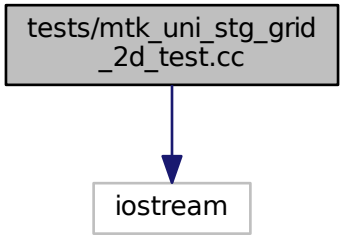
```
00098
00099   mtk::Real aa = 0.0;
00100   mtk::Real bb = 1.0;
00101
00102   int nn = 5;
00103
00104   mtk::UniStgGrid1D gg(aa, bb, nn);
00105
00106   bool assertion{true};
00107
00108   gg.BindScalarField(ScalarField);
00109
00110   assertion =
00111     assertion &&
00112     gg.discrete_field_u()[0] == 0.0 &&
00113     gg.discrete_field_u()[gg.num_cells_x() + 2 - 1] == 2.0;
00114
00115   if(!gg.WriteToFile("mtk_uni_stg_grid_1d_test_03.dat", "x", "u(x)")) {
00116     std::cerr << "Error writing to file." << std::endl;
00117     assertion = false;
00118   }
00119
00120   mtk::Tools::EndUnitTestNo(3);
00121   mtk::Tools::Assert(assertion);
00122 }
00123
00124 mtk::Real VectorFieldPComponent(mtk::Real xx) {
00125
00126   return xx*xx;
00127 }
00128
00129 void TestBindVectorField() {
00130
00131   mtk::Tools::BeginUnitTestNo(4);
00132
00133   mtk::Real aa = 0.0;
00134   mtk::Real bb = 1.0;
00135
00136   int nn = 20;
00137
00138   mtk::UniStgGrid1D gg(aa, bb, nn, mtk::VECTOR);
00139
00140   bool assertion{true};
00141
00142   gg.BindVectorField(VectorFieldPComponent);
00143
00144   assertion =
00145     assertion &&
00146     gg.discrete_field_u()[0] == 0.0 &&
00147     gg.discrete_field_u()[gg.num_cells_x() + 1 - 1] == 1.0;
00148
00149   if(!gg.WriteToFile("mtk_uni_stg_grid_1d_test_04.dat", "x", "v(x)")) {
00150     std::cerr << "Error writing to file." << std::endl;
00151     assertion = false;
00152   }
00153
00154   mtk::Tools::EndUnitTestNo(4);
00155   mtk::Tools::Assert(assertion);
00156 }
00157
00158 int main () {
00159
00160   std::cout << "Testing mtk::UniStgGrid1D class." << std::endl;
00161
00162   TestDefaultConstructor();
00163   TestConstructWithWestBndyEastBndyNumCellsOStreamOperatorBindScalarField();
00164   TestBindScalarFieldWriteToFile();
00165   TestBindVectorField();
00166 }
00167
00168 #else
00169 #include <iostream>
00170 using std::cout;
00171 using std::endl;
00172 int main () {
00173   cout << "This code HAS to be compiled with support for C++11." << endl;
00174   cout << "Exiting..." << endl;
00175 }
00176 #endif
```

17.113 tests/mtk_uni_stg_grid_2d_test.cc File Reference

Test file for the mtk::UniStgGrid2D class.

```
#include <iostream>
```

Include dependency graph for mtk_uni_stg_grid_2d_test.cc:



Functions

- int main ()

17.113.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_uni_stg_grid_2d_test.cc.

17.113.2 Function Documentation

17.113.2.1 int main ( )

Definition at line 202 of file mtk_uni_stg_grid_2d_test.cc.

17.114 mtk_uni_stg_grid_2d_test.cc

```
00001
00008 /*
00009 Copyright (C) 2015, Computational Science Research Center, San Diego State
00010 University. All rights reserved.
00011
00012 Redistribution and use in source and binary forms, with or without modification,
00013 are permitted provided that the following conditions are met:
00014
00015 1. Modifications to source code should be reported to: esanchez@mail.sdsu.edu
00016 and a copy of the modified files should be reported once modifications are
00017 completed, unless these modifications are made through the project's GitHub
```

```
00018 page: http://www.csrc.sdsu.edu/mtk. Documentation related to said modifications
00019 should be developed and included in any deliverable.
00020
00021 2. Redistributions of source code must be done through direct
00022 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00023
00024 3. Redistributions in binary form must reproduce the above copyright notice,
00025 this list of conditions and the following disclaimer in the documentation and/or
00026 other materials provided with the distribution.
00027
00028 4. Usage of the binary form on proprietary applications shall require explicit
00029 prior written permission from the the copyright holders, and due credit should
00030 be given to the copyright holders.
00031
00032 5. Neither the name of the copyright holder nor the names of its contributors
00033 may be used to endorse or promote products derived from this software without
00034 specific prior written permission.
00035
00036 The copyright holders provide no reassurances that the source code provided does
00037 not infringe any patent, copyright, or any other intellectual property rights of
00038 third parties. The copyright holders disclaim any liability to any recipient for
00039 claims brought against recipient by any third party for infringement of that
00040 parties intellectual property rights.
00041
00042 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00043 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00044 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00045 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
00046 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
00047 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
00048 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
00049 ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00050 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00051 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00052 */
00053
00054 #if __cplusplus == 201103L
00055
00056 #include <cmath>
00057 #include <ctime>
00058
00059 #include <iostream>
00060
00061 #include "mtk.h"
00062
00063 void TestDefaultConstructor() {
00064
00065   mtk::Tools::BeginUnitTestNo(1);
00066
00067   mtk::UniStgGrid2D gg;
00068
00069   mtk::Tools::EndUnitTestNo(1);
00070   mtk::Tools::Assert(gg.delta_x() == mtk::kZero && gg.
      delta_y() == mtk::kZero);
00071 }
00072
00073 void
00074 TestConstructWithWestEastNumCellsXSouthNorthBndysNumCellsYOStreamOperator() {
00075
00076   mtk::Tools::BeginUnitTestNo(2);
00077
00078   mtk::Real aa = 0.0;
00079   mtk::Real bb = 1.0;
00080   mtk::Real cc = 0.0;
00081   mtk::Real dd = 1.0;
00082
00083   int nn = 5;
00084   int mm = 7;
00085
00086   mtk::UniStgGrid2D gg(aa, bb, nn, cc, dd, mm);
00087
00088   std::cout << gg << std::endl;
00089
00090   mtk::Tools::EndUnitTestNo(2);
00091   mtk::Tools::Assert(gg.delta_x() == 0.2 &&
00092                      abs(gg.delta_y() - 0.142857) <
      mtk::kDefaultTolerance);
00093 }
00094
00095 void TestGetters() {
00096
```

```
00097   mtk::Tools::BeginUnitTestNo(3);
00098
00099   mtk::Real aa = 0.0;
00100   mtk::Real bb = 1.0;
00101   mtk::Real cc = 0.0;
00102   mtk::Real dd = 1.0;
00103
00104   int nn = 5;
00105   int mm = 7;
00106
00107   mtk::UniStgGrid2D gg(aa, bb, nn, cc, dd, mm);
00108
00109   bool assertion{true};
00110
00111   assertion = assertion && (gg.west_bndy() == aa);
00112   assertion = assertion && (gg.east_bndy() == bb);
00113   assertion = assertion && (gg.num_cells_x() == nn);
00114   assertion = assertion && (gg.south_bndy() == cc);
00115   assertion = assertion && (gg.north_bndy() == dd);
00116   assertion = assertion && (gg.num_cells_y() == mm);
00117
00118   mtk::Tools::EndUnitTestNo(3);
00119   mtk::Tools::Assert(assertion);
00120 }
00121
00122 mtk::Real ScalarField(mtk::Real xx, mtk::Real yy) {
00123
00124   mtk::Real aux{-(1.0/2.0)*xx*xx - (1.0/2.0)*yy*yy};
00125
00126   return xx*yy*exp(aux);
00127 }
00128
00129 void TestBindScalarFieldWriteToFile() {
00130
00131   mtk::Tools::BeginUnitTestNo(4);
00132
00133   mtk::Real aa = 0.0;
00134   mtk::Real bb = 1.0;
00135   mtk::Real cc = 0.0;
00136   mtk::Real dd = 1.0;
00137
00138   int nn = 5;
00139   int mm = 5;
00140
00141   mtk::UniStgGrid2D gg(aa, bb, nn, cc, dd, mm);
00142
00143   gg.BindScalarField(ScalarField);
00144
00145   if(!gg.WriteToFile("mtk_uni_stg_grid_2d_test_04.dat", "x", "y", "u(x,y)")) {
00146     std::cerr << "Error writing to file." << std::endl;
00147   }
00148
00149   mtk::Tools::EndUnitTestNo(4);
00150 }
00151
00152 mtk::Real VectorFieldPComponent(mtk::Real xx, mtk::Real yy) {
00153
00154   return xx + 0.01;
00155 }
00156
00157 mtk::Real VectorFieldQComponent(mtk::Real xx, mtk::Real yy) {
00158
00159   return yy + 0.01;
00160 }
00161
00162 void TestBindVectorField() {
00163
00164   mtk::Tools::BeginUnitTestNo(5);
00165
00166   mtk::Real aa = 0.0;
00167   mtk::Real bb = 1.0;
00168   mtk::Real cc = 0.0;
00169   mtk::Real dd = 1.0;
00170
00171   int nn = 5;
00172   int mm = 5;
00173
00174   mtk::UniStgGrid2D gg(aa, bb, nn, cc, dd, mm, mtk::VECTOR);
00175
00176   gg.BindVectorField(VectorFieldPComponent, VectorFieldQComponent);
00177
```

```
00178   std::cout << gg << std::endl;
00179
00180   if(!gg.WriteToFile("mtk_uni_stg_grid_2d_test_05.dat", "x", "y", "v(x,y)")) {
00181     std::cerr << "Error writing to file." << std::endl;
00182   }
00183
00184   mtk::Tools::EndUnitTestNo(5);
00185 }
00186
00187 int main () {
00188
00189   std::cout << "Testing mtk::UniStgGrid2D class." << std::endl;
00190
00191   TestDefaultConstructor();
00192   TestConstructWithWestEastNumCellsXSouthNorthBndysNumCellsYOStreamOperator();
00193   TestGetters();
00194   TestBindScalarFieldWriteToFile();
00195   TestBindVectorField();
00196 }
00197
00198 #else
00199 #include <iostream>
00200 using std::cout;
00201 using std::endl;
00202 int main () {
00203   cout << "This code HAS to be compiled with support for C++11." << endl;
00204   cout << "Exiting..." << endl;
00205 }
00206 #endif
```

Index