MTK: Mimetic Methods Toolkit

Generated by Doxygen 1.8.6

Sun Sep 13 2015 22:12:46

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 a set of classes for **mimetic quadratures**, **mimetic interpolation**, and **mimetic discretization methods** for the numerical solution of ordinary and partial differential equations.

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.

Chapter 3

Licensing and Modifications

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 discretization** methods for the numerical solution of ordinary and partial differential equations.

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

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

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

For OS X:

```
There are no dependences for OS X.
```

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:
- make: builds only the library and the examples.
- all: builds the library, the examples and the documentation.
- mtklib: builds the library, i.e. generates the archive files.
```

```
- test: generates the tests.
- example: generates the examples.
- 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 processor 1.73GHz 2048 KB of cache and stepping of 8
   gcc version 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5)
```

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::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 ()
: Review the definition of sparse matrices properties.

Member mtk::Matrix::IncreaseNumZero ()
: Review the definition of sparse matrices properties.

Member mtk::Tools::Prevent (const bool condition, const char ∗fname, int lineno, const char ∗fxname)
: Check if this is the best way of stalling execution.

Member mtk::Tools::test_number_
: Check usage of static methods and private members.

File mtk_dense_matrix.h
: Add sparse matrices support: BANDED and CRS.
: Contemplate manipulation of sparse metrics.
: Implement Kronecker product using the BLAS.

File mtk_div_1d.cc
: Overload ostream operator as in mtk::Lap1D.
: Implement creation of ■ w. mtk::BLASAdapter.

File mtk_glpk_adapter.cc
: Document better this file.

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_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 mechanism.

File mtk_uni_stg_grid_1d.h

Create overloaded binding routines that read data from files.

Chapter 9

Bug List

Member mtk::Matrix::set_num_null (int in)

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

Member mtk::Matrix::set_num_zero (int in)

-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::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 {1.e-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 157 of file mtk_roots.h.

14.1.3.2 mtk::kCriticalOrderAccuracyGrad {10}

Definition at line 166 of file mtk_roots.h.

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

Warning

Declared as double if MTK_PRECISION_DOUBLE is defined.

Definition at line 147 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 133 of file mtk_roots.h.

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

Definition at line 121 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 108 of file mtk_roots.h.

14.1.3.7 mtk::kZero {0.0f}

Warning

Declared as double if MTK_PRECISION_DOUBLE is defined.

Definition at line 107 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.

14.2.1 Detailed Description

Enumerations.

14.2.2 Enumeration Type Documentation

14.2.2.1 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.

14.2.2.2 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.

14.2.2.3 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.

14.6.1 Detailed Description

Uniform rectangular staggered grids.

14.7 Mimetic operators.

Mimetic operators.

Classes

- class mtk::Div1D

    Implements a 1D mimetic divergence operator.
- class mtk::Grad1D

    Implements a 1D mimetic gradient operator.
- class mtk::Lap1D

    Implements a 1D mimetic Laplacian operator.
- class mtk::Quad1D

    Implements a 1D mimetic quadrature.

14.7.1 Detailed Description

Mimetic operators.

Chapter 15

Namespace Documentation

15.1 mtk Namespace Reference

Mimetic Methods Toolkit namespace.

Classes

- class BCDesc1D
- 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 GLPKAdapter

  Adapter class for the GLPK API.
- class Grad1D

  Implements a 1D mimetic gradient operator.
- class Lap1D

  Implements a 1D 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.

Typedefs

- 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.

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::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)

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 kDefaultTolerance {1e-7f}

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

    Default order of accuracy for mimetic operators.
- const float kDefaultMimeticThreshold {1.e-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::UniStgGrid1D & *in*)

1. Print spatial coordinates.

Definition at line 68 of file mtk_uni_stg_grid_1d.cc.

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

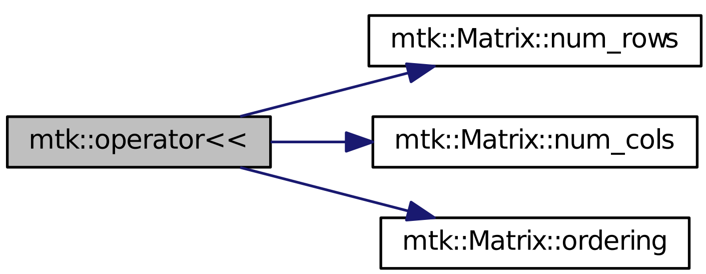
1. Print order of accuracy.
2. Print approximating coefficients for the interior.
3. No weights, thus print the mimetic boundary coefficients.

Definition at line 73 of file mtk_lap_1d.cc.

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

Definition at line 75 of file mtk_dense_matrix.cc.

Here is the call graph for this function:



15.1.1.4 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.

15.1.1.5 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.

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

Here is the caller graph for this function:

15.1.1.7 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.

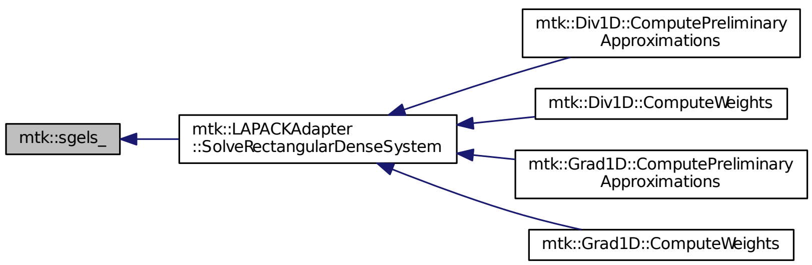
See Also

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

Parameters

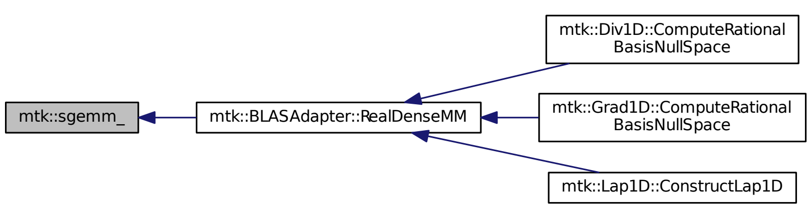
in	*trans*	Am I giving the transpose of the matrix?
in	*m*	The number of rows of the matrix a. m >= 0.
in	*n*	The number of columns of the matrix a. n >= 0.
in	*nrhs*	The number of right-hand sides.
in,out	*a*	On entry, the m-by-n matrix a.
in	*lda*	The leading dimension of a. lda >= max(1,m).
in,out	*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:



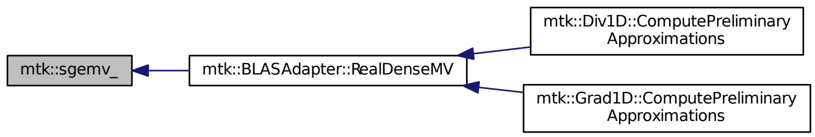
15.1.1.8 void mtk::sgemm_ (char ∗ *transa,* char ∗ *transb,* int ∗ *m,* int ∗ *n,* int ∗ *k,* double ∗ *alpha,* double ∗ *a,* int ∗ *lda,* double ∗ *b,* aamm int ∗ *ldb,* double ∗ *beta,* double ∗ *c,* int ∗ *ldc*)

Here is the caller graph for this function:



15.1.1.9 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:



15.1.1.10 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

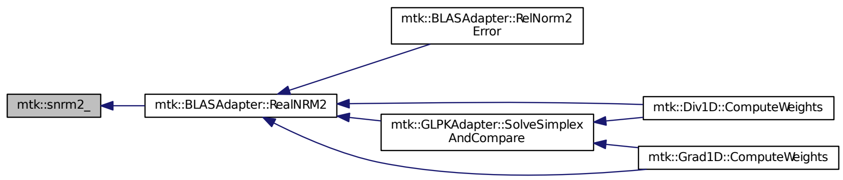
Parameters

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.

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

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

Here is the caller graph for this function:



15.1.1.13 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.

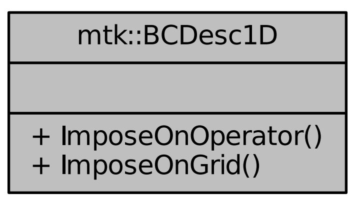
Chapter 16

Class Documentation

16.1 mtk::BCDesc1D Class Reference

```
#include <mtk_bc_desc_1d.h>
```

Collaboration diagram for mtk::BCDesc1D:



Static Public Member Functions

- static void ImposeOnOperator (DenseMatrix &matrix, const std::vector< Real > &west, const std::vector< Real > &east)
- static void ImposeOnGrid (UniStgGrid1D &grid, const Real &omega, const Real &epsilon)

16.1.1 Detailed Description

Definition at line 9 of file mtk_bc_desc_1d.h.

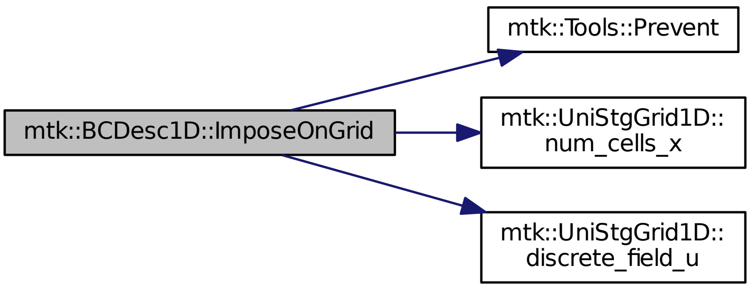
16.1.2 Member Function Documentation

16.1.2.1 void mtk::BCDesc1D::ImposeOnGrid (**mtk::UniStgGrid1D** & *grid*, const **Real** & *omega*, const **Real** & *epsilon*) [static]

1. Assign the west condition.
2. Assign the east condition.

Definition at line 30 of file mtk_bc_desc_1d.cc.

Here is the call graph for this function:

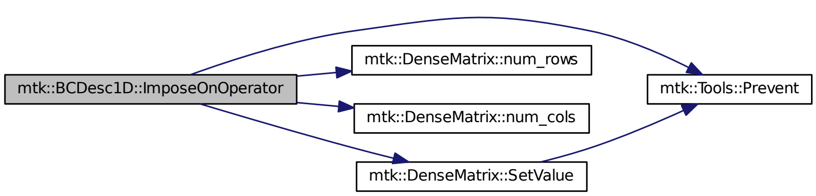


16.1.2.2 void mtk::BCDesc1D::ImposeOnOperator (**mtk::DenseMatrix** & *matrix*, const std::vector< **Real** > & *west*, const std::vector< **Real** > & *east*) [static]

1. Assign the west array.
2. Assign the east array.

Definition at line 5 of file mtk_bc_desc_1d.cc.

Here is the call graph for this function:



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

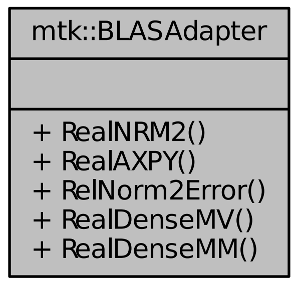
- include/mtk_bc_desc_1d.h
- src/mtk_bc_desc_1d.cc

16.2 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.

16.2.1 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.

16.2.2 Member Function Documentation

16.2.2.1 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

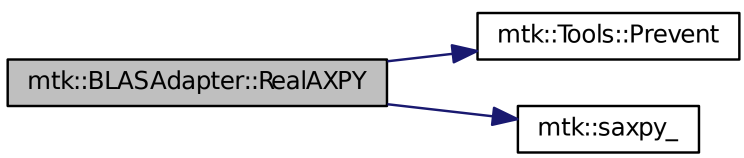
in	*alpha*	Scalar of the first array.
in	*xx*	First array.
in	*yy*	Second array.
in	*in_length*	Lengths of the given arrays.

Returns

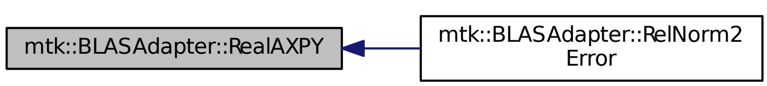
Norm-2 of the given array.

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:



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

Performs:

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

Parameters

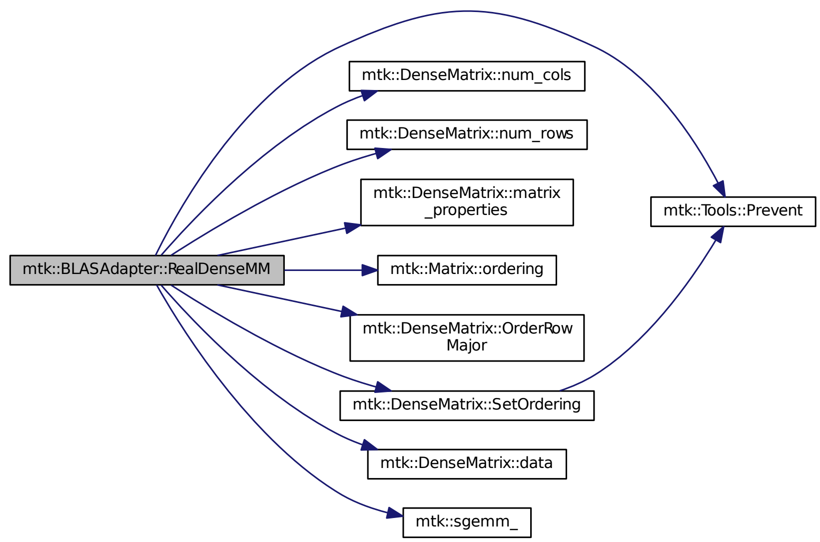
in	*aa*	First matrix.
in	*bb*	Second matrix.

See Also

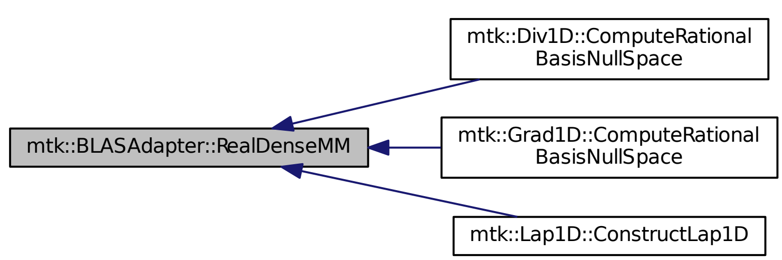
http://ejspeiro.github.io/Netlib-and-CPP/

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:



16.2.2.3 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

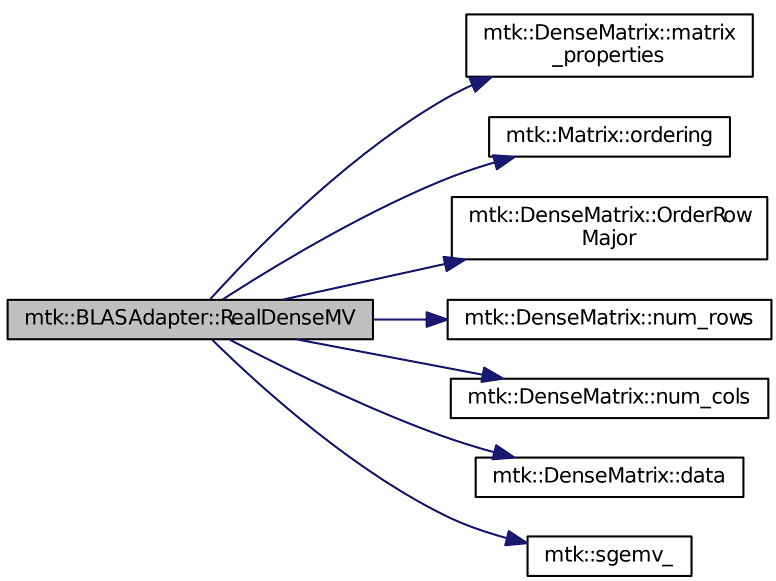
in	*alpha*	First scalar.
in	*aa*	Given matrix.
in	*xx*	First vector.
in	*beta*	Second scalar.
in,out	*yy*	Second vector (output).

See Also

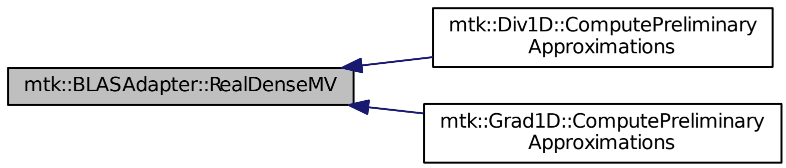
http://ejspeiro.github.io/Netlib-and-CPP/

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:



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

Parameters

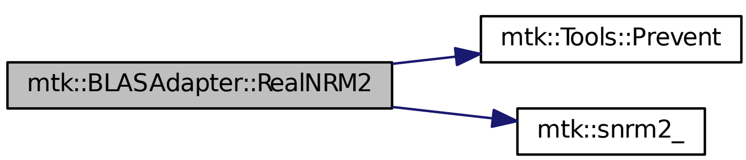
in	*in*	Input array.
in	*in_length*	Length of the array.

Returns

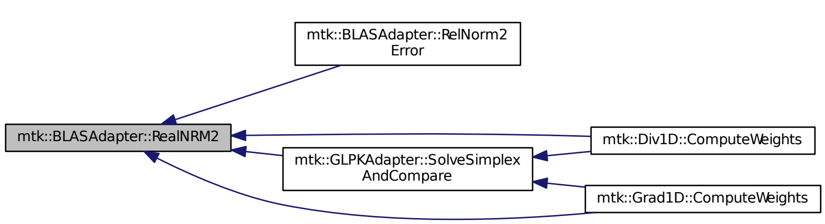
Norm-2 of the given array.

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:



16.2.2.5 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

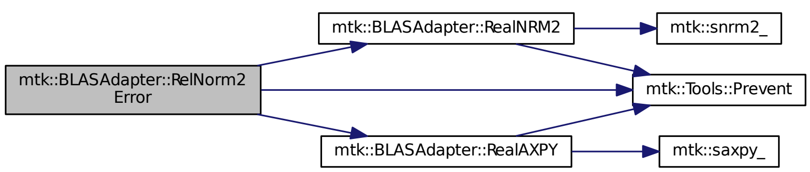
in	*known*	Array containing the computed solution.
in	*computed*	Array containing the known solution (ref. solution).

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:

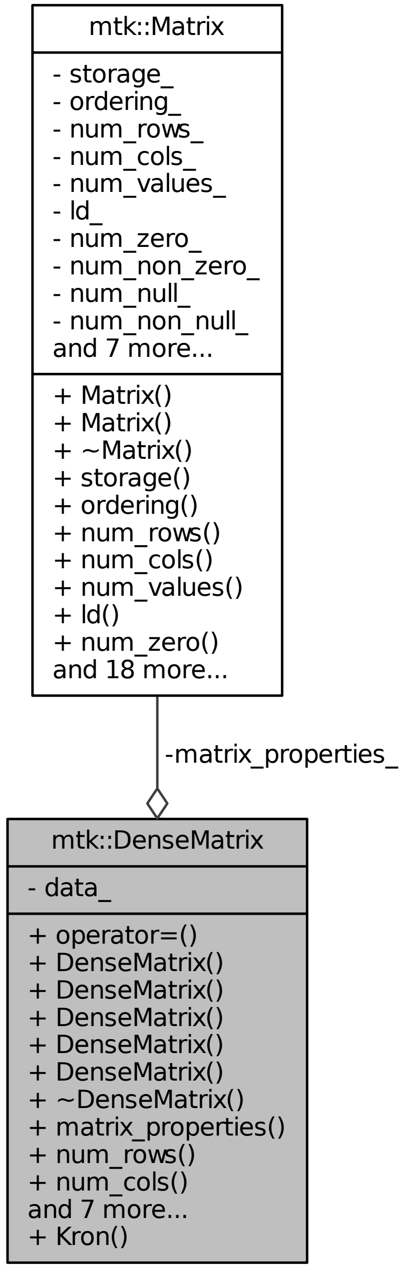
- include/mtk_blas_adapter.h
- src/mtk_blas_adapter.cc

16.3 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.

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

  Construct a dense Vandermonde matrix.

- ∼DenseMatrix ()

  Destructor.

- Matrix matrix_properties () const

  Provides access to the matrix data.

- int num_rows () const

  Gets the number of rows.

- int num_cols () const

  Gets the number of columns.

- Real * data () const

  Provides access to the matrix value array.

- void SetOrdering (mtk::MatrixOrdering oo)

  Sets the ordering of the matrix.

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

  Gets a value on the given coordinates.

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

  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.

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 contiguouos position in memory.

Friends

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

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

16.3.1 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.

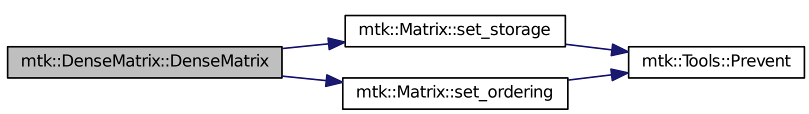
Definition at line 98 of file mtk_dense_matrix.h.

16.3.2 Constructor & Destructor Documentation

16.3.2.1 mtk::DenseMatrix::DenseMatrix ( )

Definition at line 138 of file mtk_dense_matrix.cc.

Here is the call graph for this function:



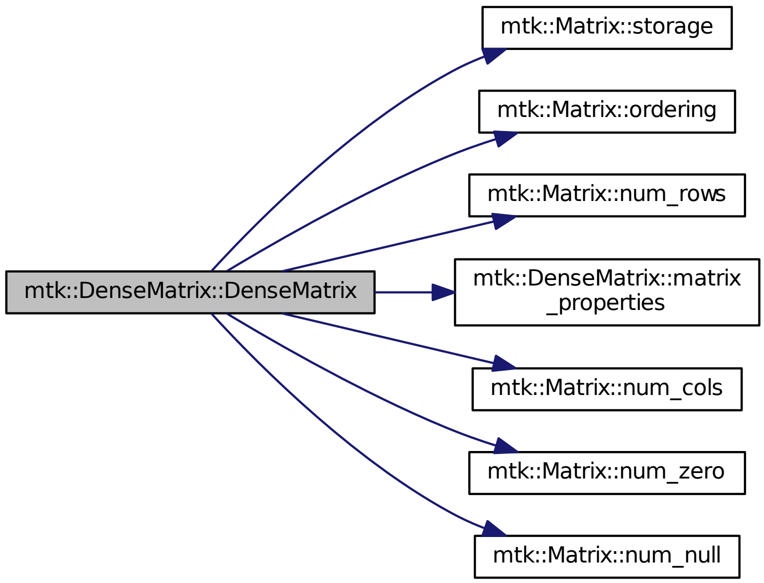
16.3.2.2 mtk::DenseMatrix::DenseMatrix (const DenseMatrix & *in*)

Parameters

in	*in*	Given matrix.

Definition at line 144 of file mtk_dense_matrix.cc.

Here is the call graph for this function:



16.3.2.3 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 177 of file mtk_dense_matrix.cc.

Here is the call graph for this function:

16.3.2.4 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 199 of file mtk_dense_matrix.cc.

Here is the call graph for this function:

16.3.2.5 mtk::DenseMatrix::DenseMatrix (const Real $*$ *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 237 of file mtk_dense_matrix.cc.

Here is the call graph for this function:

16.3.2.6 mtk::DenseMatrix::∼DenseMatrix ( )

Definition at line 285 of file mtk_dense_matrix.cc.

16.3.3 Member Function Documentation

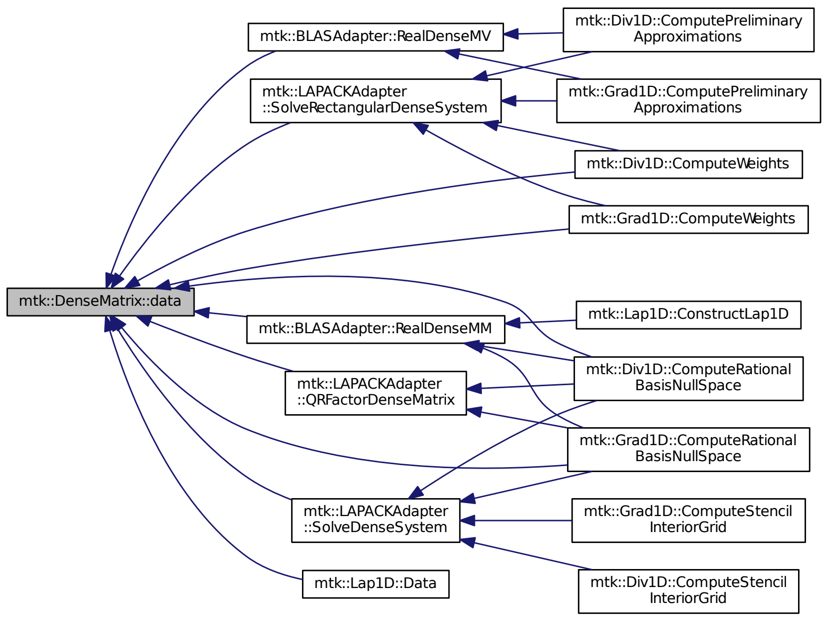
16.3.3.1 mtk::Real ∗ mtk::DenseMatrix::data ( ) const

Returns

Pointer to an array of mtk::Real.

Definition at line 316 of file mtk_dense_matrix.cc.

Here is the caller graph for this function:



16.3.3.2 mtk::Real mtk::DenseMatrix::GetValue (const int & *row_coord*, const int & *col_coord*) const

Parameters

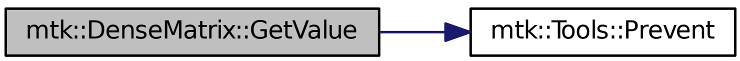
in	*row_coord*	Row coordinate.
in	*col_coord*	Column coordinate.

Returns

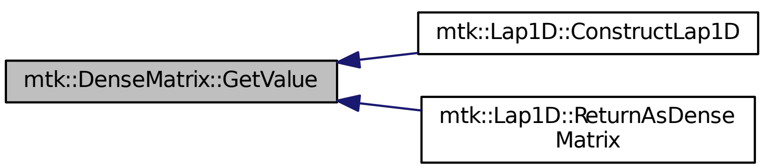
The required value at the specified coordinates.

Definition at line 321 of file mtk_dense_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



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

Parameters

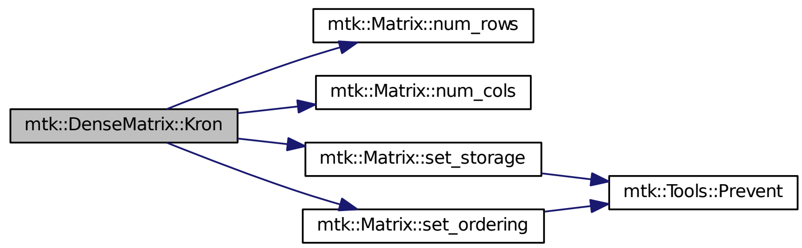
in	*aa*	First matrix.
in	*bb*	Second matrix.

Exceptions

std::bad_alloc	

Definition at line 463 of file mtk_dense_matrix.cc.

Here is the call graph for this function:



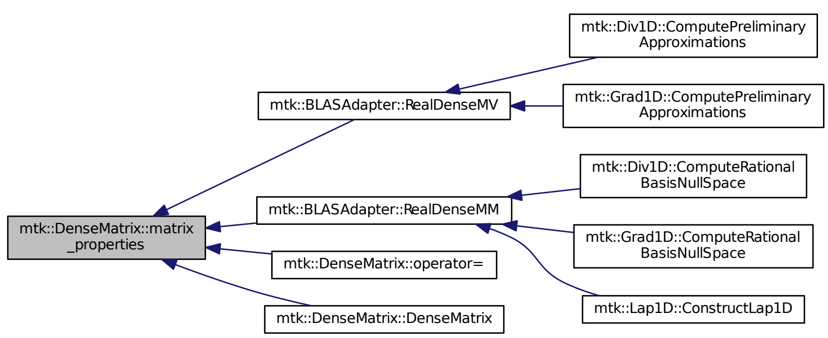
16.3.3.4 mtk::Matrix mtk::DenseMatrix::matrix_properties ( ) const

Returns

Pointer to a Matrix.

Definition at line 291 of file mtk_dense_matrix.cc.

Here is the caller graph for this function:



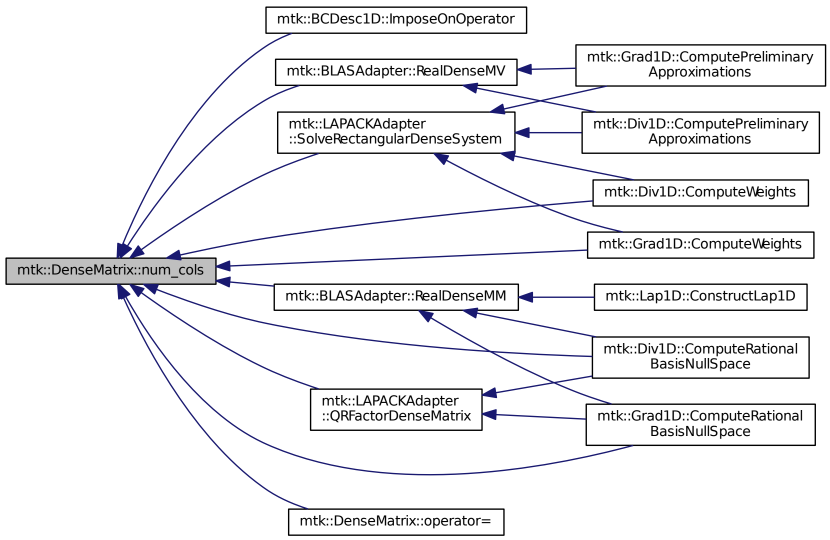
16.3.3.5 int mtk::DenseMatrix::num_cols ( ) const

Returns

Number of columns of the matrix.

Definition at line 311 of file mtk_dense_matrix.cc.

Here is the caller graph for this function:



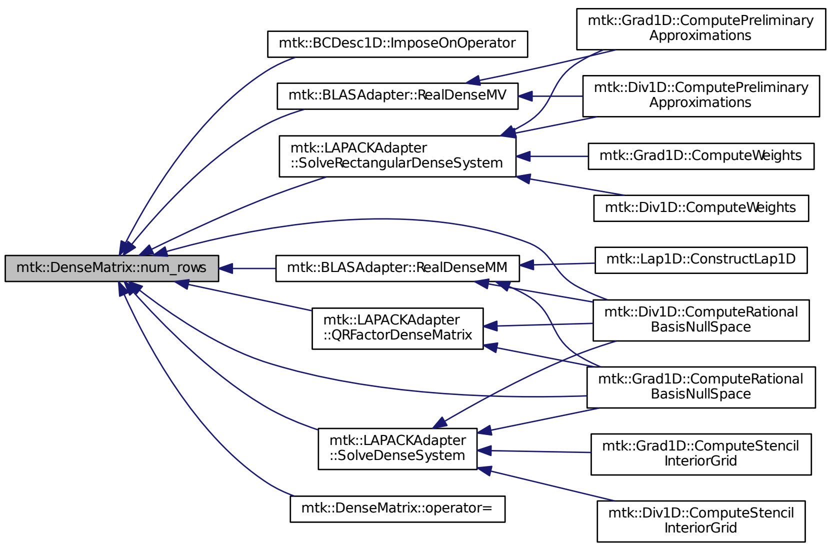
16.3.3.6 int mtk::DenseMatrix::num_rows ( ) const

Returns

Number of rows of the matrix.

Definition at line 306 of file mtk_dense_matrix.cc.

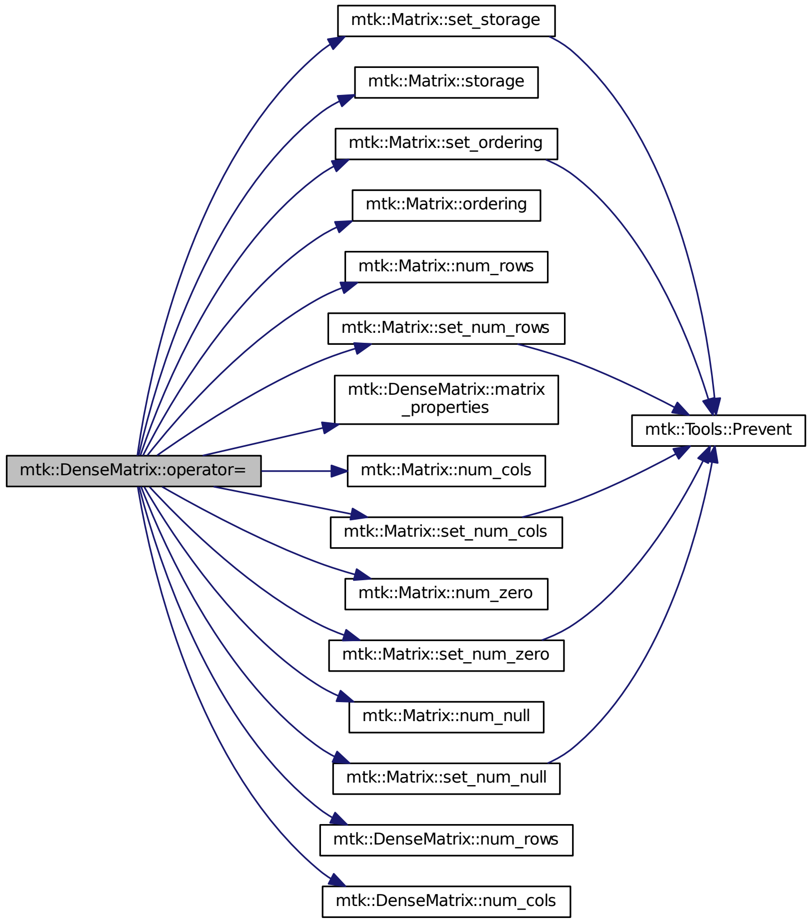
Here is the caller graph for this function:



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

Definition at line 97 of file mtk_dense_matrix.cc.

Here is the call graph for this function:

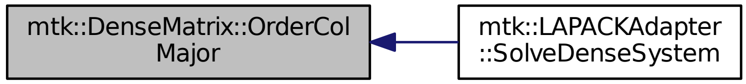


16.3.3.8 void mtk::DenseMatrix::OrderColMajor ( )

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

Definition at line 424 of file mtk_dense_matrix.cc.

Here is the caller graph for this function:

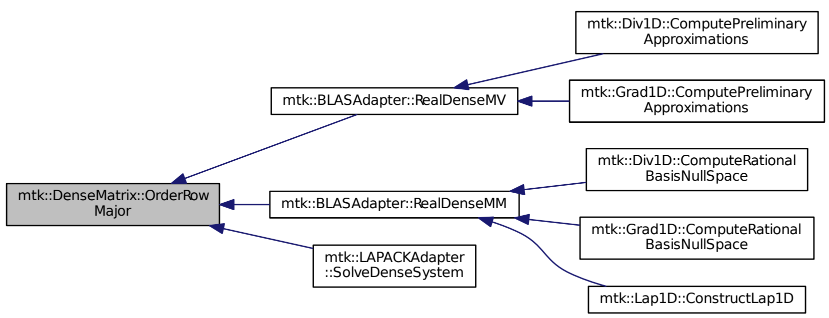


16.3.3.9 void mtk::DenseMatrix::OrderRowMajor ( )

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

Definition at line 383 of file mtk_dense_matrix.cc.

Here is the caller graph for this function:



16.3.3.10 void mtk::DenseMatrix::SetOrdering (mtk::MatrixOrdering *oo*)

Parameters

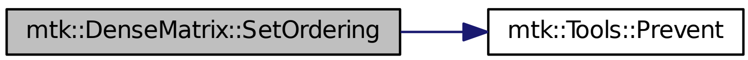
in	*oo*	Ordering.

Returns

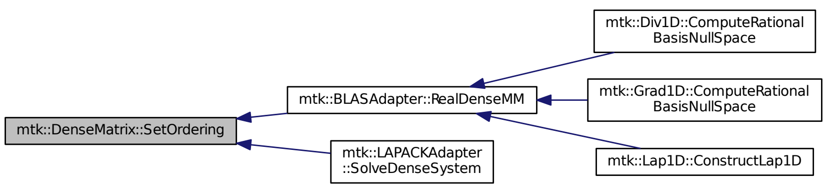
The required value at the specified coordinates.

Definition at line 296 of file mtk_dense_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



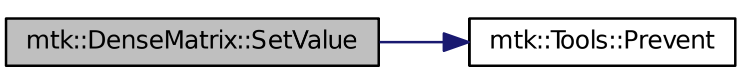
16.3.3.11 void mtk::DenseMatrix::SetValue (const int & *row_coord*, const int & *col_coord*, const Real & *val*)

Parameters

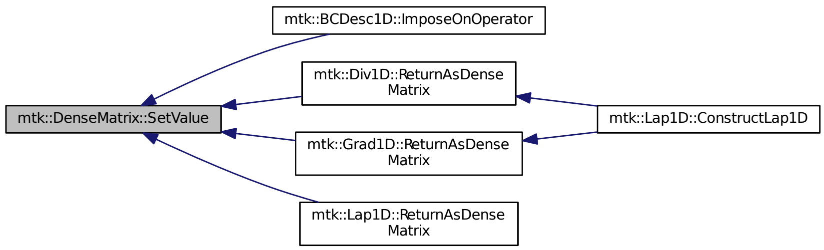
in	*row_coord*	Row coordinate.
in	*col_coord*	Column coordinate.
in	*val*	Row Actual value to be inserted.

Definition at line 333 of file mtk_dense_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:

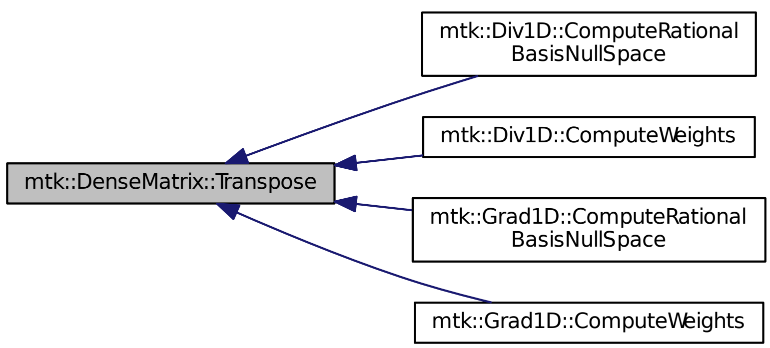


16.3.3.12 void mtk::DenseMatrix::Transpose ( )

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

Definition at line 346 of file mtk_dense_matrix.cc.

Here is the caller graph for this function:



16.3.4 Friends And Related Function Documentation

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

Definition at line 75 of file mtk_dense_matrix.cc.

16.3.5 Member Data Documentation

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

Definition at line 274 of file mtk_dense_matrix.h.

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

Definition at line 272 of file mtk_dense_matrix.h.

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

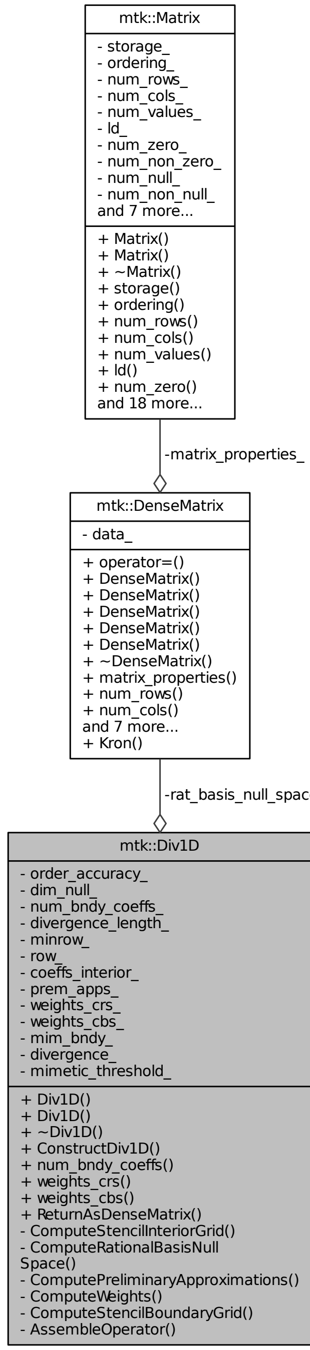
- include/mtk_dense_matrix.h
- src/mtk_dense_matrix.cc

16.4 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 ()

 Returns how many coefficients are approximating at the boundary.
- Real ∗ weights_crs (void)

 Return collection of weights as computed by the CRSA.
- Real ∗ weights_cbs (void)

 Return collection of weights as computed by the CBSA.
- DenseMatrix ReturnAsDenseMatrix (const UniStgGrid1D &grid)

 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.
- mtk::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.

16.4.1 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.

16.4.2 Constructor & Destructor Documentation

16.4.2.1 mtk::Div1D::Div1D ( )

Definition at line 125 of file mtk_div_1d.cc.

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

Parameters

in	*div*	Given divergence.

Definition at line 140 of file mtk_div_1d.cc.

16.4.2.3 mtk::Div1D::∼Div1D ( )

Definition at line 155 of file mtk_div_1d.cc.

16.4.3 Member Function Documentation

16.4.3.1 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 1311 of file mtk_div_1d.cc.

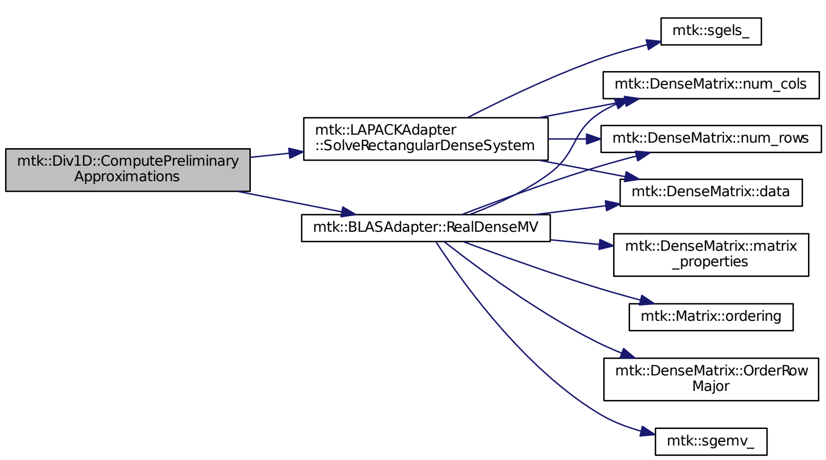
16.4.3.2 bool mtk::Div1D::ComputePreliminaryApproximations (void) [private]

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 667 of file mtk_div_1d.cc.

Here is the call graph for this function:



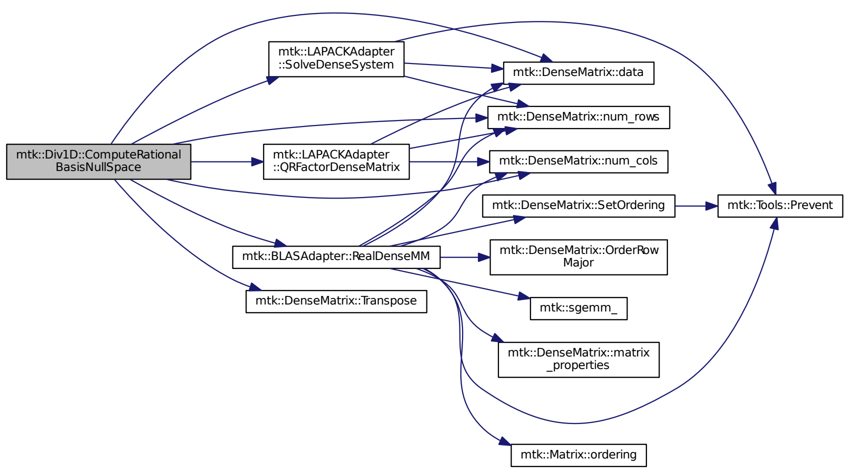
16.4.3.3 bool mtk::Div1D::ComputeRationalBasisNullSpace (void) [private]

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 491 of file mtk_div_1d.cc.

Here is the call graph for this function:



16.4.3.4 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 1212 of file mtk_div_1d.cc.

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

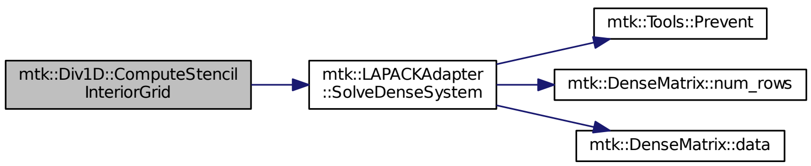
Compute the stencil approximating the interior of the staggered grid.

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 392 of file mtk_div_1d.cc.

Here is the call graph for this function:



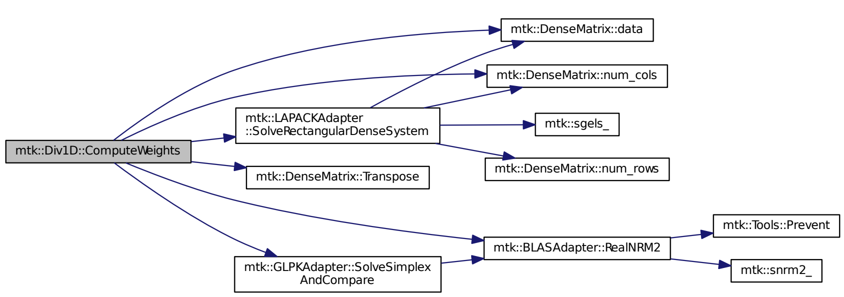
16.4.3.6 bool mtk::Div1D::ComputeWeights (void) [private]

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**): ■$\mathbf{q} = \mathbf{h}$.
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 887 of file mtk_div_1d.cc.

Here is the call graph for this function:



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

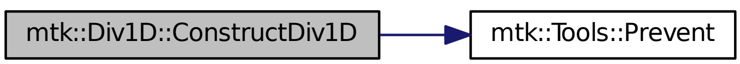
Returns

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:

16.4.3.8 int mtk::Div1D::num_bndy_coeffs ( )

Returns

How many coefficients are approximating at the boundary.

Definition at line 315 of file mtk_div_1d.cc.

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

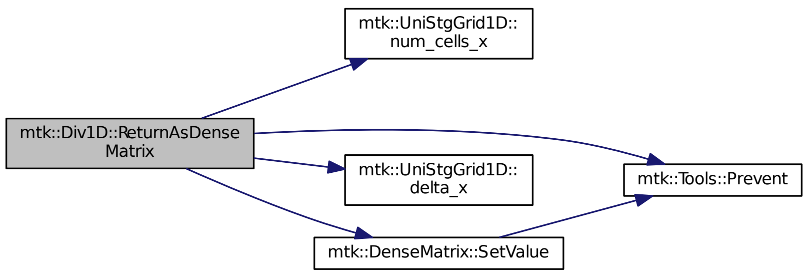
Returns

The operator as a dense matrix.

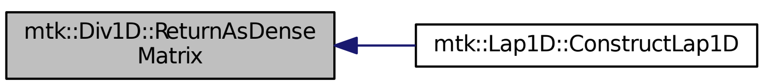
1. Insert mimetic boundary at the west.
2. Insert coefficients for the interior of the grid.
3. Impose center-skew symmetry by permuting the mimetic boundaries.

Definition at line 330 of file mtk_div_1d.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



16.4.3.10 mtk::Real ∗ mtk::Div1D::weights_cbs (void)

Returns

Collection of weights as computed by the CBSA.

Definition at line 325 of file mtk_div_1d.cc.

16.4.3.11 mtk::Real ∗ mtk::Div1D::weights_crs (void)

Returns

Collection of weights as computed by the CRSA.

Definition at line 320 of file mtk_div_1d.cc.

16.4.4 Friends And Related Function Documentation

16.4.4.1 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.

16.4.5 Member Data Documentation

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

Definition at line 189 of file mtk_div_1d.h.

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

Definition at line 180 of file mtk_div_1d.h.

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

Definition at line 194 of file mtk_div_1d.h.

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

Definition at line 182 of file mtk_div_1d.h.

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

Definition at line 193 of file mtk_div_1d.h.

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

Definition at line 196 of file mtk_div_1d.h.

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

Definition at line 184 of file mtk_div_1d.h.

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

Definition at line 181 of file mtk_div_1d.h.

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

Definition at line 179 of file mtk_div_1d.h.

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

Definition at line 190 of file mtk_div_1d.h.

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

Definition at line 187 of file mtk_div_1d.h.

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

Definition at line 185 of file mtk_div_1d.h.

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

Definition at line 192 of file mtk_div_1d.h.

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

Definition at line 191 of file mtk_div_1d.h.

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

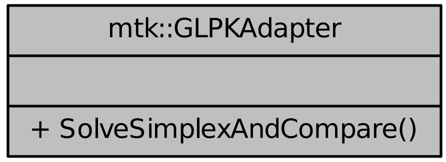
- include/mtk_div_1d.h
- src/mtk_div_1d.cc

16.5 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.

16.5.1 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 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.

16.5.2 Member Function Documentation

16.5.2.1 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.

Warning

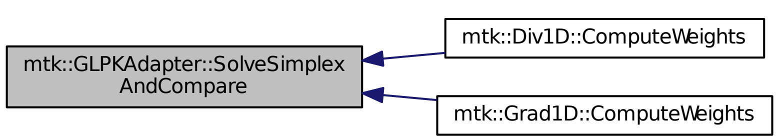
GLPK indexes in [1,n], so we must get the extra space needed.

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 78 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:

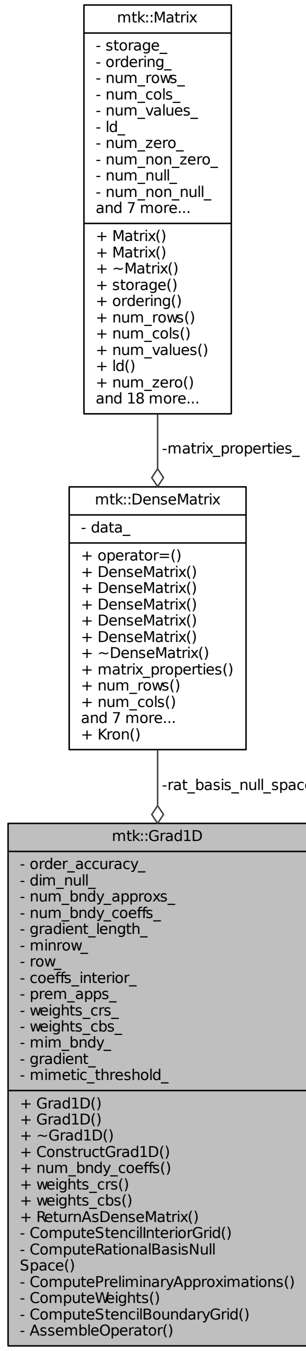
- include/mtk_glpk_adapter.h
- src/mtk_glpk_adapter.cc

16.6 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=kDefaultMimeticThreshold)

  Factory method implementing the CBS Algorithm to build operator.
- int num_bndy_coeffs ()

  Returns how many coefficients are approximating at the boundary.
- Real ∗ weights_crs (void)

  Returns collection of weights as computed by the CRSA.
- Real ∗ weights_cbs (void)

  Returns collection of weights as computed by the CBSA.
- DenseMatrix ReturnAsDenseMatrix (const UniStgGrid1D &grid)

  Returns 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_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.
- mtk::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.

16.6.1 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.

16.6.2 Constructor & Destructor Documentation

16.6.2.1 mtk::Grad1D::Grad1D ( )

Definition at line 129 of file mtk_grad_1d.cc.

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

Parameters

in	*div*	Given divergence.

Definition at line 145 of file mtk_grad_1d.cc.

16.6.2.3 mtk::Grad1D::∼Grad1D ( )

Definition at line 161 of file mtk_grad_1d.cc.

16.6.3 Member Function Documentation

16.6.3.1 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 1349 of file mtk_grad_1d.cc.

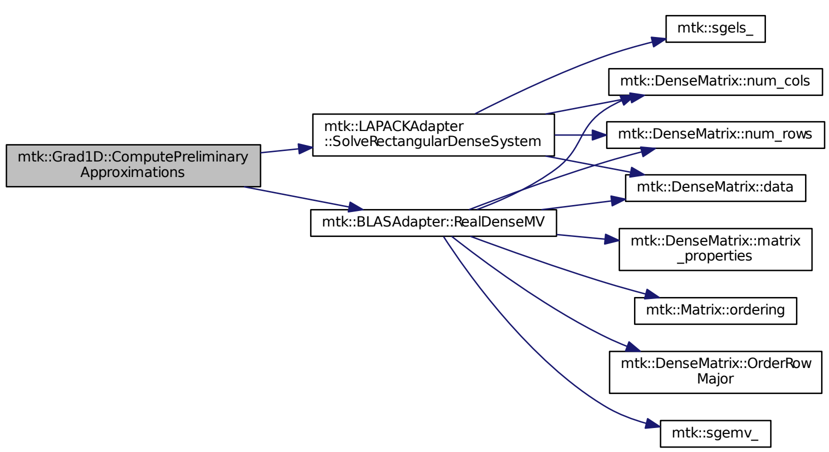
16.6.3.2 bool mtk::Grad1D::ComputePreliminaryApproximations (void) [private]

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 685 of file mtk_grad_1d.cc.

Here is the call graph for this function:



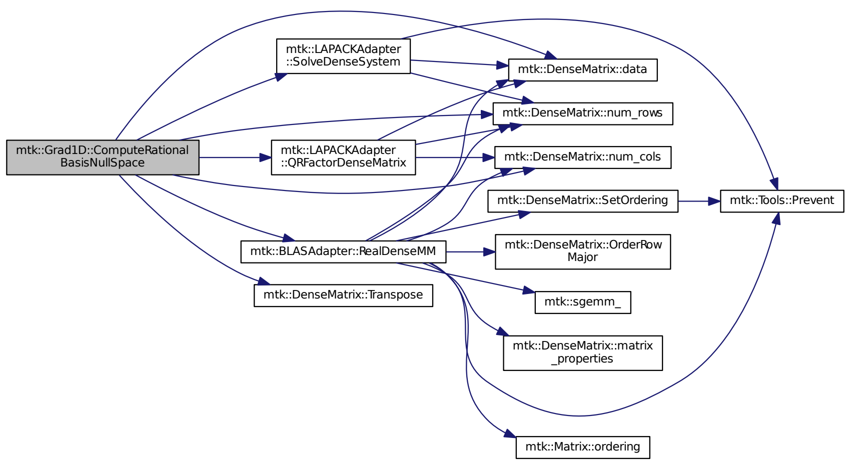
16.6.3.3 bool mtk::Grad1D::ComputeRationalBasisNullSpace (void) [private]

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 502 of file mtk_grad_1d.cc.

Here is the call graph for this function:



16.6.3.4 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 1243 of file mtk_grad_1d.cc.

16.6.3.5 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 406 of file mtk_grad_1d.cc.

Here is the call graph for this function:

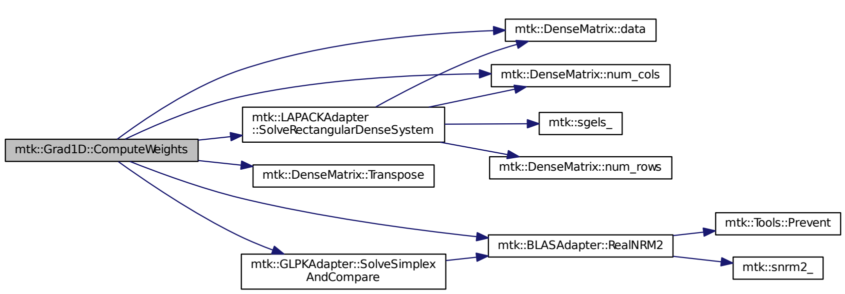
16.6.3.6 bool mtk::Grad1D::ComputeWeights (void) [private]

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 905 of file mtk_grad_1d.cc.

Here is the call graph for this function:



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

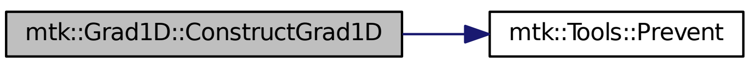
Returns

Success of the solution.

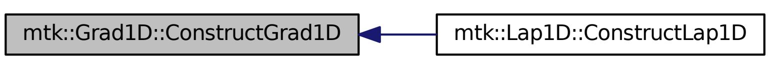
1. Compute stencil for the interior cells.
2. Compute a rational null-space from the first matrix transposed.
3. Compute preliminary approximation (non-mimetic) on the boundaries.
4. Compute quadrature weights to impose the mimetic conditions.
5. Compute real approximation (mimetic) on the boundaries.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



16.6.3.8 int mtk::Grad1D::num_bndy_coeffs ( )

Returns

How many coefficients are approximating at the boundary.

Definition at line 325 of file mtk_grad_1d.cc.

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

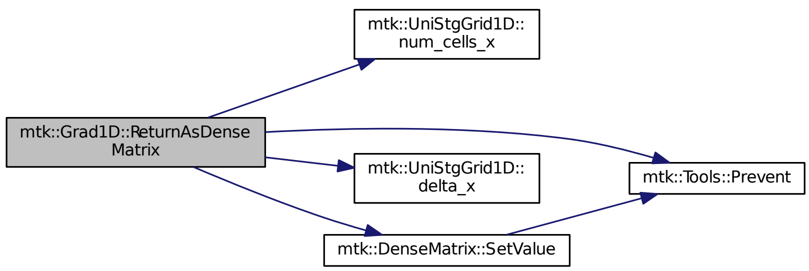
Returns

The operator as a dense matrix.

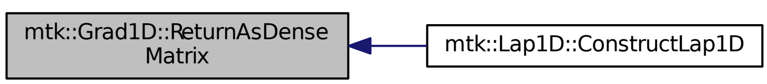
1. Insert mimetic boundary at the west.
2. Insert coefficients for the interior of the grid.
3. Impose center-skew symmetry by permuting the mimetic boundaries.

Definition at line 340 of file mtk_grad_1d.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



16.6.3.10 mtk::Real * mtk::Grad1D::weights_cbs (void)

Returns

Collection of weights as computed by the CBSA.

Definition at line 335 of file mtk_grad_1d.cc.

16.6.3.11 mtk::Real * mtk::Grad1D::weights_crs (void)

Returns

Success of the solution.

Definition at line 330 of file mtk_grad_1d.cc.

16.6.4 Friends And Related Function Documentation

16.6.4.1 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.

16.6.5 Member Data Documentation

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

Definition at line 190 of file mtk_grad_1d.h.

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

Definition at line 180 of file mtk_grad_1d.h.

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

Definition at line 195 of file mtk_grad_1d.h.

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

Definition at line 183 of file mtk_grad_1d.h.

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

Definition at line 194 of file mtk_grad_1d.h.

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

Definition at line 197 of file mtk_grad_1d.h.

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

Definition at line 185 of file mtk_grad_1d.h.

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

Definition at line 181 of file mtk_grad_1d.h.

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

Definition at line 182 of file mtk_grad_1d.h.

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

Definition at line 179 of file mtk_grad_1d.h.

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

Definition at line 191 of file mtk_grad_1d.h.

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

Definition at line 188 of file mtk_grad_1d.h.

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

Definition at line 186 of file mtk_grad_1d.h.

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

Definition at line 193 of file mtk_grad_1d.h.

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

Definition at line 192 of file mtk_grad_1d.h.

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

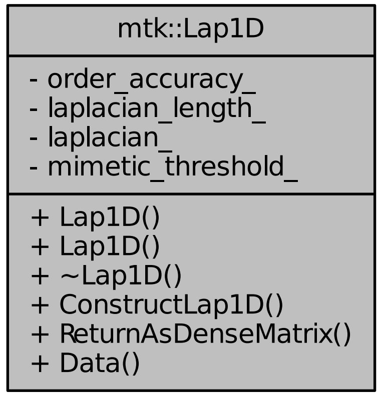
- include/mtk_grad_1d.h
- src/mtk_grad_1d.cc

16.7 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)

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

  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.

16.7.1 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.

16.7.2 Constructor & Destructor Documentation

16.7.2.1 mtk::Lap1D::Lap1D ( )

Definition at line 108 of file mtk_lap_1d.cc.

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

Parameters

in	*lap*	Given Laplacian.

16.7.2.3 mtk::Lap1D::∼Lap1D ( )

Definition at line 113 of file mtk_lap_1d.cc.

16.7.3 Member Function Documentation

16.7.3.1 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.

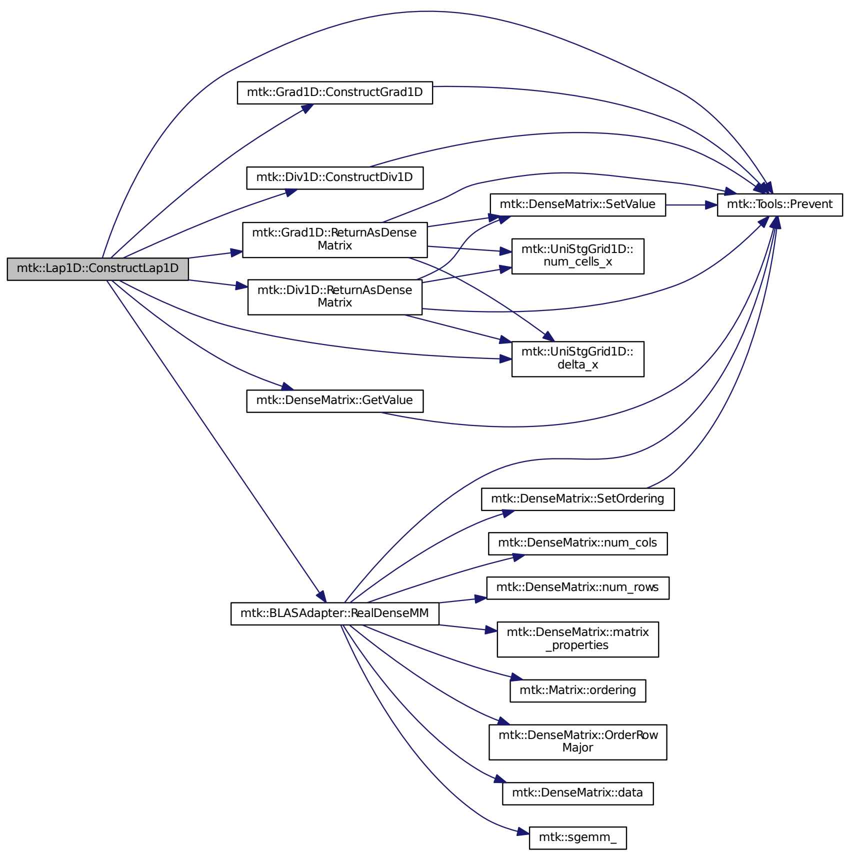
Warning

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:



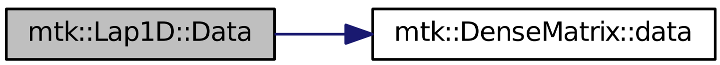
16.7.3.2 mtk::Real ∗ mtk::Lap1D::Data (const UniStgGrid1D & *grid*)

Returns

The operator as a dense array.

Definition at line 332 of file mtk_lap_1d.cc.

Here is the call graph for this function:



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

Returns

The operator as a dense matrix.

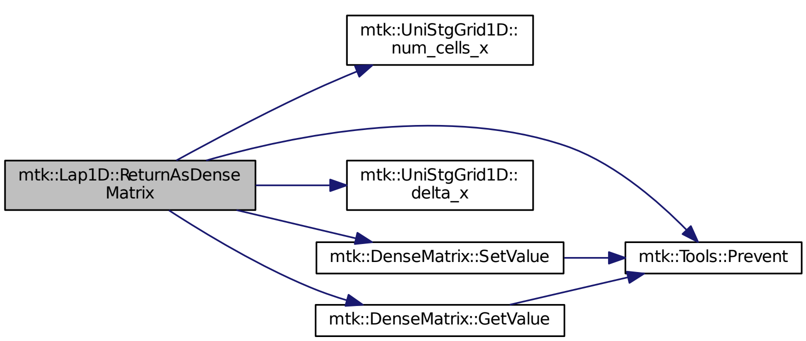
1. Extract mimetic coefficients from the west boundary.
2. Extract interior coefficients.
3. Extract mimetic coefficients from the west boundary to go east.

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:



16.7.4 Friends And Related Function Documentation

16.7.4.1 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.

16.7.5 Member Data Documentation

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

Definition at line 120 of file mtk_lap_1d.h.

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

Definition at line 118 of file mtk_lap_1d.h.

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

Definition at line 122 of file mtk_lap_1d.h.

16.7.5.4 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:

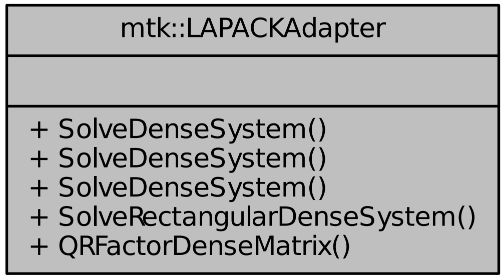
- include/mtk_lap_1d.h
- src/mtk_lap_1d.cc

16.8 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.

16.8.1 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** 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 90 of file mtk_lapack_adapter.h.

16.8.2 Member Function Documentation

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

Adapts the MTK to LAPACK's routine.

Parameters

in,out	*matrix*	Input matrix.

Returns

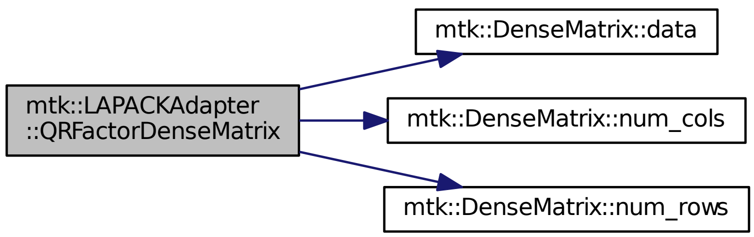
Matrix **Q**.

Exceptions

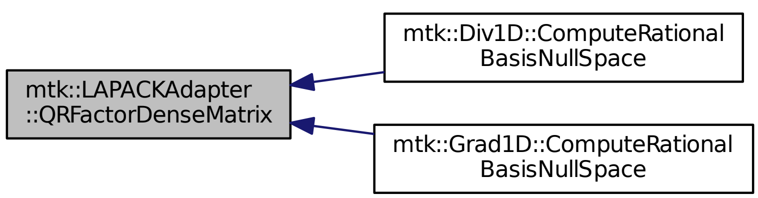
std::bad_alloc	

Definition at line 553 of file mtk_lapack_adapter.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



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

Adapts the MTK to LAPACK's dgesv_ routine.

Parameters

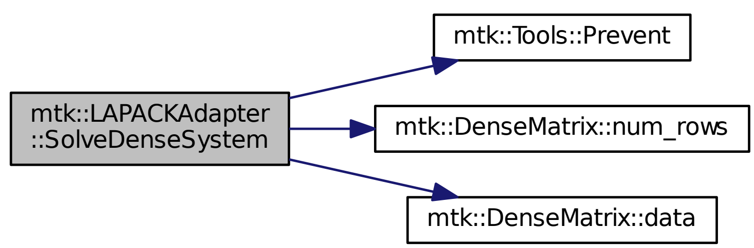
in	*matrix*	Input matrix.
in	*rhs*	Input right-hand sides vector.

Exceptions

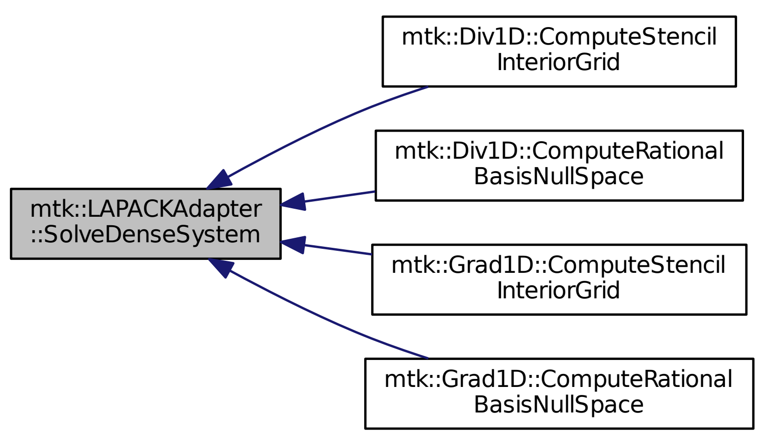
std::bad_alloc	

Definition at line 427 of file mtk_lapack_adapter.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



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

Adapts the MTK to LAPACK's dgesv_ routine.

Parameters

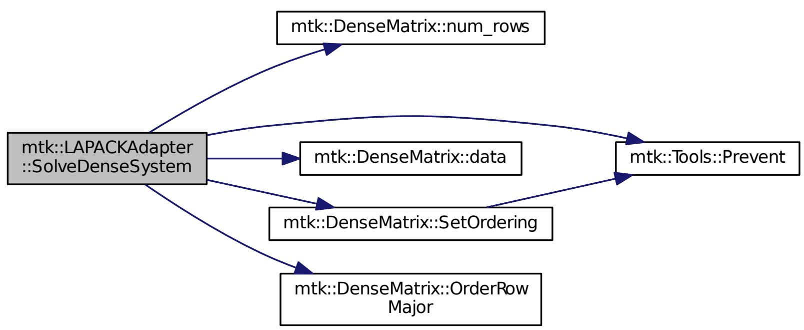
in	*matrix*	Input matrix.
in	*rr*	Input right-hand sides matrix.

Exceptions

std::bad_alloc	

Definition at line 463 of file mtk_lapack_adapter.cc.

Here is the call graph for this function:



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

Adapts the MTK to LAPACK's dgesv_ routine.

Parameters

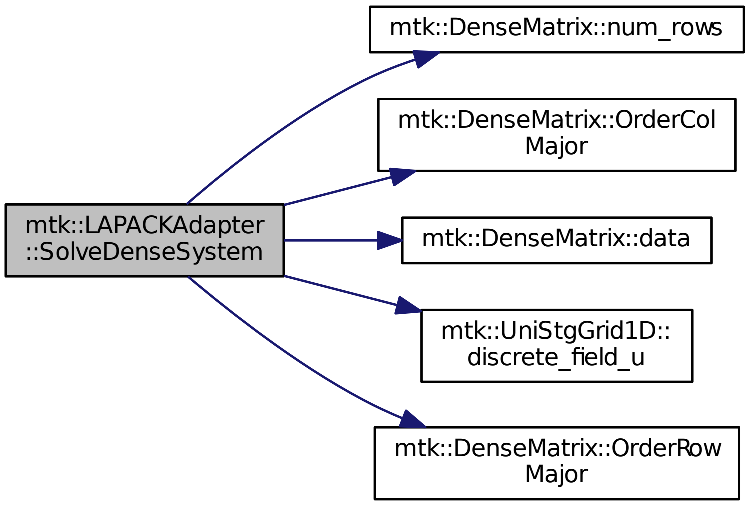
in	*matrix*	Input matrix.
in	*rr*	Input right-hand side from info on a grid.

Exceptions

std::bad_alloc	

Definition at line 515 of file mtk_lapack_adapter.cc.

Here is the call graph for this function:



16.8.2.5 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

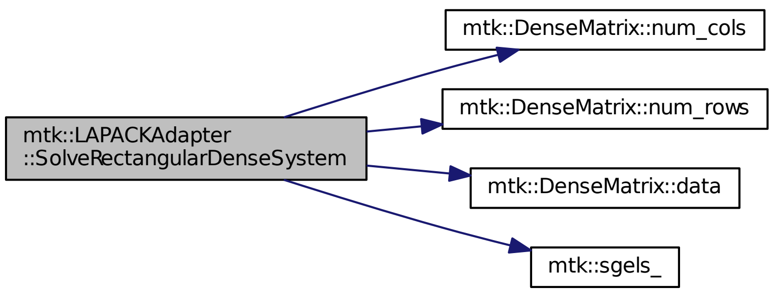
Success of the solution.

Exceptions

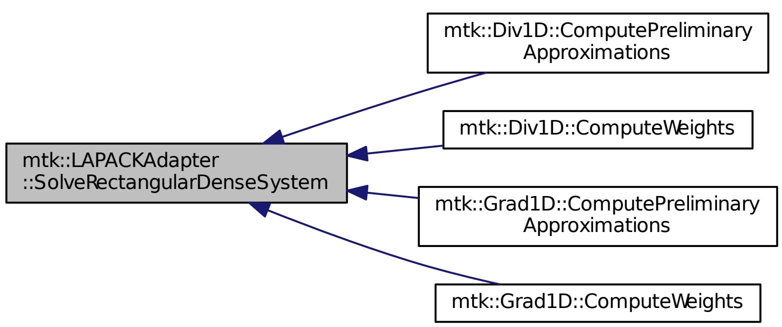
std::bad_alloc	

Definition at line 754 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:

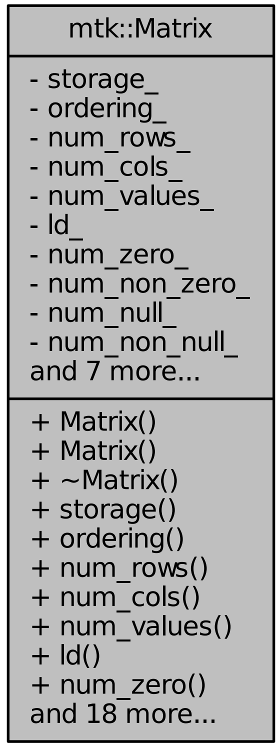
- include/mtk_lapack_adapter.h
- src/mtk_lapack_adapter.cc

16.9 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 ()

    Destructor.
- MatrixStorage storage () const

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

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

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

    Gets the number of rows.

- int num_values () const

    Gets the number of values.

- int ld () const

    Gets the matrix' leading dimension.

- int num_zero () const

    Gets the number of zeros.

- int num_non_zero () const

    Gets the number of non-zero values.

- int num_null () const

    Gets the number of null values.

- int num_non_null () const

    Gets the number of non-null values.

- int kl () const

    Gets the number of lower diagonals.

- int ku () const

    Gets the number of upper diagonals.

- int bandwidth () const

    Gets the bandwidth.

- Real abs_density () const

    Gets the absolute density.

- Real rel_density () const

    Gets the relative density.

- Real abs_sparsity () const

    Gets the Absolute sparsity.

- Real rel_sparsity () const

    Gets the Relative sparsity.

- void set_storage (const MatrixStorage &tt)

    Sets the storage type of the matrix.

- void set_ordering (const MatrixOrdering &oo)

    Sets the ordering of the matrix.

- void set_num_rows (int num_rows)

    Sets the number of rows of the matrix.

- void set_num_cols (int num_cols)

    Sets the number of columns of the matrix.

- void set_num_zero (int in)

    Sets the number of zero values of the matrix that matter.

- void set_num_null (int in)

    Sets the number of zero values of the matrix that DO NOT matter.

- void IncreaseNumZero ()

    Increases the number of values that equal zero but with meaning.

- void IncreaseNumNull ()

    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.

16.9.1 Detailed Description

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

Definition at line 75 of file mtk_matrix.h.

16.9.2 Constructor & Destructor Documentation

16.9.2.1 mtk::Matrix::Matrix ( )

Definition at line 72 of file mtk_matrix.cc.

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

Parameters

in	*in*	Given matrix.

Definition at line 91 of file mtk_matrix.cc.

16.9.2.3 mtk::Matrix::∼Matrix ( )

Definition at line 110 of file mtk_matrix.cc.

16.9.3 Member Function Documentation

16.9.3.1 Real mtk::Matrix::abs_density ( ) const

See Also

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

Returns

Absolute density of the matrix.

16.9.3.2 mtk::Real mtk::Matrix::abs_sparsity ( ) const

See Also

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

Returns

Absolute sparsity of the matrix.

Definition at line 182 of file mtk_matrix.cc.

16.9.3.3 int mtk::Matrix::bandwidth ( ) const

Returns

Bandwidth of the matrix.

Definition at line 172 of file mtk_matrix.cc.

16.9.3.4 void mtk::Matrix::IncreaseNumNull ( )

Todo Review the definition of sparse matrices properties.

Definition at line 279 of file mtk_matrix.cc.

16.9.3.5 void mtk::Matrix::IncreaseNumZero ( )

Todo Review the definition of sparse matrices properties.

Definition at line 269 of file mtk_matrix.cc.

16.9.3.6 int mtk::Matrix::kl ( ) const

Returns

Number of lower diagonals.

Definition at line 162 of file mtk_matrix.cc.

16.9.3.7 int mtk::Matrix::ku ( ) const

Returns

Number of upper diagonals.

Definition at line 167 of file mtk_matrix.cc.

16.9.3.8 int mtk::Matrix::ld ( ) const

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 137 of file mtk_matrix.cc.

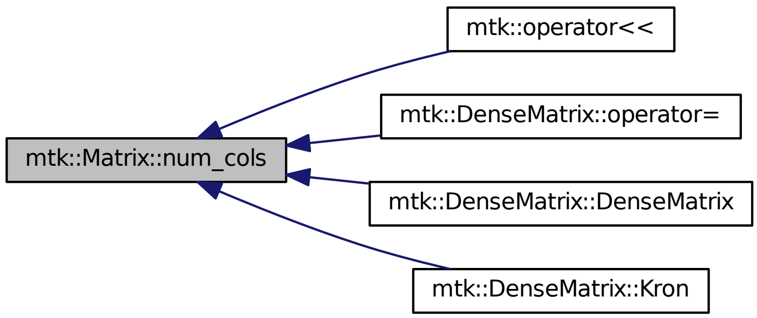
16.9.3.9 int mtk::Matrix::num_cols ( ) const

Returns

Number of rows of the matrix.

Definition at line 127 of file mtk_matrix.cc.

Here is the caller graph for this function:



16.9.3.10 int mtk::Matrix::num_non_null ( ) const

See Also

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

Returns

Number of non-null values of the matrix.

Definition at line 157 of file mtk_matrix.cc.

16.9.3.11 int mtk::Matrix::num_non_zero ( ) const

Returns

Number of non-zero values of the matrix.

Definition at line 147 of file mtk_matrix.cc.

16.9.3.12 int mtk::Matrix::num_null ( ) const

See Also

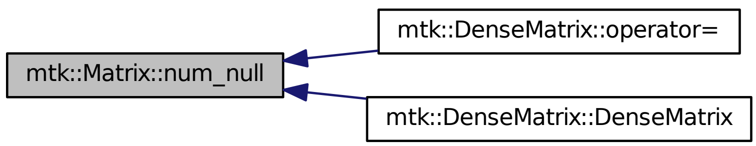
http://www.csrc.sdsu.edu/research_reports/CSRCR2013-01.pdf

Returns

Number of null values of the matrix.

Definition at line 152 of file mtk_matrix.cc.

Here is the caller graph for this function:



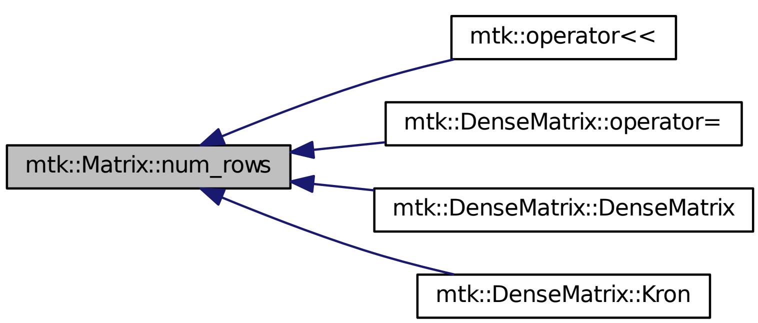
16.9.3.13 int mtk::Matrix::num_rows ( ) const

Returns

Number of rows of the matrix.

Definition at line 122 of file mtk_matrix.cc.

Here is the caller graph for this function:



16.9.3.14 int mtk::Matrix::num_values ( ) const

Returns

Number of values of the matrix.

Definition at line 132 of file mtk_matrix.cc.

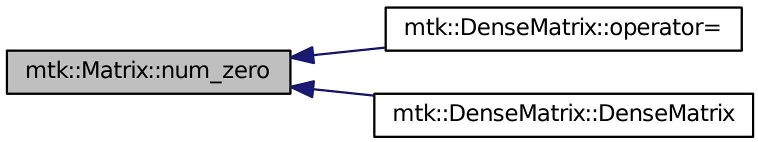
16.9.3.15 int mtk::Matrix::num_zero ( ) const

Returns

Number of zeros of the matrix.

Definition at line 142 of file mtk_matrix.cc.

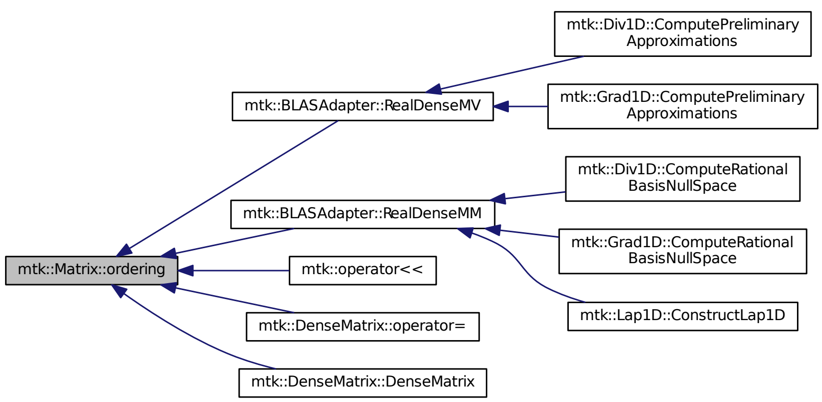
Here is the caller graph for this function:



16.9.3.16 mtk::MatrixOrdering mtk::Matrix::ordering ( ) const

Definition at line 117 of file mtk_matrix.cc.

Here is the caller graph for this function:



16.9.3.17 mtk::Real mtk::Matrix::rel_density ( ) const

See Also

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

Returns

Relative density of the matrix.

Definition at line 177 of file mtk_matrix.cc.

16.9.3.18 mtk::Real mtk::Matrix::rel_sparsity ( ) const

See Also

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

Returns

Relative sparsity of the matrix.

Definition at line 187 of file mtk_matrix.cc.

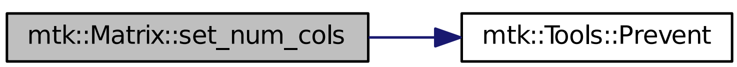
16.9.3.19 void mtk::Matrix::set_num_cols (int *num_cols*)

Parameters

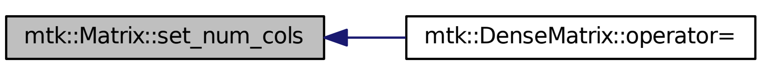
in	*num_cols*	Number of columns.

Definition at line 229 of file mtk_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



16.9.3.20 void mtk::Matrix::set_num_null (int *in*)

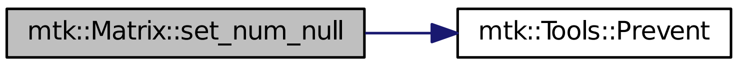
Parameters

in	*in*	Number of zero values.

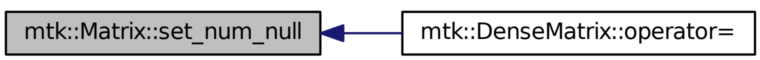
Bug -nan assigned on construction time due to num_values_ being 0.

Definition at line 255 of file mtk_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



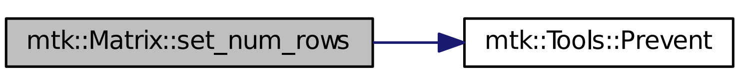
16.9.3.21 void mtk::Matrix::set_num_rows (int *num_rows*)

Parameters

in	*num_rows*	Number of rows.

Definition at line 217 of file mtk_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:

16.9.3.22 void mtk::Matrix::set_num_zero (int *in*)

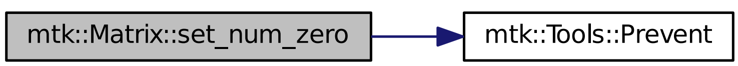
Parameters

in	*in*	Number of zero values.

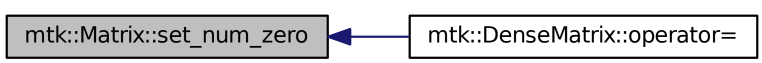
Bug -nan assigned on construction time due to num_values_ being 0.

Definition at line 241 of file mtk_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



16.9.3.23 void mtk::Matrix::set_ordering (const MatrixOrdering & *oo*)

See Also

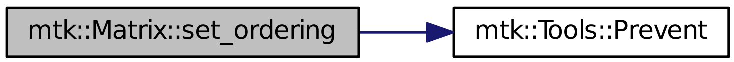
MatrixOrdering

Parameters

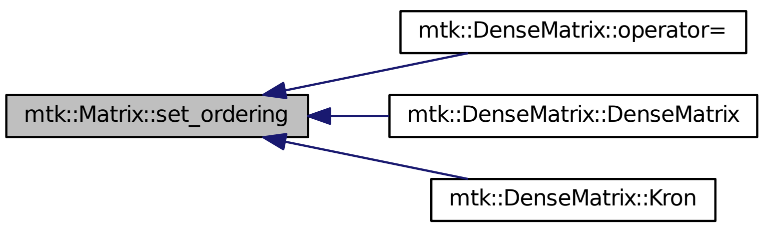
in	*oo*	Ordering of the matrix.

Definition at line 204 of file mtk_matrix.cc.

Here is the call graph for this function:



Here is the caller graph for this function:



16.9.3.24 void mtk::Matrix::set_storage (const MatrixStorage & *tt*)

See Also

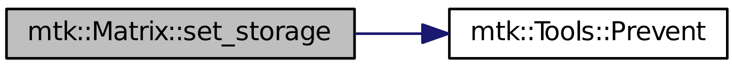
MatrixStorage

Parameters

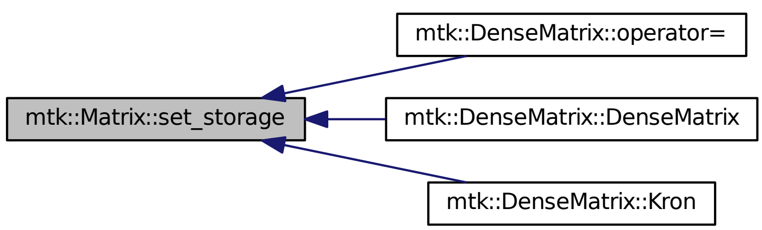
in	*tt*	Type of the matrix storage.

Definition at line 192 of file mtk_matrix.cc.

Here is the call graph for this function:



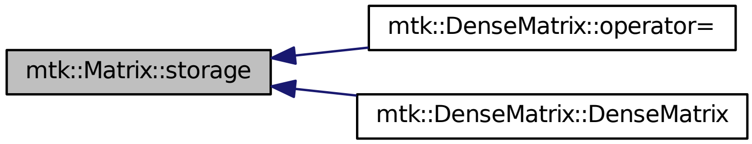
Here is the caller graph for this function:



16.9.3.25 mtk::MatrixStorage mtk::Matrix::storage ( ) const

Definition at line 112 of file mtk_matrix.cc.

Here is the caller graph for this function:



16.9.4 Member Data Documentation

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

Definition at line 288 of file mtk_matrix.h.

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

Definition at line 290 of file mtk_matrix.h.

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

Definition at line 286 of file mtk_matrix.h.

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

Definition at line 284 of file mtk_matrix.h.

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

Definition at line 285 of file mtk_matrix.h.

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

Definition at line 277 of file mtk_matrix.h.

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

Definition at line 275 of file mtk_matrix.h.

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

Definition at line 282 of file mtk_matrix.h.

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

Definition at line 280 of file mtk_matrix.h.

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

Definition at line 281 of file mtk_matrix.h.

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

Definition at line 274 of file mtk_matrix.h.

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

Definition at line 276 of file mtk_matrix.h.

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

Definition at line 279 of file mtk_matrix.h.

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

Definition at line 272 of file mtk_matrix.h.

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

Definition at line 289 of file mtk_matrix.h.

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

Definition at line 291 of file mtk_matrix.h.

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

Definition at line 270 of file mtk_matrix.h.

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

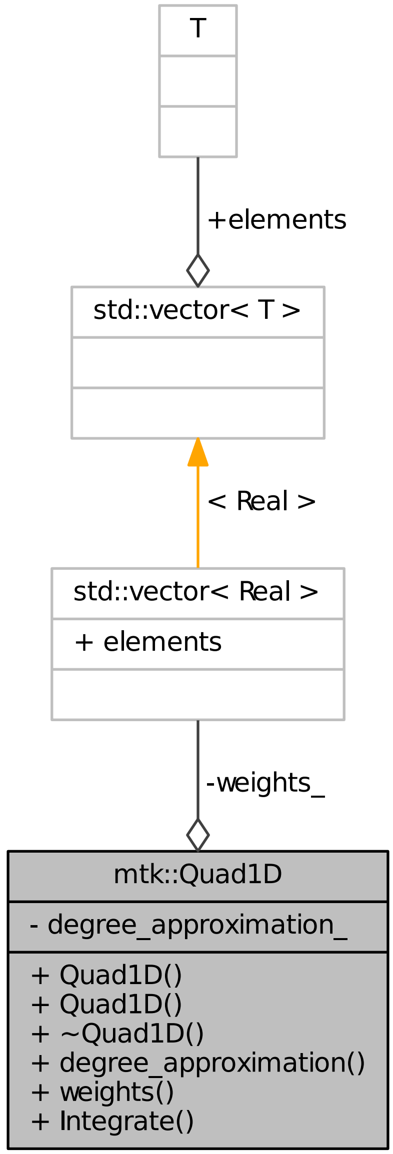
- include/mtk_matrix.h
- src/mtk_matrix.cc

16.10 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)
  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.

16.10.1 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.

16.10.2 Constructor & Destructor Documentation

16.10.2.1 mtk::Quad1D::Quad1D ( )

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

Parameters

in	*div*	Given quadrature.

16.10.2.3 mtk::Quad1D::∼Quad1D ( )

16.10.3 Member Function Documentation

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

Returns

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

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

Parameters

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

Returns

Result of the integration.

16.10.3.3 Real* mtk::Quad1D::weights ( ) const

Returns

Collection of weights.

16.10.4 Friends And Related Function Documentation

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

16.10.5 Member Data Documentation

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

Definition at line 124 of file mtk_quad_1d.h.

16.10.5.2 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:

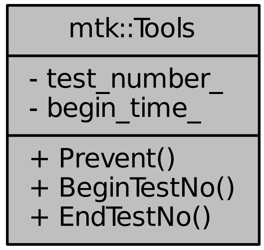
- include/mtk_quad_1d.h

16.11 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 condition, const char ∗fname, int lineno, const char ∗fxname)

  Enforces pre-conditions by preventing their complements from occur.
- static void BeginTestNo (const int &nn)

  Begins the execution of a test.
- static void EndTestNo (const int &nn)

  Ends the execution of a test.

Static Private Attributes

- static int test_number_

  Current test being executed.
- static clock_t begin_time_

  Elapsed time on current test.

16.11.1 Detailed Description

Basic tools to ensure execution correctness.

Definition at line 72 of file mtk_tools.h.

16.11.2 Member Function Documentation

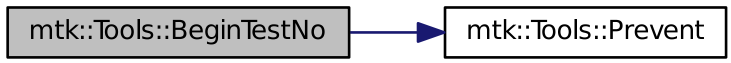
16.11.2.1 void mtk::Tools::BeginTestNo (const int & *nn*) [static]

Parameters

in	*nn*	Number of the test.

Definition at line 89 of file mtk_tools.cc.

Here is the call graph for this function:



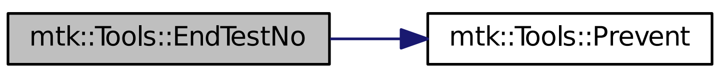
16.11.2.2 void mtk::Tools::EndTestNo (const int & *nn*) [static]

Parameters

in	*nn*	Number of the test.

Definition at line 101 of file mtk_tools.cc.

Here is the call graph for this function:



16.11.2.3 void mtk::Tools::Prevent (const bool *condition,* const char ∗ *fname,* int *lineno,* const char ∗ *fxname*) [static]

See Also

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

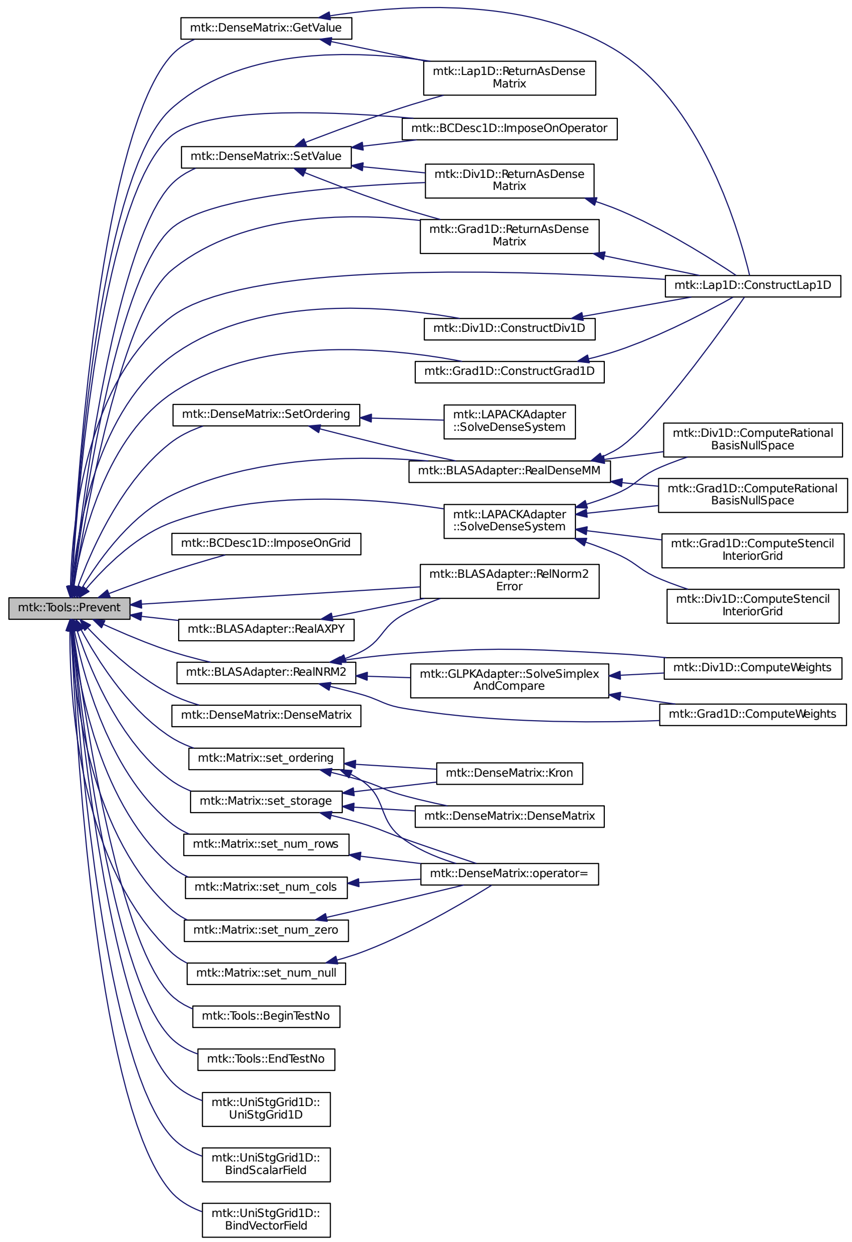
Parameters

in	*condition*	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:



16.11.3 Member Data Documentation

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

Definition at line 106 of file mtk_tools.h.

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

Todo Check usage of static methods and private members.

Definition at line 104 of file mtk_tools.h.

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

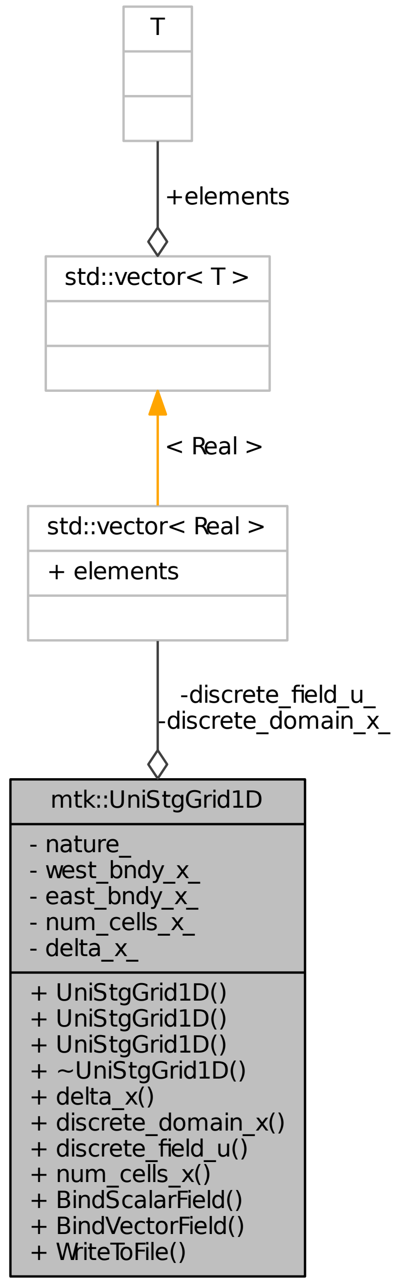
- include/mtk_tools.h
- src/mtk_tools.cc

16.12 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 delta_x () const

  Provides access to the computed $ x $.

- Real ∗ discrete_domain_x ()

  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)

  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.

16.12.1 Detailed Description

Uniform 1D Staggered Grid.

Definition at line 77 of file mtk_uni_stg_grid_1d.h.

16.12.2 Constructor & Destructor Documentation

16.12.2.1 mtk::UniStgGrid1D::UniStgGrid1D ( )

Definition at line 97 of file mtk_uni_stg_grid_1d.cc.

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

Parameters

in	*grid*	Given grid.

Definition at line 106 of file mtk_uni_stg_grid_1d.cc.

16.12.2.3 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

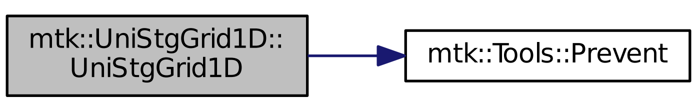
in	*west_bndy_x*	Coordinate for the west boundary.
in	*east_bndy_x*	Coordinate for the east boundary.
in	*num_cells_x*	Number of cells of the required grid.
in	*nature*	Nature of the discrete field to hold.

See Also

mtk::FieldNature

Definition at line 122 of file mtk_uni_stg_grid_1d.cc.

Here is the call graph for this function:



16.12.2.4 mtk::UniStgGrid1D::∼UniStgGrid1D ( )

Definition at line 142 of file mtk_uni_stg_grid_1d.cc.

16.12.3 Member Function Documentation

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

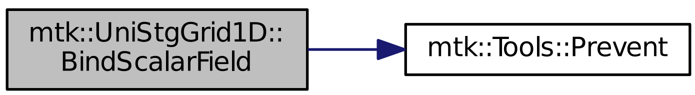
Parameters

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 164 of file mtk_uni_stg_grid_1d.cc.

Here is the call graph for this function:



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

We assume the field to be of the form:

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

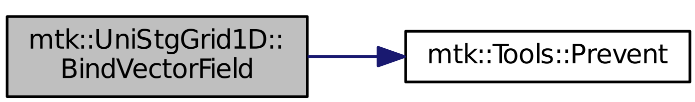
Parameters

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 200 of file mtk_uni_stg_grid_1d.cc.

Here is the call graph for this function:



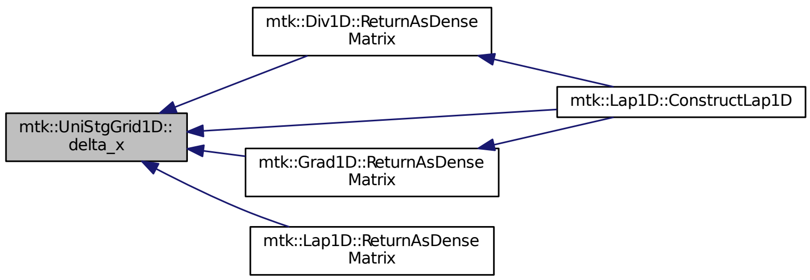
16.12.3.3 mtk::Real mtk::UniStgGrid1D::delta_x ( ) const

Returns

Computed $ x $.

Definition at line 144 of file mtk_uni_stg_grid_1d.cc.

Here is the caller graph for this function:



16.12.3.4 mtk::Real * mtk::UniStgGrid1D::discrete_domain_x ( )

Returns

Pointer to the spatial data.

Definition at line 149 of file mtk_uni_stg_grid_1d.cc.

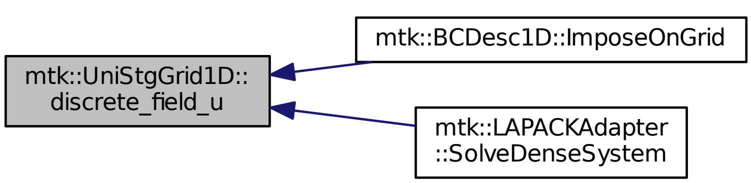
16.12.3.5 mtk::Real * mtk::UniStgGrid1D::discrete_field_u ( )

Returns

Pointer to the field data.

Definition at line 154 of file mtk_uni_stg_grid_1d.cc.

Here is the caller graph for this function:



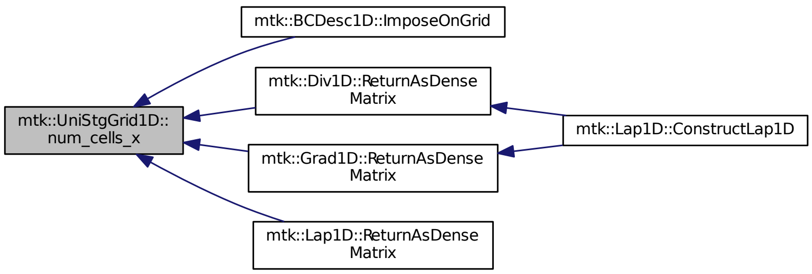
16.12.3.6 int mtk::UniStgGrid1D::num_cells_x ( ) const

Returns

Number of cells of the grid.

Definition at line 159 of file mtk_uni_stg_grid_1d.cc.

Here is the caller graph for this function:



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

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 228 of file mtk_uni_stg_grid_1d.cc.

16.12.4 Friends And Related Function Documentation

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

1. Print spatial coordinates.

Definition at line 68 of file mtk_uni_stg_grid_1d.cc.

16.12.5 Member Data Documentation

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

Definition at line 182 of file mtk_uni_stg_grid_1d.h.

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

Definition at line 176 of file mtk_uni_stg_grid_1d.h.

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

Definition at line 177 of file mtk_uni_stg_grid_1d.h.

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

Definition at line 180 of file mtk_uni_stg_grid_1d.h.

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

Definition at line 174 of file mtk_uni_stg_grid_1d.h.

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

Definition at line 181 of file mtk_uni_stg_grid_1d.h.

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

Definition at line 179 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

Chapter 17

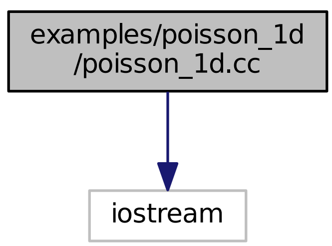
File Documentation

17.1 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 ()

17.1.1 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 poisson_1d.cc.

17.1.2 Function Documentation

17.1.2.1 int main ( )

Definition at line 261 of file poisson_1d.cc.

17.2 poisson_1d.cc

```

/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

```

```
6. 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.
*/

#if __cplusplus == 201103L

#include <iostream>
#include <fstream>
#include <cmath>

#include <vector>

#include "mtk.h"

mtk::Real Source(mtk::Real xx) {

  mtk::Real lambda = -1.0;

  return lambda*lambda*exp(lambda*xx)/(exp(lambda) - 1.0);
}

mtk::Real KnownSolution(mtk::Real xx) {

  mtk::Real lambda = -1.0;

  return (exp(lambda*xx) - 1.0)/(exp(lambda) - 1.0);
}

int main () {

  std::cout << "Example: Poisson Equation on a 1D Uniform Staggered Grid ";
  std::cout << "with Robin BCs." << std::endl;


  mtk::Real lambda = -1.0;
  mtk::Real alpha = -exp(lambda);
  mtk::Real beta = (exp(lambda) - 1.0)/lambda;
  mtk::Real omega = -1.0;
  mtk::Real epsilon = 0.0;


  mtk::Real west_bndy_x = 0.0;
  mtk::Real east_bndy_x = 1.0;
  int num_cells_x = 5;

  mtk::UniStgGrid1D comp_sol(west_bndy_x, east_bndy_x, num_cells_x);


  int order_of_accuracy{2};  // Desired order of accuracy for approximation.

  mtk::Grad1D grad;  // Mimetic gradient operator.

  mtk::Lap1D lap;  // Mimetic Laplacian operator.

  if (!lap.ConstructLap1D(order_of_accuracy)) {
    std::cerr << "Mimetic lap could not be built." << std::endl;
    return EXIT_FAILURE;
  }

  mtk::DenseMatrix lapm(lap.ReturnAsDenseMatrix(comp_sol));

  std::cout << "Mimetic Laplacian operator: " << std::endl;
  std::cout << lapm << std::endl;
```

```
  if (!grad.ConstructGrad1D(order_of_accuracy)) {
    std::cerr << "Mimetic grad could not be built." << std::endl;
    return EXIT_FAILURE;
  }

  mtk::DenseMatrix gradm(grad.ReturnAsDenseMatrix(comp_sol));

  std::cout << "Mimetic gradient operator: " << std::endl;
  std::cout << gradm << std::endl;


  mtk::UniStgGrid1D source(west_bndy_x, east_bndy_x, num_cells_x);

  source.BindScalarField(Source);

  std::cout << source << std::endl;


  // Since we need to approximate the first derivative times beta, we must use
  // the approximation of the gradient at the boundary. We could extract them
  // from the gradient operator as packed in the grad object. BUT, since we have
  // generated at matrix containing this operator, we can extract these from the
  // matrix.

  // Array containing the coefficients for the west boundary condition.
  std::vector<mtk::Real> west_coeffs;

  for (auto ii = 0; ii < grad.num_bndy_coeffs(); ++ii) {
    west_coeffs.push_back(-beta*gradm.GetValue(0, ii));
  }

  // Array containing the coefficients for the east boundary condition.
  std::vector<mtk::Real> east_coeffs;

  for (auto ii = 0; ii < grad.num_bndy_coeffs(); ++ii) {
    east_coeffs.push_back(beta*gradm.GetValue(gradm.num_rows() - 1,
                                              gradm.num_cols() - 1 - ii));
  }

  // To impose the Dirichlet condition, we simple add its coefficient to the
  // first entry of the west, and the last entry of the east array.

  west_coeffs[0] += alpha;

  east_coeffs[0] += alpha;

  // Now that we have the coefficients that should be in the operator, we create
  // a boundary condition descriptor object, which will encapsulate the
  // complexity of assigning them in the matrix, to complete the construction of
  // the mimetic operator.

  mtk::BCDesc1D::ImposeOnOperator(lapm, west_coeffs, east_coeffs);

  std::cout << "Mimetic Laplacian with Robin conditions:" << std::endl;
  std::cout << lapm << std::endl;

  mtk::BCDesc1D::ImposeOnGrid(source, omega, epsilon);

  std::cout << "Source term with imposed BCs:" << std::endl;
  std::cout << source << std::endl;

  source.WriteToFile("poisson_1d_source.dat", "x", "s(x)");


  int info{mtk::LAPACKAdapter::SolveDenseSystem(lapm, source)};

  if (!info) {
    std::cout << "System solved! Problem solved!" << std::endl;
    std::cout << std::endl;
  }
  else {
    std::cerr << "Something wrong solving system! info = " << info << std::endl;
    std::cerr << "Exiting..." << std::endl;
    return EXIT_FAILURE;
  }

  std::cout << "Computed solution:" << std::endl;
  std::cout << source << std::endl;

  source.WriteToFile("poisson_1d_comp_sol.dat", "x", "~u(x)");
```

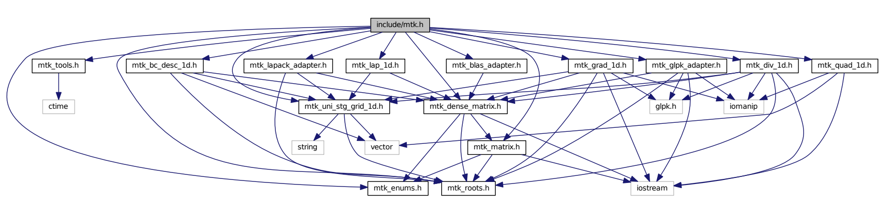
```
  mtk::UniStgGrid1D known_sol(west_bndy_x, east_bndy_x, num_cells_x);

  known_sol.BindScalarField(KnownSolution);

  std::cout << "known_sol =" << std::endl;
  std::cout << known_sol << std::endl;

  known_sol.WriteToFile("poisson_1d_known_sol.dat", "x", "u(x)");

  mtk::Real relative_norm_2_error{};  // Relative norm 2 of the error.

  relative_norm_2_error =
    mtk::BLASAdapter::RelNorm2Error(source.discrete_field_u(),
                                    known_sol.discrete_field_u(),
                                    known_sol.num_cells_x());

  std::cout << "relative_norm_2_error = ";
  std::cout << relative_norm_2_error << std::endl;
}

#else
#include <iostream>
using std::cout;
using std::endl;
int main () {
  cout << "This code HAS to be compiled with support for C++11." << endl;
  cout << "Exiting..." << endl;
  return EXIT_SUCCESS;
}
#endif
```

17.3 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_grad_1d.h"
#include "mtk_div_1d.h"
#include "mtk_lap_1d.h"
#include "mtk_quad_1d.h"
#include "mtk_bc_desc_1d.h"
```

Include dependency graph for mtk.h:



17.3.1 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.

17.4 mtk.h

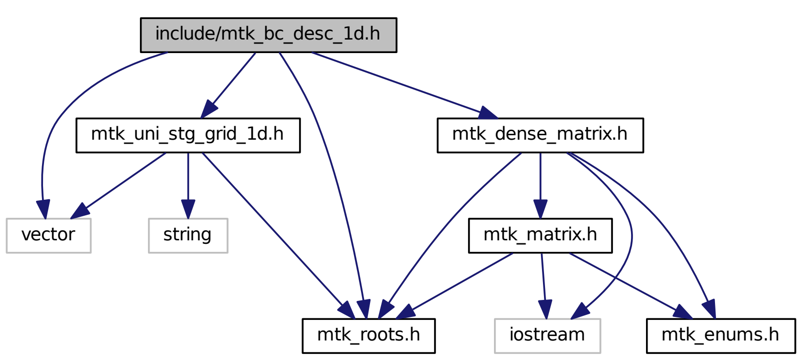
```

/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/
#ifndef MTK_INCLUDE_MTK_H_
#define MTK_INCLUDE_MTK_H_

#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_grad_1d.h"
#include "mtk_div_1d.h"
#include "mtk_lap_1d.h"
#include "mtk_quad_1d.h"

#include "mtk_bc_desc_1d.h"

#endif // End of: MTK_INCLUDE_MTK_H_
```

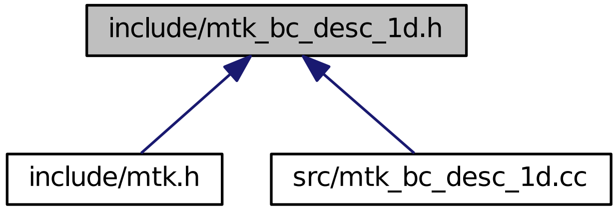
17.5 include/mtk_bc_desc_1d.h File Reference

```
#include <vector>
#include "mtk_roots.h"
#include "mtk_dense_matrix.h"
#include "mtk_uni_stg_grid_1d.h"
```

Include dependency graph for mtk_bc_desc_1d.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mtk::BCDesc1D

Namespaces

- mtk

    Mimetic Methods Toolkit namespace.

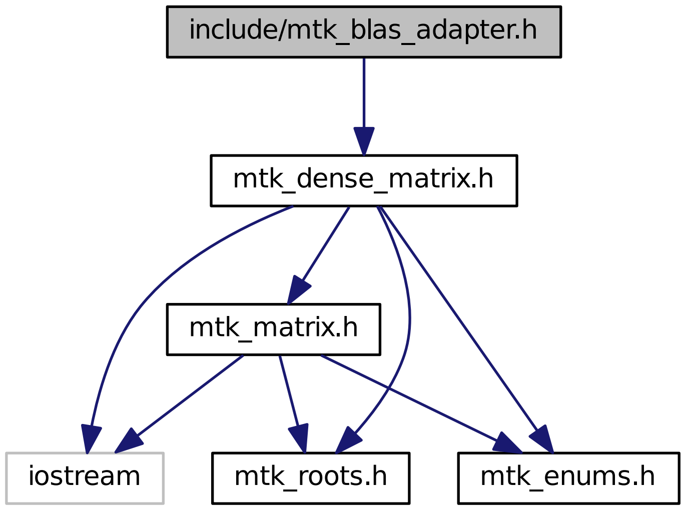
17.6 mtk_bc_desc_1d.h

```
#include <vector>

#include "mtk_roots.h"
#include "mtk_dense_matrix.h"
#include "mtk_uni_stg_grid_1d.h"

namespace mtk {

class BCDesc1D {
 public:
  static void ImposeOnOperator(DenseMatrix &matrix,
                               const std::vector<Real> &west,
                               const std::vector<Real> &east);

  static void ImposeOnGrid(UniStgGrid1D &grid,
                           const Real &omega,
                           const Real &epsilon);
};
}
```

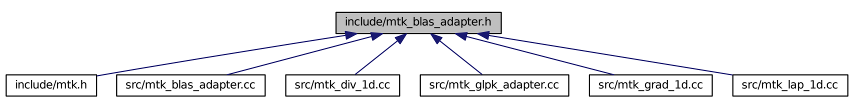
17.7 include/mtk_blas_adapter.h File Reference

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.

17.7.1 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 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.

17.8 mtk_blas_adapter.h

```
/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#ifndef MTK_INCLUDE_BLAS_ADAPTER_H_
#define MTK_INCLUDE_BLAS_ADAPTER_H_

#include "mtk_dense_matrix.h"
```

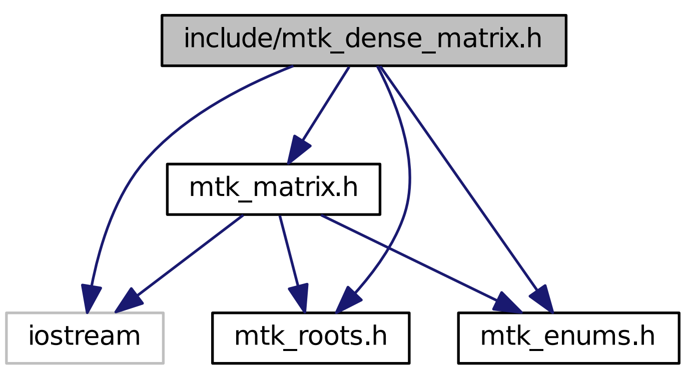
```
namespace mtk {

class BLASAdapter {
 public:
  static Real RealNRM2(Real *in, int &in_length);

  static void RealAXPY(Real alpha, Real *xx, Real *yy, int &in_length);

  static Real RelNorm2Error(Real *computed, Real *known, int length);

  static void RealDenseMV(Real &alpha,
                          DenseMatrix &aa,
                          Real *xx,
                          Real &beta,
                          Real *yy);

  static DenseMatrix RealDenseMM(DenseMatrix &aa,
 DenseMatrix &bb);
};
}
#endif  // End of: MTK_INCLUDE_BLAS_ADAPTER_H_
```

17.9 include/mtk_dense_matrix.h File Reference

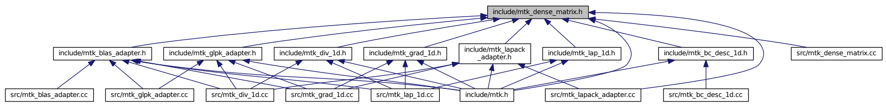
Defines a common dense matrix, using a 1D array.

```
#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.

17.9.1 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

Todo Add sparse matrices support: BANDED and CRS.

Todo Contemplate manipulation of sparse metrics.

Todo Implement Kronecker product using the BLAS.

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.

17.10 mtk_dense_matrix.h

```
/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#ifndef MTK_INCLUDE_DENSE_MATRIX_H_
#define MTK_INCLUDE_DENSE_MATRIX_H_

#include <iostream>

#include "mtk_roots.h"
#include "mtk_enums.h"
#include "mtk_matrix.h"

namespace mtk {

class DenseMatrix {
 public:
  friend std::ostream& operator <<(std::ostream &stream, DenseMatrix &in);

  DenseMatrix& operator =(const DenseMatrix &in);

  DenseMatrix();

  DenseMatrix(const DenseMatrix &in);

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

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

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

  ~DenseMatrix();
```

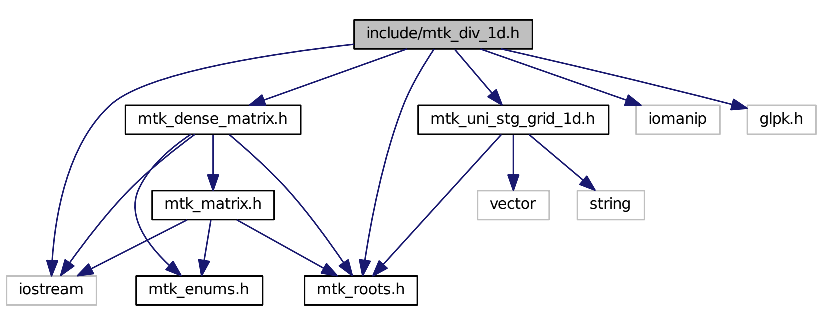
```
  Matrix matrix_properties() const;
  int num_rows() const;
  int num_cols() const;
  Real* data() const;
  void SetOrdering(mtk::MatrixOrdering oo);
  Real GetValue(const int &row_coord, const int &col_coord) const;
  void SetValue(const int &row_coord,
                const int &col_coord,
                const Real &val);
  void Transpose();
  void OrderRowMajor();
  void OrderColMajor();
  static DenseMatrix Kron(const DenseMatrix &aa, const
DenseMatrix &bb);
 private:
  Matrix matrix_properties_;
  Real *data_;
};
}
#endif  // End of: MTK_INCLUDE_MTK_DENSE_MATRIX_H_
```

17.11 include/mtk_div_1d.h File Reference

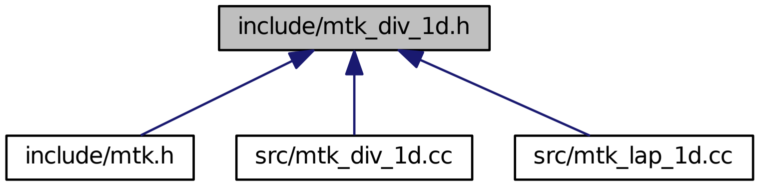
Includes the definition of the class Div1D.

```
#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.

17.11.1 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.

17.12 mtk_div_1d.h

```

/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.
```

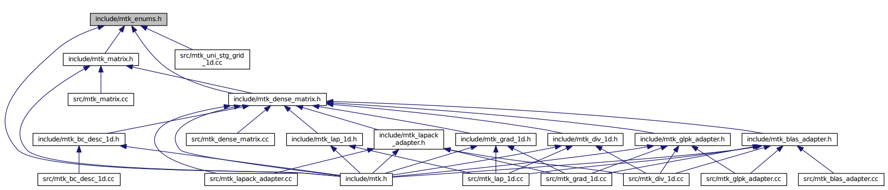
```

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#ifndef MTK_INCLUDE_DIV_1D_H_
#define MTK_INCLUDE_DIV_1D_H_

#include <iostream>
#include <iomanip>

#include "glpk.h"

#include "mtk_roots.h"
#include "mtk_dense_matrix.h"
#include "mtk_uni_stg_grid_1d.h"

namespace mtk {

class Div1D {
 public:
  friend std::ostream& operator <<(std::ostream& stream, Div1D &in);

  Div1D();

  Div1D(const Div1D &div);

  ~Div1D();

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

  int num_bndy_coeffs();

  Real* weights_crs(void);

  Real* weights_cbs(void);

  DenseMatrix ReturnAsDenseMatrix(const
UniStgGrid1D &grid);

 private:
  bool ComputeStencilInteriorGrid(void);

  bool ComputeRationalBasisNullSpace(void);

  bool ComputePreliminaryApproximations(void);

  bool ComputeWeights(void);

  bool ComputeStencilBoundaryGrid(void);

  bool AssembleOperator(void);

  int order_accuracy_;
  int dim_null_;
```

```
  int num_bndy_coeffs_;
  int divergence_length_;

  int minrow_;
  int row_;

  mtk::DenseMatrix rat_basis_null_space_;

  Real *coeffs_interior_;
  Real *prem_apps_;
  Real *weights_crs_;
  Real *weights_cbs_;
  Real *mim_bndy_;
  Real *divergence_;

  Real mimetic_threshold_;
};
}
#endif  // End of: MTK_INCLUDE_DIV_1D_H_
```

17.13 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.

17.13.1 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.

17.14 mtk_enums.h

```
/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#ifndef MTK_INCLUDE_ENUMS_H_
#define MTK_INCLUDE_ENUMS_H_

namespace mtk {

enum MatrixStorage {
  DENSE,
  BANDED,
  CRS
};

enum MatrixOrdering {
  ROW_MAJOR,
  COL_MAJOR
};

enum FieldNature {
  SCALAR,
  VECTOR
};
}
```

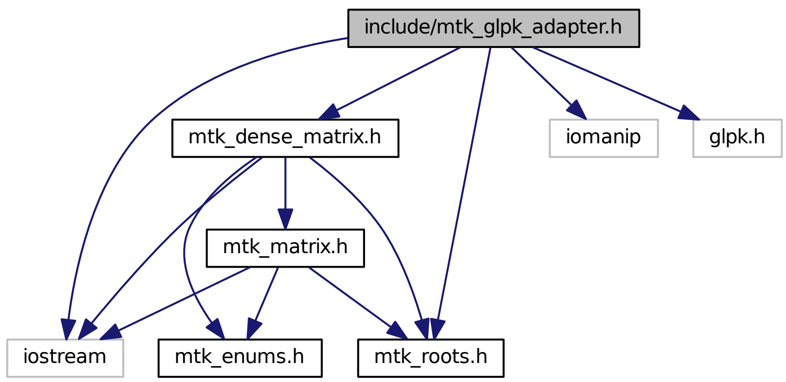
```
00118 #endif  // End of: MTK_INCLUDE_ENUMS_H_
```

17.15 include/mtk_glpk_adapter.h File Reference

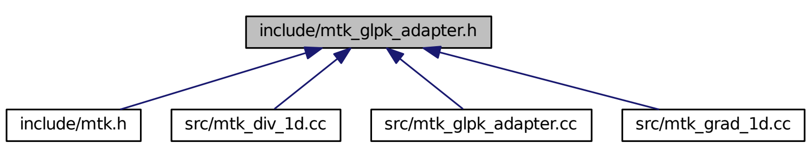
Adapter class for the GLPK API.

```
#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.

17.15.1 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.

17.16 mtk_glpk_adapter.h

```

/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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
```

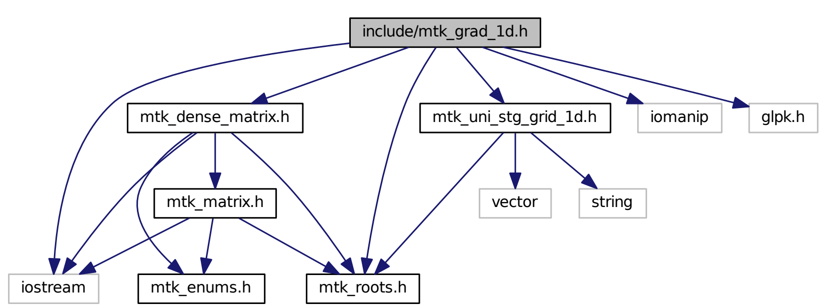
```
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#ifndef MTK_INCLUDE_GLPK_ADAPTER_H_
#define MTK_INCLUDE_GLPK_ADAPTER_H_

#include <iostream>
#include <iomanip>

#include "glpk.h"

#include "mtk_roots.h"
#include "mtk_dense_matrix.h"

namespace mtk {

class GLPKAdapter {
 public:
  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);
};
}
#endif  // End of: MTK_INCLUDE_MTK_GLPK_ADAPTER_H_
```

17.17 include/mtk_grad_1d.h File Reference

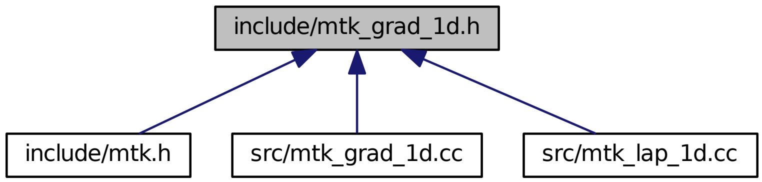
Includes the definition of the class Grad1D.

```
#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.

17.17.1 Detailed Description

This class implements a 1D gradient 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_grad_1d.h.

17.18 mtk_grad_1d.h

```

/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.
```

```
4. 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.
5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.
6. 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.
*/
#ifndef MTK_INCLUDE_GRAD_1D_H_
#define MTK_INCLUDE_GRAD_1D_H_
#include <iostream>
#include <iomanip>
#include "glpk.h"
#include "mtk_roots.h"
#include "mtk_dense_matrix.h"
#include "mtk_uni_stg_grid_1d.h"
namespace mtk {
class Grad1D {
 public:
  friend std::ostream& operator <<(std::ostream& stream, Grad1D &in);
  Grad1D();
  Grad1D(const Grad1D &grad);
  ~Grad1D();
  bool ConstructGrad1D(int order_accuracy = kDefaultOrderAccuracy,
                       Real mimetic_threshold = kDefaultMimeticThreshold);
  int num_bndy_coeffs();
  Real *weights_crs(void);
  Real *weights_cbs(void);
  DenseMatrix ReturnAsDenseMatrix(const
UniStgGrid1D &grid);
 private:
  bool ComputeStencilInteriorGrid(void);
  bool ComputeRationalBasisNullSpace(void);
  bool ComputePreliminaryApproximations(void);
  bool ComputeWeights(void);
  bool ComputeStencilBoundaryGrid(void);
  bool AssembleOperator(void);
  int order_accuracy_;
  int dim_null_;
```

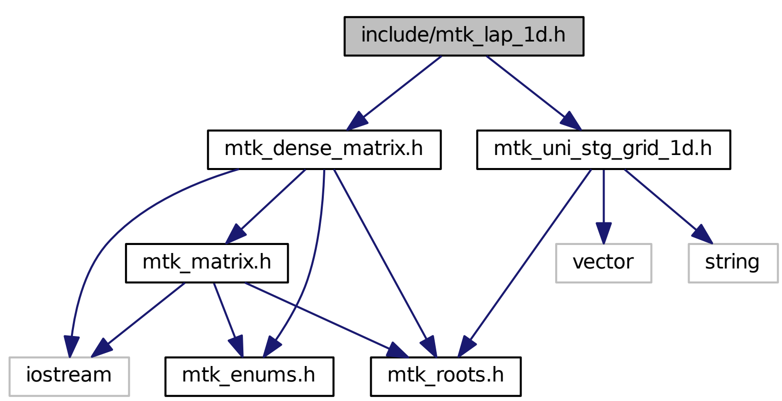
```
  int num_bndy_approxs_;
  int num_bndy_coeffs_;
  int gradient_length_;

  int minrow_;
  int row_;

  mtk::DenseMatrix rat_basis_null_space_;

  Real *coeffs_interior_;
  Real *prem_apps_;
  Real *weights_crs_;
  Real *weights_cbs_;
  Real *mim_bndy_;
  Real *gradient_;

  Real mimetic_threshold_;
};
}
#endif  // End of: MTK_INCLUDE_GRAD_1D_H_
```

17.19 include/mtk_lap_1d.h File Reference

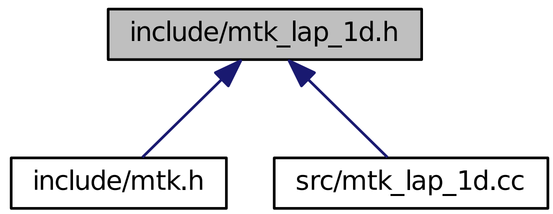
Includes the definition of the class Lap1D.

```
#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.

17.19.1 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.

17.20 mtk_lap_1d.h

```
/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

2. Redistributions of source code must be done through direct
downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
```

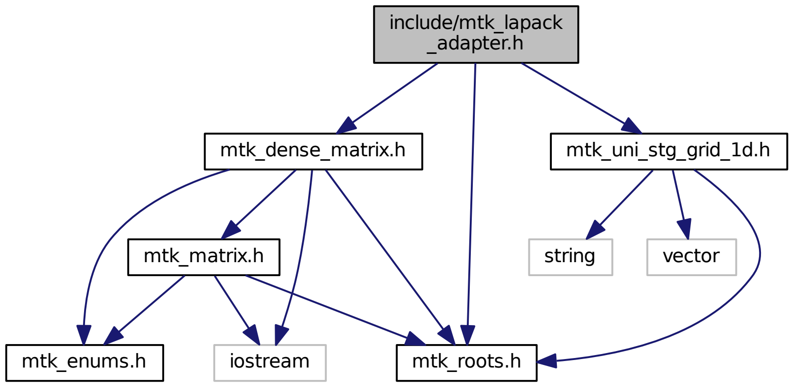
```

3. Redistributions of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#ifndef MTK_INCLUDE_LAP_1D_H_
#define MTK_INCLUDE_LAP_1D_H_

#include "mtk_dense_matrix.h"

#include "mtk_uni_stg_grid_1d.h"

namespace mtk {

class Lap1D {
 public:
  friend std::ostream& operator <<(std::ostream& stream, Lap1D &in);

  Lap1D();

  Lap1D(const Lap1D &lap);

  ~Lap1D();

  bool ConstructLap1D(int order_accuracy = kDefaultOrderAccuracy,
                      Real mimetic_threshold = kDefaultMimeticThreshold);

  DenseMatrix ReturnAsDenseMatrix(const
UniStgGrid1D &grid);

  mtk::Real* Data(const UniStgGrid1D &grid);

 private:
  int order_accuracy_;
  int laplacian_length_;

  Real *laplacian_;

  Real mimetic_threshold_;
};
}
#endif  // End of: MTK_INCLUDE_LAP_1D_H_
```

17.21 include/mtk_lapack_adapter.h File Reference

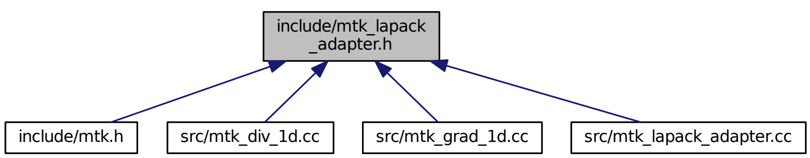
Adapter class for the LAPACK API.

```
#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.

17.21.1 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** 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.

17.22 mtk_lapack_adapter.h

```
/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#ifndef MTK_INCLUDE_LAPACK_ADAPTER_H_
#define MTK_INCLUDE_LAPACK_ADAPTER_H_

#include "mtk_roots.h"
#include "mtk_dense_matrix.h"
#include "mtk_uni_stg_grid_1d.h"

namespace mtk {

```

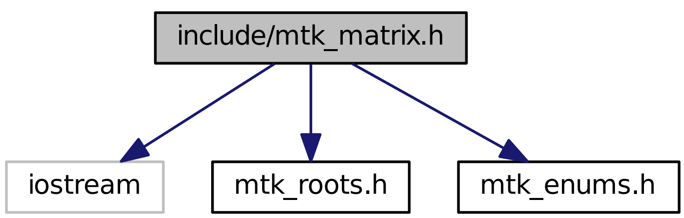
```
class LAPACKAdapter {
 public:
  static int SolveDenseSystem(mtk::DenseMatrix &mm,
                              mtk::Real *rhs);

  static int SolveDenseSystem(mtk::DenseMatrix &mm,
                              mtk::DenseMatrix &rr);

  static int SolveDenseSystem(mtk::DenseMatrix &mm,
                              mtk::UniStgGrid1D &rhs);

  static int SolveRectangularDenseSystem(const
     mtk::DenseMatrix &aa,
                                         mtk::Real *ob_,
                                         int ob_ld_);

  static mtk::DenseMatrix QRFactorDenseMatrix(
     DenseMatrix &matrix);
};
}
#endif  // End of: MTK_INCLUDE_LAPACK_ADAPTER_H_
```

17.23 include/mtk_matrix.h File Reference

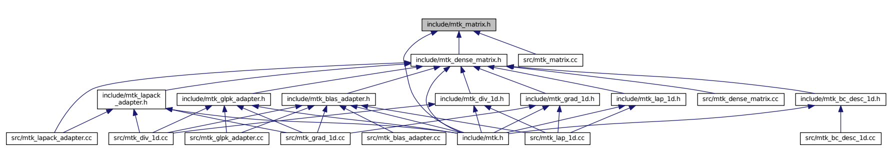
Definition of the representation of a matrix in the MTK.

```
#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.

17.23.1 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.

17.24 mtk_matrix.h

```

/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#ifndef MTK_INCLUDE_MATRIX_H_
```

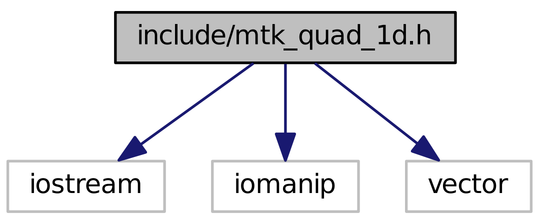
```
#define MTK_INCLUDE_MATRIX_H_
#include <iostream>
#include "mtk_roots.h"
#include "mtk_enums.h"
namespace mtk {
class Matrix {
 public:
  Matrix();
  Matrix(const Matrix &in);
  ~Matrix();
  MatrixStorage storage() const;
  MatrixOrdering ordering() const;
  int num_rows() const;
  int num_cols() const;
  int num_values() const;
  int ld() const;
  int num_zero() const;
  int num_non_zero() const;
  int num_null() const;
  int num_non_null() const;
  int kl() const;
  int ku() const;
  int bandwidth() const;
  Real abs_density() const;
  Real rel_density() const;
  Real abs_sparsity() const;
  Real rel_sparsity() const;
  void set_storage(const MatrixStorage &tt);
  void set_ordering(const MatrixOrdering &oo);
  void set_num_rows(int num_rows);
  void set_num_cols(int num_cols);
  void set_num_zero(int in);
  void set_num_null(int in);
  void IncreaseNumZero();
  void IncreaseNumNull();
 private:
  MatrixStorage storage_;
  MatrixOrdering ordering_;
  int num_rows_;
  int num_cols_;
  int num_values_;
  int ld_;
  int num_zero_;
  int num_non_zero_;
  int num_null_;
  int num_non_null_;
```

```
  int kl_;
  int ku_;
  int bandwidth_;

  Real abs_density_;
  Real rel_density_;
  Real abs_sparsity_;
  Real rel_sparsity_;
};
}
#endif  // End of: MTK_INCLUDE_MATRIX_H_
```

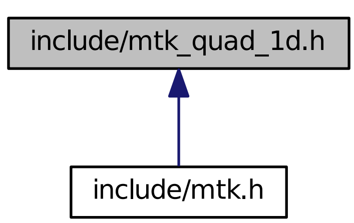
17.25 include/mtk_quad_1d.h File Reference

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.

17.25.1 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.

17.26 mtk_quad_1d.h

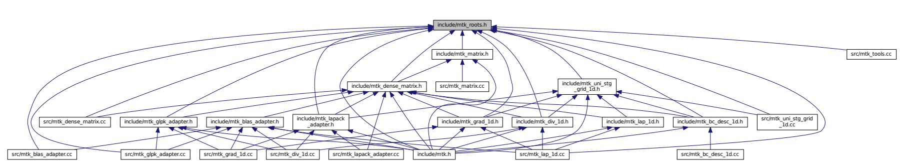
```

/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#ifndef MTK_INCLUDE_QUAD_1D_H_
#define MTK_INCLUDE_QUAD_1D_H_

#include <iostream>
#include <iomanip>

#include <vector>

namespace mtk {

class Quad1D {
 public:
  friend std::ostream& operator <<(std::ostream& stream, Quad1D &in);

  Quad1D();

  Quad1D(const Quad1D &quad);

  ~Quad1D();

  int degree_approximation() const;

  Real *weights() const;

  Real Integrate(Real (*Integrand)(Real xx), UniStgGrid1D grid);

 private:
  int degree_approximation_;

  std::vector<Real> weights_;
};
}
#endif  // End of: MTK_INCLUDE_QUAD_1D_H_
```

17.27 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::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 {1.e-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.

17.27.1 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 mechanism.

Definition in file mtk_roots.h.

17.28 mtk_roots.h

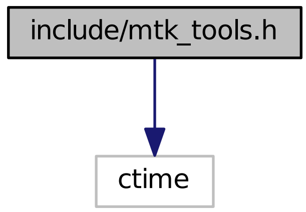
```
/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

```

```
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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.
4. 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.
5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.
6. 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.
*/
#ifndef MTK_INCLUDE_ROOTS_H_
#define MTK_INCLUDE_ROOTS_H_
namespace mtk {
#ifdef MTK_PRECISION_DOUBLE
typedef double Real;
#else
typedef float Real;
#endif
#ifdef MTK_PRECISION_DOUBLE
const double kZero{0.0};
const double kOne{1.0};
#else
const float kZero{0.0f};
const float kOne{1.0f};
#endif
#ifdef MTK_PRECISION_DOUBLE
const double kDefaultTolerance{1e-7};
#else
const float kDefaultTolerance{1e-7f};
#endif
const int kDefaultOrderAccuracy{2};
#ifdef MTK_PRECISION_DOUBLE
const double kDefaultMimeticThreshold{1.e-6};
#else
const float kDefaultMimeticThreshold{1.e-6f};
#endif
const int kCriticalOrderAccuracyDiv{8};
const int kCriticalOrderAccuracyGrad{10};
}
#endif  // End of: MTK_INCLUDE_ROOTS_H_
```

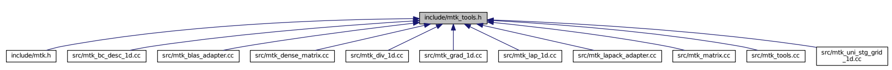
17.29 include/mtk_tools.h File Reference

Tool manager class.

`#include <ctime>`
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.

17.29.1 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.h.

17.30 mtk_tools.h

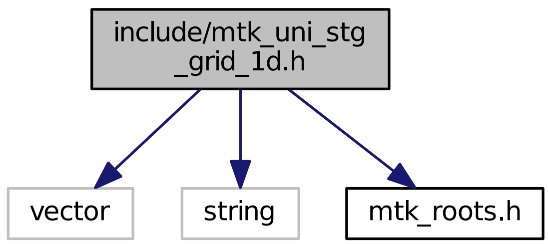
```

/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#ifndef MTK_INCLUDE_TOOLS_H_
#define MTK_INCLUDE_TOOLS_H_

#include <ctime>

namespace mtk {

class Tools {
 public:
  static void Prevent(const bool condition,
                      const char *fname,
                      int lineno,
                      const char *fxname);

  static void BeginTestNo(const int &nn);

  static void EndTestNo(const int &nn);

 private:
  static int test_number_;

  static clock_t begin_time_;
};
}
#endif  // End of: MTK_INCLUDE_TOOLS_H_
```

17.31 include/mtk_uni_stg_grid_1d.h File Reference

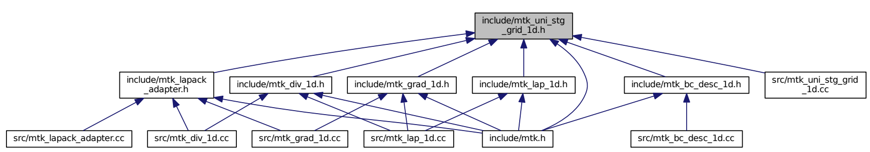
Definition of an 1D uniform staggered grid.

```
#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.

17.31.1 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.

17.32 mtk_uni_stg_grid_1d.h

```

/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#ifndef MTK_INCLUDE_UNI_STG_GRID_1D_H_
#define MTK_INCLUDE_UNI_STG_GRID_1D_H_

#include <vector>
#include <string>

#include "mtk_roots.h"

namespace mtk {

class UniStgGrid1D {
 public:
  friend std::ostream& operator <<(std::ostream& stream, UniStgGrid1D &in);

  UniStgGrid1D();

  UniStgGrid1D(const UniStgGrid1D &grid);

```

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

  ~UniStgGrid1D();

  Real delta_x() const;

  Real *discrete_domain_x();

  Real *discrete_field_u();

  int num_cells_x() const;

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

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

  bool WriteToFile(std::string filename,
                   std::string space_name,
                   std::string field_name);

 private:
  FieldNature nature_;

  std::vector<Real> discrete_domain_x_;
  std::vector<Real> discrete_field_u_;

  Real west_bndy_x_;
  Real east_bndy_x_;
  Real num_cells_x_;
  Real delta_x_;
};
}
#endif  // End of: MTK_INCLUDE_UNI_STG_GRID_1D_H_
```

17.33 Makefile.inc File Reference

17.34 Makefile.inc

```
# Makefile setup file for MTK.

SHELL := /bin/bash

# Please set the following variables up:

#   2. Absolute path to base directory of the MTK... where is the MTK?
#   ___________________________________________________________________

BASE = /home/ejspeiro/Dropbox/MTK

#   2. The machine (platform) identifier and required precision.
#   ___________________________________________________________________

# Options are:
# - LINUX: A LINUX box installation.
# - OSX: Uses OS X optimized solvers.

PLAT = LINUX

# Options are:
# - SINGLE: Use 4 B floating point numbers.
# - DOUBLE: Use 8 B floating point numbers.

PRECISION = DOUBLE

#   3. Optimized solvers and operations by means of ATLAS in Linux?
#   ___________________________________________________________________

# If you have selected OSX in step 1, then you don't need to worry about this.

# Options are ON xor OFF:

ATL_OPT = OFF
```

```

#   4. Paths to dependencies (header files for compiling).
#   __________________________________________________________________

# GLPK include path (soon to go):

GLPK_INC = $(HOME)/Libraries/glpk-4.55/include

# Linux: If ATLAS optimization is ON, users should only provide the path to
# ATLAS:

ATLAS_INC = $(HOME)/Libraries/ATLAS_3.8.4-CORE/include

# OS X: Do nothing.

#   5. Paths to dependencies (archive files for (static) linking).
#   __________________________________________________________________

# GLPK linking path (soon to go):

GLPK_LIB = $(HOME)/Libraries/glpk-4.55/lib/libglpk.a

# If optimization is OFF, then provide the paths for:

BLAS_LIB = $(HOME)/Libraries/BLAS/libblas.a
LAPACK_LIB = $(HOME)/Libraries/lapack-3.4.1/liblapack.a

# WARNING: Vendor libraries should be used whenever they are available.

# However, if optimization is ON, please provide the path the ATLAS' archive:

ATLAS_LIB   = $(HOME)/Libraries/ATLAS_3.8.4-CORE/ATLAS_3.8.4-BUILD-Citadel/lib

#   6. Compiler and its flags.
#   __________________________________________________________________

CC = g++

# Debug Level. Options are:
# 0. NO debug at all NOR any run-time checks... be cautious!
# 1. Verbose (execution messages) AND run-time checks.
# 2. Level 1 plus intermediate scalar-valued results.
# 3. Level 2 plus intermediate array-valued results.

DEBUG_LEVEL = 3

# Flags recommended for release code:

CCFLAGS = -Wall -O2

# Flags recommended for debugging code:

CCFLAGS = -Wall -g

#   7. Archiver, its flags, and ranlib:
#   __________________________________________________________________

ARCH      = ar
ARCHFLAGS = cr

# If your system does not have "ranlib" then set: "RANLIB = echo":

RANLIB = echo

# But, if possible:

RANLIB = ranlib

#   8. Valgrind's memcheck options:
#   __________________________________________________________________

MEMCHECK_OPTS = -v --tool=memcheck --leak-check=full --show-leak-kinds=all \
--track-origins=yes --freelist-vol=20000000

# Done!

#   __________________________________________________________________
#   __________________________________________________________________
#   __________________________________________________________________

#   MTK-related.
```

```
#   ___________________________________________________________________

SRC       = $(BASE)/src
INCLUDE   = $(BASE)/include
LIB       = $(BASE)/lib
MTK_LIB   = $(LIB)/libmtk.a
TESTS     = $(BASE)/tests
EXAMPLES  = $(BASE)/examples

#   Compiling-related.
#   ___________________________________________________________________

CCFLAGS += -std=c++11 -fPIC -DMTK_DEBUG_LEVEL=$(DEBUG_LEVEL) -I$(INCLUDE)  -c

ifeq ($(PRECISION),DOUBLE)
  CCFLAGS += -DMTK_PRECISION_DOUBLE
else
  CCFLAGS += -DMTK_PRECISION_SINGLE
endif

# Only the GLPK is included because the other dependencies are coded in Fortran.

ifeq ($(ATL_OPT),ON)
  CCFLAGS  += -I$(GLPK_INC) $(ATLAS_INC)
else
  CCFLAGS  += -I$(GLPK_INC)
endif

#   Linking-related.
#   ___________________________________________________________________

NOOPT_LIBS  = $(LAPACK_LIB) $(BLAS_LIB) -lm $(GLPK_LIB) -lstdc++

OPT_LIBS    = -L$(ATLAS_LIB) -latlas -llapack -lblas -lm -latlas -lstdc++

ifeq ($(PLAT),OSX)
  LINKER  = g++
  LINKER  += -framework Accelerate $(GLPK_LIB) $(MTK_LIB)
else
  ifeq ($(ATL_OPT),ON)
    LINKER  = g++
    LIBS = $(MTK_LIB)
    LIBS += $(OPT_LIBS)
  else
    LINKER  = gfortran
    LIBS = $(MTK_LIB)
    LIBS += $(NOOPT_LIBS)
  endif
endif

#   Documentation-related.
#   ___________________________________________________________________

DOCGEN      = doxygen
DOCFILENAME = doc_config.dxcf
DOC         = $(BASE)/doc
DOCFILE     = $(BASE)/$(DOCFILENAME)
```

17.35 README.md File Reference

17.36 README.md

```
# 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 discretization** methods for the
```

```
numerical solution of ordinary and partial differential equations.
    ___________________________________________________________________

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

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

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

For OS X:

There are no dependences for OS X.
    ____

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:
- make: builds only the library and the examples.
- all: builds the library, the examples and the documentation.
- mtklib: builds the library, i.e. generates the archive files.
- tests: generates the tests.
- examples: generates the examples.
- gendoc: generates the documentation for the library.
- checkheaders: checks syntax of the header files.

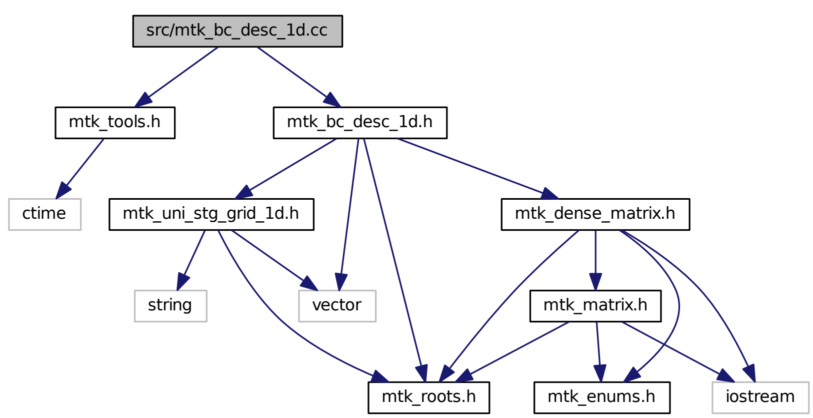
- clean: cleans ALL the generated files.
- cleanlib: cleans the generated archive and object files.
- cleantests: cleans the generated tests executables.
- cleanexamples: cleans the generated examples executables.
-----
```

PART 2. BUILD THE LIBRARY.

```
$ make
```

If successful you'll read (before building the tests and 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: When will the other flavors be ready?
A: Soon! I'm working on getting help on developing those.

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)](http://www.csrc.sdsu.edu/)
at [San Diego State University (SDSU)](http://www.sdsu.edu/).

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:

- **Eduardo J. Sanchez, Ph.D. - esanchez at mail dot sdsu dot edu** - @ejspeiro
- Jose E. Castillo, Ph.D. - jcastillo at mail dot sdsu dot edu
- Guillermo F. Miranda, Ph.D. - unigrav at hotmail dot com
- Christopher P. Paolini, Ph.D. - paolini at engineering dot sdsu dot edu
- Angel Boada.
- Johnny Corbino.
- 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!
```

17.37 src/mtk_bc_desc_1d.cc File Reference

```
#include "mtk_tools.h"
#include "mtk_bc_desc_1d.h"
```

Include dependency graph for mtk_bc_desc_1d.cc:



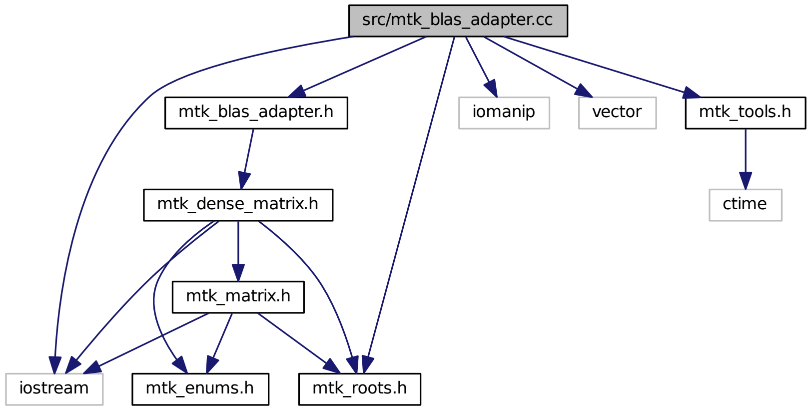
17.38 mtk_bc_desc_1d.cc

```
#include "mtk_tools.h"

#include "mtk_bc_desc_1d.h"

void mtk::BCDesc1D::ImposeOnOperator(
mtk::DenseMatrix &matrix,
                                     const std::vector<mtk::Real> &west,
                                     const std::vector<mtk::Real> &east) {

  mtk::Tools::Prevent(matrix.num_rows() == 0, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent(west.size() > (unsigned int) matrix.
num_cols(),
                      __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent(east.size() > (unsigned int) matrix.
num_cols(),
                      __FILE__, __LINE__, __func__);


  for (unsigned int ii = 0; ii < west.size(); ++ii) {
    matrix.SetValue(0, ii, west[ii]);
  }


  for (unsigned int ii = 0; ii < east.size(); ++ii) {
    matrix.SetValue(matrix.num_rows() - 1,
                    matrix.num_cols() - 1 - ii,
                    east[ii]);
  }
}

void mtk::BCDesc1D::ImposeOnGrid(mtk::UniStgGrid1D &grid,
                                 const mtk::Real &omega,
                                 const mtk::Real &epsilon) {

  mtk::Tools::Prevent(grid.num_cells_x() == 0, __FILE__, __LINE__, __func__);


  grid.discrete_field_u()[0] = omega;


  grid.discrete_field_u()[grid.num_cells_x() + 2 - 1] = epsilon;
}
```

17.39 src/mtk_blas_adapter.cc File Reference

```
#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)

17.40 mtk_blas_adapter.cc

```

/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.
```

```
5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#include <iostream>
#include <iomanip>

#include <vector>

#include "mtk_roots.h"
#include "mtk_tools.h"
#include "mtk_blas_adapter.h"

namespace mtk {

extern "C" {

#ifdef MTK_PRECISION_DOUBLE

double dnrm2_(int *n, double *x, int *incx);
#else

float snrm2_(int *n, float *x, int *incx);
#endif

#ifdef MTK_PRECISION_DOUBLE

void daxpy_(int *n, double *da, double *dx, int *incx, double *dy, int *incy);
#else

void saxpy_(int *n, float *sa, float *sx, int *incx, float *sy, int *incy);
#endif

#ifdef MTK_PRECISION_DOUBLE

void dgemv_(char *trans,
            int *m,
            int *n,
            double *alpha,
            double *a,
            int *lda,
            double *x,
            int *incx,
            double *beta,
            double *y,
            int *incy);
#else

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

```
#endif

#ifdef MTK_PRECISION_DOUBLE

void dgemm_(char *transa,
            char* transb,
            int *m,
            int *n,
            int *k,
            double *alpha,
            double *a,
            int *lda,
            double *b,
            int *ldb,
            double *beta,
            double *c,
            int *ldc);
}
#else

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);
}
#endif
}

mtk::Real mtk::BLASAdapter::RealNRM2(Real *in, int &in_length) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(in_length <= 0, __FILE__, __LINE__, __func__);
  #endif

  int incx{1};  // Increment for the elements of xx. ix >= 0.

  #ifdef MTK_PRECISION_DOUBLE
  return dnrm2_(&in_length, in, &incx);
  #else
  return snrm2_(&in_length, in, &incx);
  #endif
}

void mtk::BLASAdapter::RealAXPY(mtk::Real alpha,
                                mtk::Real *xx,
                                mtk::Real *yy,
                                int &in_length) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(xx == nullptr, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent(yy == nullptr, __FILE__, __LINE__, __func__);
  #endif

  int incx{1};  // Increment for the elements of xx. ix >= 0.

  #ifdef MTK_PRECISION_DOUBLE
  daxpy_(&in_length, &alpha, xx, &incx, yy, &incx);
  #else
  saxpy_(&in_length, &alpha, xx, &incx, yy, &incx);
  #endif
}

mtk::Real mtk::BLASAdapter::RelNorm2Error(
mtk::Real *computed,
                                         mtk::Real *known,
                                         int length) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(computed == nullptr, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent(known == nullptr, __FILE__, __LINE__, __func__);
  #endif

```

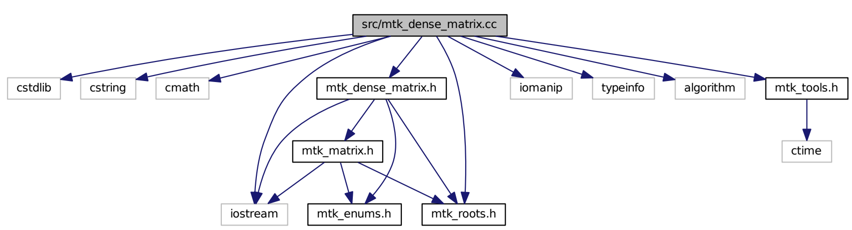
```
  mtk::Real norm_2_computed{mtk::BLASAdapter::RealNRM2(known, length)};

  mtk::Real alpha{-mtk::kOne};

  mtk::BLASAdapter::RealAXPY(alpha, known, computed, length);

  mtk::Real norm_2_difference{mtk::BLASAdapter::RealNRM2(computed,
length)};

  return norm_2_difference/norm_2_computed;
}

void mtk::BLASAdapter::RealDenseMV(mtk::Real &alpha,
                                   mtk::DenseMatrix &aa,
                                   mtk::Real *xx,
                                   mtk::Real &beta,
                                   mtk::Real *yy) {

  // Make sure input matrices are row-major ordered.

  if (aa.matrix_properties().ordering() ==
mtk::COL_MAJOR) {
    aa.OrderRowMajor();
  }

  char transa{'T'}; // State that now, the input WILL be in row-major ordering.

  int mm{aa.num_rows()};                  // Rows of aa.
  int nn{aa.num_cols()};                  // Columns of aa.
  int lda{(aa.matrix_properties()).ld()}; // Leading dimension.
  int incx{1};                            // Increment of values in x.
  int incy{1};                            // Increment of values in y.

  std::swap(mm,nn);
  #ifdef MTK_PRECISION_DOUBLE
  dgemv_(&transa, &mm, &nn, &alpha, aa.data(), &lda,
         xx, &incx, &beta, yy, &incy);
  #else
  sgemv_(&transa, &mm, &nn, &alpha, aa.data(), &lda,
        xx, &incx, &beta, yy, &incy);
  #endif
  std::swap(mm,nn);
}

mtk::DenseMatrix mtk::BLASAdapter::RealDenseMM(
mtk::DenseMatrix &aa,
                                               mtk::DenseMatrix &bb) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(aa.num_cols() != bb.num_rows(),
                      __FILE__, __LINE__, __func__);
  #endif

  // Make sure input matrices are row-major ordered.

  if (aa.matrix_properties().ordering() ==
mtk::COL_MAJOR) {
    aa.OrderRowMajor();
  }
  if (bb.matrix_properties().ordering() ==
mtk::COL_MAJOR) {
    bb.OrderRowMajor();
  }

  char ta{'T'}; // State that input matrix aa is in row-wise ordering.
  char tb{'T'}; // State that input matrix bb is in row-wise ordering.

  int mm{aa.num_rows()};  // Rows of aa and rows of cc.
  int nn{bb.num_cols()};  // Cols of bb and cols of cc.
  int kk{aa.num_cols()};  // Cols of aa and rows of bb.

  int cc_num_rows{mm};  // Rows of cc.
  int cc_num_cols{nn};  // Columns of cc.

  int lda{std::max(1,kk)};  // Leading dimension of the aa matrix.
  int ldb{std::max(1,nn)};  // Leading dimension of the bb matrix.
  int ldc{std::max(1,mm)};  // Leading dimension of the cc matrix.

  mtk::Real alpha{1.0}; // First scalar coefficient.
  mtk::Real beta{0.0};  // Second scalar coefficient.

```

```
  mtk::DenseMatrix cc_col_maj_ord(cc_num_rows,cc_num_cols); // Output matrix.

  cc_col_maj_ord.SetOrdering(mtk::COL_MAJOR);

  #ifdef MTK_PRECISION_DOUBLE
  dgemm_(&ta, &tb, &mm, &nn, &kk, &alpha, aa.data(), &lda,
         bb.data(), &ldb, &beta, cc_col_maj_ord.data(), &ldc);
  #else
  sgemm_(&ta, &tb, &mm, &nn, &kk, &alpha, aa.data(), &lda,
         bb.data(), &ldb, &beta, cc_col_maj_ord.data(), &ldc);
  #endif

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "cc_col_maj_ord =" << std::endl;
  std::cout << cc_col_maj_ord << std::endl;
  #endif

  cc_col_maj_ord.OrderRowMajor();

  return cc_col_maj_ord;
}
```

17.41 src/mtk_dense_matrix.cc File Reference

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

#include <cstdlib>
#include <cstring>
#include <cmath>
#include <iostream>
#include <iomanip>
#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)

17.41.1 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

Definition in file mtk_dense_matrix.cc.

17.42 mtk_dense_matrix.cc

```

/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#include <cstdlib>
#include <cstring>
#include <cmath>

#include <iostream>
#include <iomanip>
#include <typeinfo>

#include <algorithm>

#include "mtk_roots.h"
```

```
#include "mtk_dense_matrix.h"
#include "mtk_tools.h"

namespace mtk {

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

  int mm{in.matrix_properties_.num_rows()};  // Auxiliary.
  int nn{in.matrix_properties_.num_cols()};  // Auxiliary.

  if (in.matrix_properties_.ordering() == mtk::COL_MAJOR) {
    std::swap(mm, nn);
  }
  for (auto ii = 0; ii < mm; ii++) {
    for (auto jj = 0; jj < nn; jj++) {
      mtk::Real value = in.data_[ii*nn + jj];
      stream << std::setw(13) << value;
    }
    stream << std::endl;
  }
  if (in.matrix_properties_.ordering() == mtk::COL_MAJOR) {
    std::swap(mm, nn);
  }
  return stream;
}
}

mtk::DenseMatrix& mtk::DenseMatrix::operator =(const mtk::DenseMatrix &in) {

  if(this == &in) {
    return *this;
  }

  matrix_properties_.set_storage(in.matrix_properties_.storage());

  matrix_properties_.set_ordering(in.matrix_properties_.ordering());

  auto aux = in.matrix_properties_.num_rows();
  matrix_properties_.set_num_rows(aux);

  aux = in.matrix_properties().num_cols();
  matrix_properties_.set_num_cols(aux);

  aux = in.matrix_properties().num_zero();
  matrix_properties_.set_num_zero(aux);

  aux = in.matrix_properties().num_null();
  matrix_properties_.set_num_null(aux);

  auto num_rows = matrix_properties_.num_rows();
  auto num_cols = matrix_properties_.num_cols();

  delete [] data_;

  try {
    data_ = new mtk::Real[num_rows*num_cols];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(data_, mtk::kZero, sizeof(data_[0])*num_rows*num_cols);

  std::copy(in.data_, in.data_ + num_rows*num_cols, data_);

  return *this;
}

mtk::DenseMatrix::DenseMatrix(): data_(nullptr) {

  matrix_properties_.set_storage(mtk::DENSE);
  matrix_properties_.set_ordering(mtk::ROW_MAJOR);
}

mtk::DenseMatrix::DenseMatrix(const
```

```
mtk::DenseMatrix &in) {

  matrix_properties_.set_storage(in.matrix_properties_.storage());

  matrix_properties_.set_ordering(in.matrix_properties_.
ordering());

  auto aux = in.matrix_properties_.num_rows();
  matrix_properties_.set_num_rows(aux);

  aux = in.matrix_properties().num_cols();
  matrix_properties_.set_num_cols(aux);

  aux = in.matrix_properties().num_zero();
  matrix_properties_.set_num_zero(aux);

  aux = in.matrix_properties().num_null();
  matrix_properties_.set_num_null(aux);

  auto num_rows = in.matrix_properties_.num_rows();
  auto num_cols = in.matrix_properties_.num_cols();

  try {
    data_ = new mtk::Real[num_rows*num_cols];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(data_, mtk::kZero, sizeof(data_[0])*num_rows*num_cols);

  std::copy(in.data_,in.data_ + num_rows*num_cols,data_);
}

mtk::DenseMatrix::DenseMatrix(const int &num_rows, const int &num_cols) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(num_rows < 1, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent(num_cols < 1, __FILE__, __LINE__, __func__);
  #endif

  matrix_properties_.set_storage(mtk::DENSE);
  matrix_properties_.set_ordering(mtk::ROW_MAJOR);
  matrix_properties_.set_num_rows(num_rows);
  matrix_properties_.set_num_cols(num_cols);

  try {
    data_ = new mtk::Real[num_rows*num_cols];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(data_, mtk::kZero, sizeof(data_[0])*num_rows*num_cols);
}

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

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(rank < 1, __FILE__, __LINE__, __func__);
  #endif

  int aux{};  // Used to control the padding.

  if (padded) {
    aux = 1;
  }

  matrix_properties_.set_storage(mtk::DENSE);
  matrix_properties_.set_ordering(mtk::ROW_MAJOR);
  matrix_properties_.set_num_rows(aux + rank + aux);
  matrix_properties_.set_num_cols(rank);

  try {
    data_ = new mtk::Real[matrix_properties_.num_values()];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
```

```
  }
  memset(data_,
         mtk::kZero,
         sizeof(data_[0])*(matrix_properties_.num_values()));

  for (auto ii =0 ; ii < matrix_properties_.num_rows(); ++ii) {
    for (auto jj = 0; jj < matrix_properties_.num_cols(); ++jj) {
      data_[ii*matrix_properties_.num_cols() + jj] =
        (ii == jj + aux)? mtk::kOne: mtk::kZero;
    }
  }
}

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

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(gen == nullptr, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent(gen_length < 1, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent(pro_length < 1, __FILE__, __LINE__, __func__);
  #endif

  matrix_properties_.set_storage(mtk::DENSE);
  matrix_properties_.set_ordering(mtk::ROW_MAJOR);
  if (!transpose) {
    matrix_properties_.set_num_rows(gen_length);
    matrix_properties_.set_num_cols(pro_length);
  } else {
    matrix_properties_.set_num_rows(pro_length);
    matrix_properties_.set_num_cols(gen_length);
  }

  int rr = matrix_properties_.num_rows(); // Used to construct this matrix.
  int cc = matrix_properties_.num_cols(); // Used to construct this matrix.

  try {
    data_ = new mtk::Real[rr*cc];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(data_, mtk::kZero, sizeof(data_[0])*rr*cc);

  if (!transpose) {
    for (auto ii = 0; ii < rr; ii++) {
      for (auto jj = 0; jj < cc; jj++) {
        data_[ii*cc + jj] = pow(gen[ii], (double) jj);
      }
    }
  } else {
    for (auto ii = 0; ii < rr; ii++) {
      for (auto jj = 0; jj < cc; jj++) {
        data_[ii*cc + jj] = pow(gen[jj], (double) ii);
      }
    }
  }
}

mtk::DenseMatrix::~DenseMatrix() {

  delete[] data_;
  data_ = nullptr;
}

mtk::Matrix mtk::DenseMatrix::matrix_properties() const {

  return matrix_properties_;
}

void mtk::DenseMatrix::SetOrdering(
mtk::MatrixOrdering oo) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(!(oo == mtk::ROW_MAJOR || oo ==
mtk::COL_MAJOR),
                      __FILE__, __LINE__, __func__);
  #endif

```

```
  matrix_properties_.set_ordering(oo);
}

int mtk::DenseMatrix::num_rows() const {

  return matrix_properties_.num_rows();
}

int mtk::DenseMatrix::num_cols() const {

  return matrix_properties_.num_cols();
}

mtk::Real* mtk::DenseMatrix::data() const {

  return data_;
}

mtk::Real mtk::DenseMatrix::GetValue(
    const int &rr,
    const int &cc) const {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(rr < 0, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent(cc < 0, __FILE__, __LINE__, __func__);
  #endif

  return data_[rr*matrix_properties_.num_cols() + cc];
}

void  mtk::DenseMatrix::SetValue(
    const int &rr,
    const int &cc,
    const mtk::Real &val) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(rr < 0, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent(cc < 0, __FILE__, __LINE__, __func__);
  #endif

  data_[rr*matrix_properties_.num_cols() + cc] = val;
}

void mtk::DenseMatrix::Transpose() {


  mtk::Real *data_transposed{}; // Buffer.

  int rr = matrix_properties_.num_rows(); // Used to construct this matrix.
  int cc = matrix_properties_.num_cols(); // Used to construct this matrix.

  try {
    data_transposed = new mtk::Real[rr*cc];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(data_transposed,
         mtk::kZero,
         sizeof(data_transposed[0])*rr*cc);

  // Assign the values to their transposed position.
  for (auto ii = 0; ii < rr; ++ii) {
    for (auto jj = 0; jj < cc; ++jj) {
      data_transposed[jj*rr + ii] = data_[ii*cc + jj];
    }
  }

  // Swap pointers.
  auto tmp = data_; // Temporal holder.
  data_ = data_transposed;
  delete [] tmp;
  tmp = nullptr;

  matrix_properties_.set_num_rows(cc);
  matrix_properties_.set_num_cols(rr);
}

void mtk::DenseMatrix::OrderRowMajor() {

```

```
  if (matrix_properties_.ordering() == mtk::COL_MAJOR) {


    mtk::Real *data_transposed{}; // Buffer.

    int rr = matrix_properties_.num_rows(); // Used to construct this matrix.
    int cc = matrix_properties_.num_cols(); // Used to construct this matrix.

    try {
      data_transposed = new mtk::Real[rr*cc];
    } catch (std::bad_alloc &memory_allocation_exception) {
      std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
        std::endl;
      std::cerr << memory_allocation_exception.what() << std::endl;
    }
    memset(data_transposed,
           mtk::kZero,
           sizeof(data_transposed[0])*rr*cc);

    // Assign the values to their transposed position.
    std::swap(rr, cc);
    for (auto ii = 0; ii < rr; ++ii) {
      for (auto jj = 0; jj < cc; ++jj) {
        data_transposed[jj*rr + ii] = data_[ii*cc + jj];
      }
    }
    std::swap(rr, cc);

    // Swap pointers.
    auto tmp = data_; // Temporal holder.
    data_ = data_transposed;
    delete [] tmp;
    tmp = nullptr;

    matrix_properties_.set_ordering(mtk::ROW_MAJOR);
  }
}

void mtk::DenseMatrix::OrderColMajor() {

  if (matrix_properties_.ordering() == ROW_MAJOR) {


    mtk::Real *data_transposed{}; // Buffer.

    int rr = matrix_properties_.num_rows(); // Used to construct this matrix.
    int cc = matrix_properties_.num_cols(); // Used to construct this matrix.

    try {
      data_transposed = new mtk::Real[rr*cc];
    } catch (std::bad_alloc &memory_allocation_exception) {
      std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
        std::endl;
      std::cerr << memory_allocation_exception.what() << std::endl;
    }
    memset(data_transposed,
           mtk::kZero,
           sizeof(data_transposed[0])*rr*cc);

    // Assign the values to their transposed position.
    for (auto ii = 0; ii < rr; ++ii) {
      for (auto jj = 0; jj < cc; ++jj) {
        data_transposed[jj*rr + ii] = data_[ii*cc + jj];
      }
    }

    // Swap pointers.
    auto tmp = data_; // Temporal holder.
    data_ = data_transposed;
    delete [] tmp;
    tmp = nullptr;

    matrix_properties_.set_ordering(mtk::COL_MAJOR);
  }
}

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

  int row_offset{}; // Offset for rows.
```

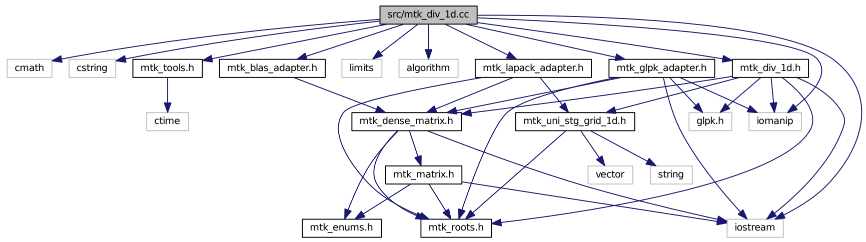
```
  int col_offset{}; // Offset for rows.

  mtk::Real aa_factor{}; // Used in computation.

  // Auxiliary variables:
  auto aux1 = aa.matrix_properties_.num_rows()*bb.matrix_properties_.num_rows();
  auto aux2 = aa.matrix_properties_.num_cols()*bb.matrix_properties_.num_cols();

  mtk::DenseMatrix output(aux1,aux2); // Output matrix.

  int kk_num_cols{output.matrix_properties_.num_cols()}; // Aux.

  auto mm = aa.matrix_properties_.num_rows(); // Rows of aa.
  auto nn = aa.matrix_properties_.num_cols(); // Cols of aa.
  auto pp = bb.matrix_properties_.num_rows(); // Rows of bb.
  auto qq = bb.matrix_properties_.num_cols(); // Cols of bb.

  for (auto ii = 0; ii < mm; ++ii) {
    row_offset = ii*pp;
    for (auto jj = 0; jj < nn; ++jj) {
      col_offset = jj*qq;
      aa_factor = aa.data_[ii*nn + jj];
      for (auto ll = 0; ll < pp; ++ll) {
        for (auto oo = 0; oo < qq; ++oo) {
          auto index = (ll + row_offset)*kk_num_cols + (oo + col_offset);
          output.data_[index] = aa_factor*bb.data_[ll*qq + oo];
        }
      }
    }
  }

  output.matrix_properties_.set_storage(mtk::DENSE);
  output.matrix_properties_.set_ordering(mtk::ROW_MAJOR);

  return output;
}

```

17.43 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)

17.43.1 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.

17.44 mtk_div_1d.cc

```

/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#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"

namespace mtk {

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


  stream << "divergence_[0] = " << std::setw(9) << in.divergence_[0] <<
    std::endl;


  stream << "divergence_[1:" << in.order_accuracy_ << "] = ";
  for (auto ii = 1; ii <= in.order_accuracy_; ++ii) {
    stream << std::setw(9) << in.divergence_[ii] << " ";
  }
  stream << std::endl;

  if (in.order_accuracy_ > 2) {


    stream << "divergence_[" << in.order_accuracy_ + 1 << ":" <<
      2*in.order_accuracy_ << "] = ";
    for (auto ii = in.order_accuracy_ + 1; ii <= 2*in.order_accuracy_; ++ii) {
      stream << std::setw(9) << in.divergence_[ii] << " ";
    }
    stream << std::endl;


    auto offset = (2*in.order_accuracy_ + 1);
    int mm{};
    for (auto ii = 0; ii < in.dim_null_; ++ii) {
      stream << "divergence_[" << offset + mm << ":" <<
        offset + mm + in.num_bndy_coeffs_ - 1 << "] = ";
      for (auto jj = 0; jj < in.num_bndy_coeffs_; ++jj) {
```

```
        auto value = in.divergence_[offset + mm];
        stream << std::setw(9) << value << " ";
        ++mm;
      }
      stream << std::endl;
    }
  }

  return stream;
}
}

mtk::Div1D::Div1D():
  order_accuracy_(mtk::kDefaultOrderAccuracy),
  dim_null_(),
  num_bndy_coeffs_(),
  divergence_length_(),
  minrow_(),
  row_(),
  coeffs_interior_(),
  prem_apps_(),
  weights_crs_(),
  weights_cbs_(),
  mim_bndy_(),
  divergence_(),
  mimetic_threshold_(mtk::kDefaultMimeticThreshold) {}

mtk::Div1D::Div1D(const Div1D &div):
  order_accuracy_(div.order_accuracy_),
  dim_null_(div.dim_null_),
  num_bndy_coeffs_(div.num_bndy_coeffs_),
  divergence_length_(div.divergence_length_),
  minrow_(div.minrow_),
  row_(div.row_),
  coeffs_interior_(div.coeffs_interior_),
  prem_apps_(div.prem_apps_),
  weights_crs_(div.weights_crs_),
  weights_cbs_(div.weights_cbs_),
  mim_bndy_(div.mim_bndy_),
  divergence_(div.divergence_),
  mimetic_threshold_(div.mimetic_threshold_) {}

mtk::Div1D::~Div1D() {

  delete[] coeffs_interior_;
  coeffs_interior_ = nullptr;

  delete[] prem_apps_;
  prem_apps_ = nullptr;

  delete[] weights_crs_;
  weights_crs_ = nullptr;

  delete[] weights_cbs_;
  weights_cbs_ = nullptr;

  delete[] mim_bndy_;
  mim_bndy_ = nullptr;

  delete[] divergence_;
  divergence_ = nullptr;
}

bool mtk::Div1D::ConstructDiv1D(int order_accuracy,
                                mtk::Real mimetic_threshold) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(order_accuracy < 2, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent((order_accuracy%2) != 0, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent(mimetic_threshold <= mtk::kZero,
                      __FILE__, __LINE__, __func__);

  if (order_accuracy >= mtk::kCriticalOrderAccuracyDiv) {
    std::cout << "WARNING: Numerical accuracy is critical." << std::endl;
  }

  std::cout << "order_accuracy_ = " << order_accuracy << std::endl;
  std::cout << "mimetic_threshold_ = " << mimetic_threshold << std::endl;
  #endif

  order_accuracy_ = order_accuracy;
```

```
mimetic_threshold_ = mimetic_threshold;


bool abort_construction = ComputeStencilInteriorGrid();

#if MTK_DEBUG_LEVEL > 0
if (!abort_construction) {
  std::cerr << "Could NOT complete stage 1." << std::endl;
  std::cerr << "Exiting..." << std::endl;
  return false;
}
#endif

// At this point, we already have the values for the interior stencil stored
// in the coeffs_interior_ array.

// It is noteworthy, that the 2nd-order-accurate divergence operator has NO
// approximation at the boundary, thus it has no weights. For this case, the
// dimension of the null-space of the Vandermonde matrices used to compute the
// approximating coefficients at the boundary is 0. Ergo, we compute this
// number first and then decide if we must compute anything at the boundary.

dim_null_ = order_accuracy_/2 - 1;

if (dim_null_ > 0) {

  #ifdef MTK_PRECISION_DOUBLE
  num_bndy_coeffs_ = (int) (3.0*((mtk::Real) order_accuracy_)/2.0);
  #else
  num_bndy_coeffs_ = (int) (3.0f*((mtk::Real) order_accuracy_)/2.0f);
  #endif


  // For this we will follow recommendations given in:
  //
  // http://icl.cs.utk.edu/lapack-forum/viewtopic.php?f=5&t=4506
  //
  // We will compute the QR Factorization of the transpose, as in the
  // following (MATLAB) pseudo-code:
  //
  // [Q,R] = qr(V'); % Full QR as defined in
  // % http://www.stanford.edu/class/ee263/notes/qr_matlab.pdf
  //
  // null-space = Q(:, last (order_accuracy_/2 - 1) columns of Q );
  //
  // However, given the nature of the Vandermonde matrices we've just
  // computed, they all posses the same null-space. Therefore, we impose the
  // convention of computing the null-space of the first Vandermonde matrix
  // (west boundary).

  abort_construction = ComputeRationalBasisNullSpace();

  #if MTK_DEBUG_LEVEL > 0
  if (!abort_construction) {
    std::cerr << "Could NOT complete stage 2.1." << std::endl;
    std::cerr << "Exiting..." << std::endl;
    return false;
  }
  #endif


  abort_construction = ComputePreliminaryApproximations();

  #if MTK_DEBUG_LEVEL > 0
  if (!abort_construction) {
    std::cerr << "Could NOT complete stage 2.2." << std::endl;
    std::cerr << "Exiting..." << std::endl;
    return false;
  }
  #endif


  abort_construction = ComputeWeights();

  #if MTK_DEBUG_LEVEL > 0
  if (!abort_construction) {
    std::cerr << "Could NOT complete stage 2.3." << std::endl;
    std::cerr << "Exiting..." << std::endl;
    return false;
  }
  #endif
```

```
    abort_construction = ComputeStencilBoundaryGrid();

    #if MTK_DEBUG_LEVEL > 0
    if (!abort_construction) {
      std::cerr << "Could NOT complete stage 2.4." << std::endl;
      std::cerr << "Exiting..." << std::endl;
      return false;
    }
    #endif

  } // End of: if (dim_null_ > 0);


  // Once we have the following three collections of data:
  //   (a) the coefficients for the interior,
  //   (b) the coefficients for the boundary (if it applies),
  //   (c) and the weights (if it applies),
  // we will store everything in the output array:

  abort_construction = AssembleOperator();

  #if MTK_DEBUG_LEVEL > 0
  if (!abort_construction) {
    std::cerr << "Could NOT complete stage 3." << std::endl;
    std::cerr << "Exiting..." << std::endl;
    return false;
  }
  #endif

  return true;
}

int mtk::Div1D::num_bndy_coeffs() {

  return num_bndy_coeffs_;
}

mtk::Real *mtk::Div1D::weights_crs() {

  return weights_crs_;
}

mtk::Real *mtk::Div1D::weights_cbs() {

  return weights_cbs_;
}

mtk::DenseMatrix mtk::Div1D::ReturnAsDenseMatrix(const
UniStgGrid1D &grid) {

  int nn{grid.num_cells_x()}; // Number of cells on the grid.

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(order_accuracy_ <= 0, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent(nn < 3*order_accuracy_ - 1, __FILE__, __LINE__, __func__);
  #endif

  mtk::Real inv_delta_x{mtk::kOne/grid.delta_x()};

  int dd_num_rows = nn + 2;
  int dd_num_cols = nn + 1;
  int elements_per_row = num_bndy_coeffs_;
  int num_extra_rows = dim_null_;

  // Output matrix featuring sizes for divergence operators.
  mtk::DenseMatrix out(dd_num_rows, dd_num_cols);


  auto ee_index = 0;
  for (auto ii = 1; ii < num_extra_rows + 1; ii++) {
    auto cc = 0;
    for(auto jj = 0 ; jj < dd_num_rows; jj++) {
      if( cc >= elements_per_row) {
        out.SetValue(ii, jj, mtk::kZero);
      } else {
        out.SetValue(ii,jj, mim_bndy_[ee_index++]*inv_delta_x);
        cc++;
      }
    }
```

```
  }


  for (auto ii = num_extra_rows + 1;
       ii < dd_num_rows - num_extra_rows - 1; ii++) {
    auto jj = ii - num_extra_rows - 1;
    for (auto cc = 0; cc < order_accuracy_; cc++, jj++) {
      out.SetValue(ii, jj, coeffs_interior_[cc]*inv_delta_x);
    }
  }


  ee_index = 0;
  for (auto ii = dd_num_rows - 2; ii >= dd_num_rows - num_extra_rows - 1; ii--) {
    auto cc = 0;
    for (auto jj = dd_num_cols - 1; jj >= 0; jj--) {
      if( cc >= elements_per_row) {
        out.SetValue(ii,jj,0.0);
      } else {
        out.SetValue(ii,jj,-mim_bndy_[ee_index++]*inv_delta_x);
        cc++;
      }
     }
  }

  return out;
}

bool mtk::Div1D::ComputeStencilInteriorGrid() {


  mtk::Real* pp{}; // Spatial coordinates to create interior stencil.

  try {
    pp = new mtk::Real[order_accuracy_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(pp, mtk::kZero, sizeof(pp[0])*order_accuracy_);

  #ifdef MTK_PRECISION_DOUBLE
  pp[0] = 1.0/2.0 - ((mtk::Real) order_accuracy_)/2.0;
  #else
  pp[0] = 1.0f/2.0f - ((mtk::Real) order_accuracy_)/2.0f;
  #endif

  for (auto ii = 1; ii < order_accuracy_; ++ii) {
    pp[ii] = pp[ii - 1] + mtk::kOne;
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "pp =" << std::endl;
  for (auto ii = 0; ii < order_accuracy_; ++ii) {
    std::cout << std::setw(12) << pp[ii];
  }
  std::cout << std::endl << std::endl;
  #endif


  bool transpose{false};

  mtk::DenseMatrix vander_matrix(pp,
                                 order_accuracy_,
                                 order_accuracy_,
                                 transpose);

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "vander_matrix = " << std::endl;
  std::cout << vander_matrix << std::endl;
  #endif


  try {
    coeffs_interior_ = new mtk::Real[order_accuracy_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
```

```
  memset(coeffs_interior_, mtk::kZero, sizeof(coeffs_interior_[0])*order_accuracy_);

  coeffs_interior_[1] = mtk::kOne;

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "oo =" << std::endl;
  for (auto ii = 0; ii < order_accuracy_; ++ii) {
    std::cout << std::setw(12) << coeffs_interior_[ii] << std::endl;
  }
  std::cout << std::endl;
  #endif


  int info{mtk::LAPACKAdapter::SolveDenseSystem(vander_matrix,
                                                coeffs_interior_)};

  #if MTK_DEBUG_LEVEL > 0
  if (!info) {
    std::cout << "System solved! Interior stencil attained!" << std::endl;
    std::cout << std::endl;
  }
  else {
    std::cerr << "Something wrong solving system! info = " << info << std::endl;
    std::cerr << "Exiting..." << std::endl;
    return false;
  }
  #endif

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "coeffs_interior_ =" << std::endl;
  for (auto ii = 0; ii < order_accuracy_; ++ii) {
    std::cout << std::setw(12) << coeffs_interior_[ii];
  }
  std::cout << std::endl << std::endl;
  #endif

  delete [] pp;
  pp = nullptr;

  return true;
}

bool mtk::Div1D::ComputeRationalBasisNullSpace(void) {

  mtk::Real* gg{}; // Generator vector for the first Vandermonde matrix.


  try {
    gg = new mtk::Real[num_bndy_coeffs_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(gg, mtk::kZero, sizeof(gg[0])*num_bndy_coeffs_);

  #ifdef MTK_PRECISION_DOUBLE
  gg[0] = -1.0/2.0;
  #else
  gg[0] = -1.0f/2.0f;
  #endif
  for (auto ii = 1; ii < num_bndy_coeffs_; ++ii) {
    gg[ii] = gg[ii - 1] + mtk::kOne;
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "gg =" << std::endl;
  for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
    std::cout << std::setw(12) << gg[ii];
  }
  std::cout << std::endl << std::endl;
  #endif


  bool tran{true}; // Should I transpose the Vandermonde matrix.

  mtk::DenseMatrix vv_west_t(gg, num_bndy_coeffs_, order_accuracy_ + 1, tran);

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "vv_west_t =" << std::endl;
  std::cout << vv_west_t << std::endl;
```

```
  #endif


  mtk::DenseMatrix qq_t(mtk::LAPACKAdapter::QRFactorDenseMatrix
(vv_west_t));

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "QQ^T = " << std::endl;
  std::cout << qq_t << std::endl;
  #endif


  int KK_num_rows_{num_bndy_coeffs_};
  int KK_num_cols_{dim_null_};

  mtk::DenseMatrix KK(KK_num_rows_, KK_num_cols_);

  for (auto ii = num_bndy_coeffs_ - dim_null_; ii < num_bndy_coeffs_; ++ii) {
    for (auto jj = 0; jj < num_bndy_coeffs_; ++jj) {
      KK.data()[jj*dim_null_ + (ii - (num_bndy_coeffs_ - dim_null_))] =
          qq_t.data()[ii*num_bndy_coeffs_ + jj];
    }
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "KK =" << std::endl;
  std::cout << KK << std::endl;
  std::cout << "KK.num_rows() = " << KK.num_rows() << std::endl;
  std::cout << "KK.num_cols() = " << KK.num_cols() << std::endl;
  std::cout << std::endl;
  #endif


  // Scale thus requesting that the last entries of the attained basis for the
  // null-space, adopt the pattern we require.
  // Essentially we will implement the following MATLAB pseudo-code:
  //  scalers = KK(num_bndy_approxs - (dim_null - 1):num_bndy_approxs,:)\B
  //  SK = KK*scalers
  // where SK is the scaled null-space.

  // In this point, we almost have all the data we need correctly allocated
  // in memory. We will create the matrix II_, and elements we wish to scale in
  // the KK array. Using the concept of the leading dimension, we could just
  // use KK, with the correct leading dimension and that is it. BUT I DO NOT
  // GET how does it work. So I will just create a matrix with the content of
  // this array that we need, solve for the scalers and then scale the
  // whole KK:

  // We will then create memory for that sub-matrix of KK (SUBK).

  mtk::DenseMatrix SUBK(dim_null_,dim_null_);

  for (auto ii = num_bndy_coeffs_ - dim_null_; ii < num_bndy_coeffs_; ++ii) {
    for (auto jj = 0; jj < dim_null_; ++jj) {
      SUBK.data()[(ii - (num_bndy_coeffs_ - dim_null_))*dim_null_ + jj] =
          KK.data()[ii*dim_null_ + jj];
    }
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "SUBK =" << std::endl;
  std::cout << SUBK << std::endl;
  #endif

  SUBK.Transpose();

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "SUBK^T =" << std::endl;
  std::cout << SUBK << std::endl;
  #endif

  bool padded{false};
  tran = false;

  mtk::DenseMatrix II(dim_null_, padded, tran);

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "II =" << std::endl;
  std::cout << II << std::endl;
  #endif

```

```
  // Solve the system to compute the scalers.
  // An example of the system to solve, for k = 8, is:
  //
  // SUBK*scalers = II_ or
  //
  // |  0.386018  -0.0339244   -0.129478 |            | 1 0 0 |
  // | -0.119774   0.0199423   0.0558632 |*scalers = | 0 1 0 |
  // | 0.0155708 -0.00349546 -0.00853182 |            | 0 0 1 |
  //
  // Notice this is a nrhs = 3 system.
  // Noteworthy: we do NOT ACTUALLY ALLOCATE space for the scalers... they
  // will be stored in the created identity matrix.
  // Let us first transpose SUBK (because of LAPACK):

  int info{mtk::LAPACKAdapter::SolveDenseSystem(SUBK, II)};

  #if MTK_DEBUG_LEVEL > 0
  if (!info) {
    std::cout << "System successfully solved!" <<
      std::endl;
  } else {
    std::cerr << "Something went wrong solving system! info = " << info <<
      std::endl;
    std::cerr << "Exiting..." << std::endl;
    return false;
  }
  std::cout << std::endl;
  #endif

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "Computed scalers:" << std::endl;
  std::cout << II << std::endl;
  #endif

  // Multiply the two matrices to attain a scaled basis for null-space.

  rat_basis_null_space_ = mtk::BLASAdapter::RealDenseMM(KK, II);

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "Rational basis for the null-space:" << std::endl;
  std::cout << rat_basis_null_space_ << std::endl;
  #endif

  // At this point, we have a rational basis for the null-space, with the
  // pattern we need! :)

  delete [] gg;
  gg = nullptr;

  return true;
}

bool mtk::Div1D::ComputePreliminaryApproximations(void) {


  mtk::Real *gg{}; // Generator vector for the first approximation.

  try {
    gg = new mtk::Real[num_bndy_coeffs_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(gg, mtk::kZero, sizeof(gg[0])*num_bndy_coeffs_);

  #ifdef MTK_PRECISION_DOUBLE
  gg[0] = -1.0/2.0;
  #else
  gg[0] = -1.0f/2.0f;
  #endif
  for (auto ii = 1; ii < num_bndy_coeffs_; ++ii) {
    gg[ii] = gg[ii - 1] + mtk::kOne;
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "gg0 =" << std::endl;
  for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
    std::cout << std::setw(12) << gg[ii];
  }
  std::cout << std::endl << std::endl;
```

```
  #endif

  // Allocate 2D array to store the collection of preliminary approximations.
  try {
    prem_apps_ = new mtk::Real[num_bndy_coeffs_*dim_null_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(prem_apps_,
         mtk::kZero,
         sizeof(prem_apps_[0])*num_bndy_coeffs_*dim_null_);



  for (auto ll = 0; ll < dim_null_; ++ll) {

    // Re-check new generator vector for every iteration except for the first.
    #if MTK_DEBUG_LEVEL > 0
    if (ll > 0) {
      std::cout << "gg" << ll << " =" << std::endl;
      for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
        std::cout << std::setw(12) << gg[ii];
      }
      std::cout << std::endl << std::endl;
    }
    #endif


    bool transpose{false};

    mtk::DenseMatrix AA_(gg,
                         num_bndy_coeffs_, order_accuracy_ + 1,
                         transpose);

    #if MTK_DEBUG_LEVEL > 0
    std::cout << "AA_" << ll << " =" << std::endl;
    std::cout << AA_ << std::endl;
    #endif


    mtk::Real *ob{};

    auto ob_ld = num_bndy_coeffs_;

    try {
      ob = new mtk::Real[ob_ld];
    } catch (std::bad_alloc &memory_allocation_exception) {
      std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
        std::endl;
      std::cerr << memory_allocation_exception.what() << std::endl;
    }
    memset(ob, mtk::kZero, sizeof(ob[0])*ob_ld);

    ob[1] = mtk::kOne;

    #if MTK_DEBUG_LEVEL > 0
    std::cout << "ob = " << std::endl << std::endl;
    for (auto ii = 0; ii < ob_ld; ++ii) {
      std::cout << std::setw(12) << ob[ii] << std::endl;
    }
    std::cout << std::endl;
    #endif


    // However, this is an under-determined system of equations. So we can not
    // use the same LAPACK routine (dgesv_). We will instead use dgels_, through
    // our LAPACKAdapter class.

    int info_{
      mtk::LAPACKAdapter::SolveRectangularDenseSystem(AA_,
ob, ob_ld)};

    #if MTK_DEBUG_LEVEL > 0
    if (!info_) {
      std::cout << "System successfully solved!" << std::endl << std::endl;
    } else {
      std::cerr << "Error solving system! info = " << info_ << std::endl;
    }
    #endif

```

```
#if MTK_DEBUG_LEVEL > 0
std::cout << "ob =" << std::endl;
for (auto ii = 0; ii < ob_ld; ++ii) {
  std::cout << std::setw(12) << ob[ii] << std::endl;
}
std::cout << std::endl;
#endif


// This implies a DAXPY operation. However, we must construct the arguments
// for this operation.

// Save them into the ob_bottom array:

Real *ob_bottom{}; // Bottom part of the attained kernel used to scale it.

try {
  ob_bottom = new mtk::Real[dim_null_];
} catch (std::bad_alloc &memory_allocation_exception) {
  std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
    std::endl;
  std::cerr << memory_allocation_exception.what() << std::endl;
}
memset(ob_bottom, mtk::kZero, sizeof(ob_bottom[0])*dim_null_);

for (auto ii = 0; ii < dim_null_; ++ii) {
  ob_bottom[(dim_null_ - 1) - ii] = ob[num_bndy_coeffs_ - ii - 1];
}

#if MTK_DEBUG_LEVEL > 0
std::cout << "ob_bottom =" << std::endl;
for (auto ii = 0; ii < dim_null_; ++ii) {
  std::cout << std::setw(12) << ob_bottom[ii] << std::endl;
}
std::cout << std::endl;
#endif


// We must computed an scaled ob, sob, using the scaled null-space in
// rat_basis_null_space_.
// Such operation is: sob = ob - rat_basis_null_space_*ob_bottom
// or:                ob = -1.0*rat_basis_null_space_*ob_bottom + 1.0*ob
// thus:               Y =    a*A    *x          +   b*Y (DAXPY).

#if MTK_DEBUG_LEVEL > 0
std::cout << "Rational basis for the null-space:" << std::endl;
std::cout << rat_basis_null_space_ << std::endl;
#endif

mtk::Real alpha{-mtk::kOne};
mtk::Real beta{mtk::kOne};

mtk::BLASAdapter::RealDenseMV(alpha, rat_basis_null_space_,
                              ob_bottom, beta, ob);

#if MTK_DEBUG_LEVEL > 0
std::cout << "scaled ob:" << std::endl;
for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
  std::cout << std::setw(12) << ob[ii] << std::endl;
}
std::cout << std::endl;
#endif

// We save the recently scaled solution, into an array containing these.
// We can NOT start building the pi matrix, simply because I want that part
// to be separated since its construction depends on the algorithm we want
// to implement.

for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
  prem_apps_[ii*dim_null_ + ll] = ob[ii];
}

// After the first iteration, simply shift the entries of the last
// generator vector used:
for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
  gg[ii]--;
}

// Garbage collection for this loop:
delete[] ob;
ob = nullptr;
```

```
    delete[] ob_bottom;
    ob_bottom = nullptr;
  } // End of: for (ll = 0; ll < dim_null; ll++);

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "Matrix post-scaled preliminary apps: " << std::endl;
  for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
    for (auto jj = 0; jj < dim_null_; ++jj) {
      std::cout << std::setw(12) << prem_apps_[ii*dim_null_ + jj];
    }
    std::cout << std::endl;
  }
  std::cout << std::endl;
  #endif

  delete[] gg;
  gg = nullptr;

  return true;
}

bool mtk::Div1D::ComputeWeights(void) {

  // Matrix to copmpute the weights as in the CRSA.
  mtk::DenseMatrix pi(num_bndy_coeffs_, num_bndy_coeffs_ - 1);


  // Assemble the pi matrix using:
  // 1. The collection of scaled preliminary approximations.
  // 2. The collection of coefficients approximating at the interior.
  // 3. The scaled basis for the null-space.

  // 1.1. Process array of scaled preliminary approximations.

  // These are queued in scaled_solutions. Each one of these, will be a column
  // of the pi matrix:
  for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
    for (auto jj = 0; jj < dim_null_; ++jj) {
      pi.data()[ii*(2*dim_null_ + (order_accuracy_/2 + 1)) + jj] =
        prem_apps_[ii*dim_null_ + jj];
    }
  }

  // 1.2. Add columns from known stencil approximating at the interior.

  // However, these must be padded by zeros, according to their position in the
  // final pi matrix:
  auto mm = 0;
  for (auto jj = dim_null_; jj < order_accuracy_; ++jj) {
    for (auto ii = 0; ii < order_accuracy_; ++ii) {
      pi.data()[(ii + mm)*(2*dim_null_ + (order_accuracy_/2 + 1)) + jj] =
        coeffs_interior_[ii];
    }
    ++mm;
  }

  rat_basis_null_space_.OrderColMajor();

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "Rational basis for the null-space (col. major):" << std::endl;
  std::cout << rat_basis_null_space_ << std::endl;
  #endif

  // 1.3. Add final set of columns: rational basis for null-space.
  for (auto jj = dim_null_ + (order_accuracy_/2 + 1); jj < num_bndy_coeffs_ - 1; ++jj) {
    for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
      auto og =
        (jj - (dim_null_ + (order_accuracy_/2 + 1)))*num_bndy_coeffs_ + ii;
      auto de = ii*(2*dim_null_ + (order_accuracy_/2 + 1)) + jj;
      pi.data()[de] = rat_basis_null_space_.data()[og];
    }
  }

  #if MTK_DEBUG_LEVEL >0
  std::cout << "coeffs_interior_ =" << std::endl;
  for (auto ii = 0; ii < order_accuracy_; ++ii) {
    std::cout << std::setw(12) << coeffs_interior_[ii];
  }
  std::cout << std::endl << std::endl;
  #endif
```

```
#if MTK_DEBUG_LEVEL >0
  std::cout << "Constructed pi matrix for CRS Algorithm: " << std::endl;
  std::cout << pi << std::endl;
  #endif


  // This imposes the mimetic condition.

  mtk::Real *hh{};  // Right-hand side to compute weights in the C{R,B}SA.

  try {
    hh = new mtk::Real[num_bndy_coeffs_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(hh, mtk::kZero, sizeof(hh[0])*num_bndy_coeffs_);

  hh[0] = -mtk::kOne;
  for (auto ii = (order_accuracy_/2 + 2 - 1); ii < num_bndy_coeffs_; ++ii) {
    auto aux_xx = mtk::kZero;
    for (auto jj = 0; jj < ((ii - (order_accuracy_/2 - 1)) - 1); ++jj) {
      aux_xx += coeffs_interior_[jj];
    }
    hh[ii] = -mtk::kOne*aux_xx;
  }


  // That is, we construct a system, to solve for the weights.

  // Once again we face the challenge of solving with LAPACK. However, for the
  // CRSA, this matrix PI is over-determined, since it has more rows than
  // unknowns. However, according to the theory, the solution to this system is
  // unique. We will use dgels_.

  try {
    weights_cbs_ = new mtk::Real[num_bndy_coeffs_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(weights_cbs_, mtk::kZero, sizeof(weights_cbs_[0])*num_bndy_coeffs_);

  int weights_ld{pi.num_cols() + 1};

  // Preserve hh.
  std::copy(hh, hh + weights_ld, weights_cbs_);

  pi.Transpose();

  int info{mtk::LAPACKAdapter::SolveRectangularDenseSystem(
pi, weights_cbs_, weights_ld)};

  #if MTK_DEBUG_LEVEL > 0
  if (!info) {
    std::cout << "System successfully solved!" << std::endl << std::endl;
  } else {
    std::cerr << "Error solving system! info = " << info << std::endl;
  }
  #endif

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "hh =" << std::endl;
  for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
    std::cout << std::setw(11) << hh[ii] << std::endl;
  }
  std::cout << std::endl;
  #endif

  // Preserve the original weights for research.

  try {
    weights_crs_ = new mtk::Real[num_bndy_coeffs_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
```

```
  memset(weights_crs_, mtk::kZero, sizeof(weights_crs_[0])*num_bndy_coeffs_);

  std::copy(weights_cbs_, weights_cbs_ + (weights_ld - 1), weights_crs_);

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "weights_CRSA + lambda =" << std::endl;
  for (auto ii = 0; ii < weights_ld - 1; ++ii) {
    std::cout << std::setw(12) << weights_crs_[ii] << std::endl;
  }
  std::cout << std::endl;
  #endif

  if (order_accuracy_ >= mtk::kCriticalOrderAccuracyDiv) {

    int minrow_{std::numeric_limits<int>::infinity()};

    mtk::Real norm_{mtk::BLASAdapter::RealNRM2(weights_cbs_,
order_accuracy_)};
    mtk::Real minnorm_{std::numeric_limits<mtk::Real>::infinity()};


    mtk::DenseMatrix phi(order_accuracy_ + 1, order_accuracy_);

    for (auto ii = 0; ii < order_accuracy_ + 1; ++ii) {
      for (auto jj = 0; jj < dim_null_; ++jj) {
        phi.data()[ii*(order_accuracy_) + jj] = prem_apps_[ii*dim_null_ + jj];
      }
    }

    int aux{};  // Auxiliary variable.
    for (auto jj = dim_null_; jj < dim_null_ + 2; ++jj) {
      for (auto ii = 0; ii < order_accuracy_; ++ii) {
        phi.data()[(ii + aux)*order_accuracy_ + jj] = coeffs_interior_[ii];
      }
      ++aux;
    }

    for(auto jj=order_accuracy_ - 1; jj >=order_accuracy_ - dim_null_; jj--) {
      for(auto ii=0; ii<order_accuracy_ + 1; ++ii) {
        phi.data()[ii*order_accuracy_+jj] = mtk::kZero;
      }
    }

    for (auto jj = 0; jj < order_accuracy_ + 1; ++jj) {
      for (auto ii = 0; ii < dim_null_; ++ii) {
        phi.data()[(ii + order_accuracy_ - dim_null_ + jj*order_accuracy_)] =
          -prem_apps_[(dim_null_ - ii - 1 + jj*dim_null_)];
      }
    }

    for(auto ii = 0; ii < order_accuracy_/2; ++ii) {
      for (auto jj = dim_null_ + 2; jj < order_accuracy_; ++jj) {
        auto swap = phi.data()[ii*order_accuracy_+jj];
        phi.data()[ii*order_accuracy_ + jj] =
          phi.data()[(order_accuracy_-ii)*order_accuracy_+jj];
        phi.data()[(order_accuracy_-ii)*order_accuracy_+jj] = swap;
      }
    }

    #if MTK_DEBUG_LEVEL > 0
    std::cout << "Constructed PHI matrix for CBS Algorithm: " << std::endl;
    std::cout << phi << std::endl;
    #endif


    mtk::Real *lamed{};  // Used to build big lambda.

    try {
      lamed = new mtk::Real[dim_null_];
    } catch (std::bad_alloc &memory_allocation_exception) {
      std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
        std::endl;
      std::cerr << memory_allocation_exception.what() << std::endl;
    }
    memset(lamed, mtk::kZero, sizeof(lamed[0])*dim_null_);

    for (auto ii = 0; ii < dim_null_; ++ii) {
      lamed[ii] = hh[ii + order_accuracy_ + 1] ;
    }

    #if MTK_DEBUG_LEVEL > 0
```

```
    std::cout << "lamed =" << std::endl;
    for (auto ii = 0; ii < dim_null_; ++ii) {
      std::cout << std::setw(12) << lamed[ii] << std::endl;
    }
    std::cout << std::endl;
    #endif

    for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
      mtk::Real temp = mtk::kZero;
      for(auto jj = 0; jj < dim_null_; ++jj) {
        temp = temp +
          lamed[jj]*rat_basis_null_space_.data()[jj*num_bndy_coeffs_ + ii];
      }
      hh[ii] = hh[ii] - temp;
    }

    #if MTK_DEBUG_LEVEL > 0
    std::cout << "big_lambda =" << std::endl;
    for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
      std::cout << std::setw(12) << hh[ii] << std::endl;
    }
    std::cout << std::endl;
    #endif

    int copy_result{};

    mtk::Real normerr_; // Norm of the error for the solution on each row.


    for(auto row_= 0; row_ < order_accuracy_ + 1; ++row_) {
      normerr_ = mtk::GLPKAdapter::SolveSimplexAndCompare(phi.data(),
                                                          order_accuracy_ + 1,
                                                          order_accuracy_,
                                                          order_accuracy_,
                                                          hh,
                                                          weights_cbs_,
                                                          row_,
                                                          mimetic_threshold_,
                                                          copy_result);
      mtk::Real aux{normerr_/norm_};

      #if MTK_DEBUG_LEVEL>0
      std::cout << "Relative norm: " << aux << " " << std::endl;
      std::cout << std::endl;
      #endif

      if (aux < minnorm_) {
        minnorm_ = aux;
        minrow_= row_;
      }
    }

    #if MTK_DEBUG_LEVEL > 0
    std::cout << "weights_CBSA + lambda (after brute force search):" <<
      std::endl;
    for (auto ii = 0; ii < num_bndy_coeffs_ - 1; ++ii) {
      std::cout << std::setw(12) << weights_cbs_[ii] << std::endl;
    }
    std::cout << std::endl;
    #endif


    // After we know which row yields the smallest relative norm that row is
    // chosen to be the objective function and the result of the optimizer is
    // chosen to be the new weights_.

    #if MTK_DEBUG_LEVEL > 0
    std::cout << "Minimum Relative Norm " << minnorm_ << " found at row " <<
      minrow_ + 1 << std::endl;
    std::cout << std::endl;
    #endif

    copy_result = 1;
    normerr_ = mtk::GLPKAdapter::SolveSimplexAndCompare(phi.data(),
                                                        order_accuracy_ + 1,
                                                        order_accuracy_,
                                                        order_accuracy_,
                                                        hh,
                                                        weights_cbs_,
```

```
                                                        minrow_,
                                                        mimetic_threshold_,
                                                        copy_result);
    mtk::Real aux_{normerr_/norm_};
    #if MTK_DEBUG_LEVEL > 0
    std::cout << "Relative norm: " << aux_ << std::endl;
    std::cout << std::endl;
    #endif

    delete [] lamed;
    lamed = nullptr;
  }

  delete [] hh;
  hh = nullptr;

  return true;
}

bool mtk::Div1D::ComputeStencilBoundaryGrid(void) {

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "weights_CBSA + lambda =" << std::endl;
  for (auto ii = 0; ii < num_bndy_coeffs_ - 1; ++ii) {
    std::cout << std::setw(12) << weights_cbs_[ii] << std::endl;
  }
  std::cout << std::endl;
  #endif


  mtk::Real *lambda{}; // Collection of bottom values from weights_.

  try {
    lambda = new mtk::Real[dim_null_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(lambda, mtk::kZero, sizeof(lambda[0])*dim_null_);

  for (auto ii = 0; ii < dim_null_; ++ii) {
    lambda[ii] = weights_cbs_[order_accuracy_ + ii];
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "lambda =" << std::endl;
  for (auto ii = 0; ii < dim_null_; ++ii) {
    std::cout << std::setw(12) << lambda[ii] << std::endl;
  }
  std::cout << std::endl;
  #endif


  mtk::Real *alpha{}; // Collection of alpha values.

  try {
    alpha = new mtk::Real[dim_null_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(alpha, mtk::kZero, sizeof(alpha[0])*dim_null_);

  for (auto ii = 0; ii < dim_null_; ++ii) {
    alpha[ii] = lambda[ii]/weights_cbs_[ii] ;
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "alpha =" << std::endl;
  for (auto ii = 0; ii < dim_null_; ++ii) {
    std::cout << std::setw(12) << alpha[ii] << std::endl;
  }
  std::cout << std::endl;
  #endif


  try {
    mim_bndy_ = new mtk::Real[num_bndy_coeffs_*dim_null_];
  } catch (std::bad_alloc &memory_allocation_exception) {
```

```
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(mim_bndy_, mtk::kZero, sizeof(mim_bndy_[0])*num_bndy_coeffs_*dim_null_);

  for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
    for (auto jj = 0; jj < dim_null_; ++jj) {
      mim_bndy_[ii*dim_null_ + jj] =
        prem_apps_[ii*dim_null_ + jj] +
        alpha[jj]*rat_basis_null_space_.data()[jj*num_bndy_coeffs_ + ii];
    }
  }

  #if MTK_DEBUG_LEVEL >0
  std::cout << "Collection of mimetic approximations:" << std::endl;
  for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
    for (auto jj = 0; jj < dim_null_; ++jj) {
      std::cout << std::setw(13) << mim_bndy_[ii*dim_null_ + jj];
    }
    std::cout << std::endl;
  }
  std::cout << std::endl;
  #endif

  delete[] lambda;
  lambda = nullptr;

  delete[] alpha;
  alpha = nullptr;

  return true;
}

bool mtk::Div1D::AssembleOperator(void) {

  // The output array will have this form:
  // 1. The first entry of the array will contain the used order order_accuracy_.
  // 2. The second entry of the array will contain the collection of
  // approximating coefficients for the interior of the grid.
  // 3. IF order_accuracy_ > 2, then the third entry will contain a collection of weights.
  // 4. IF order_accuracy_ > 2, the next dim_null_ entries will contain the collections of
  // approximating coefficients for the west boundary of the grid.

  if (order_accuracy_ > mtk::kDefaultOrderAccuracy) {
    divergence_length_ =
      1 + order_accuracy_ + order_accuracy_ + dim_null_*num_bndy_coeffs_;
  } else {
    divergence_length_ = 1 + order_accuracy_;
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "divergence_length_ = " << divergence_length_ << std::endl;
  #endif

  try {
    divergence_ = new double[divergence_length_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(divergence_, mtk::kZero, sizeof(divergence_[0])*divergence_length_);


  divergence_[0] = order_accuracy_;


  for (auto ii = 0; ii < order_accuracy_; ++ii) {
  divergence_[ii + 1] = coeffs_interior_[ii];
  }


  if (order_accuracy_ > 2) {
    for (auto ii = 0; ii < order_accuracy_; ++ii) {
      divergence_[(1 + order_accuracy_) + ii] = weights_cbs_[ii];
    }
  }


  if (order_accuracy_ > 2) {
```

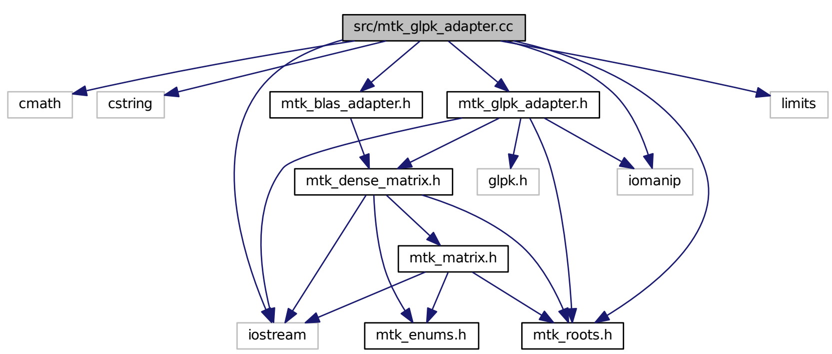
```
    auto offset = (2*order_accuracy_ + 1);
    int mm{};
    for (auto ii = 0; ii < dim_null_; ++ii) {
      for (auto jj = 0; jj < num_bndy_coeffs_; ++jj) {
        divergence_[offset + (mm)] = mim_bndy_[jj*dim_null_ + ii];
        ++mm;
      }
    }
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "1D " << order_accuracy_ << "-order div built!" << std::endl;
  std::cout << std::endl;
  #endif

  return true;
}
```

17.45 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:



17.45.1 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

Todo Document better this file.

Definition in file mtk_glpk_adapter.cc.

17.46 mtk_glpk_adapter.cc

```
/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#include <cmath>
#include <cstring>

#include <iostream>
#include <iomanip>
#include <limits>

#include "mtk_roots.h"
#include "mtk_blas_adapter.h"
#include "mtk_glpk_adapter.h"

```

```
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_threshold,
                                                  int copy) {

  #if MTK_DEBUG_LEVEL > 0
  char mps_file_name[18]; //  File name for the MPS files.
  #endif
  char rname[5];          //
  char cname[5];          //

  glp_prob *lp; // Linear programming problem.

  int *ia;  //
  int *ja;  //

  int problem_size; // Size of the problem.
  int lp_nrows;     // Number of rows.
  int lp_ncols;     // Number of columns.
  int matsize;      //
  int glp_index{1};    // Index of the objective function.
  int ii;           //
  int jj;           //

  mtk::Real *ar;            //
  mtk::Real *objective;     //
  mtk::Real *rhs;           //
  mtk::Real *err;           //
  mtk::Real x1;             //

  #if MTK_DEBUG_LEVEL > 0
  mtk::Real obj_value;      //
  #endif

  lp_nrows = kk;
  lp_ncols = kk;

  matsize = lp_nrows*lp_ncols;


  problem_size = lp_nrows*lp_ncols + 1;

  try {
    ia = new int[problem_size];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(ia, 0, sizeof(ia[0])*problem_size);

  try {
    ja = new int[problem_size];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(ja, 0, sizeof(ja[0])*problem_size);

  try {
    ar = new mtk::Real[problem_size];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(ar, mtk::kZero, sizeof(ar[0])*problem_size);

  try {
    objective = new mtk::Real[lp_ncols + 1];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
```

```
  }
  memset(objective, mtk::kZero, sizeof(objective[0])*(lp_ncols + 1));

  try {
    rhs = new mtk::Real[lp_nrows + 1];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(rhs, mtk::kZero, sizeof(rhs[0])*(lp_nrows + 1));

  try {
    err = new mtk::Real[lp_nrows];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(err, mtk::kZero, sizeof(err[0])*(lp_nrows));

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "Problem size: " << problem_size << std::endl;
  std::cout << "lp_nrows = " << lp_nrows << std::endl;
  std::cout << "lp_ncols = " << lp_ncols << std::endl;
  std::cout << std::endl;
  #endif

  lp = glp_create_prob();

  glp_set_prob_name (lp, "mtk::GLPKAdapter::Simplex");

  glp_set_obj_dir (lp, GLP_MIN);


  glp_add_rows(lp, lp_nrows);

  for (ii = 1; ii <= lp_nrows; ++ii) {
    sprintf(rname, "R%02d",ii);
    glp_set_row_name(lp, ii, rname);
  }

  glp_add_cols(lp, lp_ncols);

  for (ii = 1; ii <= lp_ncols; ++ii) {
    sprintf(cname, "Q%02d",ii);
    glp_set_col_name (lp, ii, cname);
  }


  #if MTK_DEBUG_LEVEL>0
  std::cout << "Using row " << robjective + 1 << " as objective." << std::endl;
  #endif
  for (jj = 0; jj < kk; ++jj) {
    objective[glp_index] = A[jj + robjective * ncols];
    glp_index++;
  }
  #if MTK_DEBUG_LEVEL >0
  std::cout << std::endl;
  #endif


  glp_index = 1;
  rhs[0] = mtk::kZero;
  for (ii = 0; ii <= lp_nrows; ++ii) {
    if (ii != robjective) {
      rhs[glp_index] = hh[ii];
      glp_set_row_bnds(lp, glp_index, GLP_UP, 0.0, rhs[glp_index]);
      glp_index++;
    }
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "rhs =" << std::endl;
  for (auto ii = 0; ii < lp_nrows; ++ii) {
    std::cout << std::setw(15) << rhs[ii] << std::endl;
  }
  std::cout << std::endl;
  #endif

```

```
  for (ii = 1; ii <= lp_ncols; ++ii) {
    glp_set_obj_coef (lp, ii, objective[ii]);
  }


  for (ii = 1; ii <= lp_ncols; ++ii) {
    glp_set_col_bnds (lp, ii, GLP_LO, mimetic_threshold, 0.0);
  }


  glp_index = 1;
  for (ii = 0; ii <= kk; ++ii) {
    for (jj = 0; jj < kk; ++jj) {
      if (ii != robjective) {
        ar[glp_index] = A[jj + ii * ncols];
        glp_index++;
      }
    }
  }

  glp_index = 0;

  for (ii = 1; ii < problem_size; ++ii) {
    if (((ii - 1) % lp_ncols) == 0) {
      glp_index++;
    }
    ia[ii] = glp_index;
    ja[ii] = (ii - 1) % lp_ncols + 1;
  }

  glp_load_matrix (lp, matsize, ia, ja, ar);

  #if MTK_DEBUG_LEVEL > 0
  sprintf(mps_file_name, "LP_MPS_row_%02d.mps", robjective);
  glp_write_mps(lp, GLP_MPS_FILE, nullptr, mps_file_name);
  #endif


  glp_simplex (lp, nullptr);

  // Check status of the solution.

  if (glp_get_status(lp) == GLP_OPT) {

    for(ii = 1; ii <= lp_ncols; ++ii) {
      err[ii - 1] = qq[ii - 1] - glp_get_col_prim(lp,ii);
    }

    #if MTK_DEBUG_LEVEL > 0
    obj_value = glp_get_obj_val (lp);
    std::cout << std::setw(12) << "CBS" << std::setw(12) << "CRS" << std::endl;
    for (ii = 0; ii < lp_ncols; ++ii) {
      std::cout << "q_" << ii + 1 << " = " << std::setw(12) <<
        glp_get_col_prim(lp,ii + 1) << std::setw(12) << qq[ii] << std::endl;
    }
    std::cout << "Objective function value (row " << robjective + 1 << ") = " <<
      obj_value << std::endl;
    #endif

    if (copy) {
      for(ii = 0; ii < lp_ncols; ++ii) {
        qq[ii] = glp_get_col_prim(lp,ii + 1);
      }
      // Preserve the bottom values of qq.
    }

    x1 = mtk::BLASAdapter::RealNRM2(err,lp_ncols);

  } else {
    x1 = std::numeric_limits<mtk::Real>::infinity();
  }

  glp_delete_prob (lp);
  glp_free_env ();

  delete [] ia;
  delete [] ja;
  delete [] ar;
  delete [] objective;
  delete [] rhs;
  delete [] err;
```

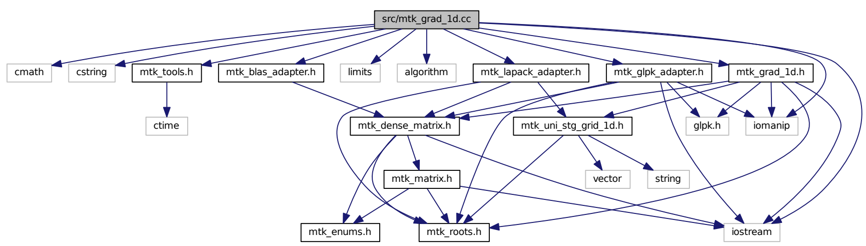
```

  return x1;
}
```

17.47 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)

17.47.1 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.

17.48 mtk_grad_1d.cc

```
/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#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"

namespace mtk {

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


  stream << "gradient_[0] = " << std::setw(9) << in.gradient_[0] << std::endl;

```

```

  stream << "gradient_[1:" << in.order_accuracy_ << "] = ";
  for (auto ii = 1; ii <= in.order_accuracy_; ++ii) {
    stream << std::setw(9) << in.gradient_[ii] << " ";
  }
  stream << std::endl;


  stream << "gradient_[" << in.order_accuracy_ + 1 << ":" <<
    2*in.order_accuracy_ << "] = ";
  for (auto ii = in.order_accuracy_ + 1; ii <= 2*in.
order_accuracy_; ++ii) {
    stream << std::setw(9) << in.gradient_[ii] << " ";
  }
  stream << std::endl;


  int offset{2*in.order_accuracy_ + 1};
  int mm {};

  stream << "gradient_[" << offset + mm << ":" <<
    offset + mm + in.num_bndy_coeffs_ - 1 << "] = ";

  if (in.order_accuracy_ > mtk::kDefaultOrderAccuracy) {
    for (auto ii = 0; ii < in.num_bndy_approxs_ ; ++ii) {
      for (auto jj = 0; jj < in.num_bndy_coeffs_; jj++) {
        auto value = in.gradient_[offset + (mm)];
        stream << std::setw(9) << value << " ";
        mm++;
      }
    }
  } else {
    stream << std::setw(9) << in.gradient_[offset + 0] << ' ';
    stream << std::setw(9) << in.gradient_[offset + 1] << ' ';
    stream << std::setw(9) << in.gradient_[offset + 2] << ' ';
  }
  stream << std::endl;

  return stream;
}
}

mtk::Grad1D::Grad1D():
  order_accuracy_(mtk::kDefaultOrderAccuracy),
  dim_null_(),
  num_bndy_approxs_(),
  num_bndy_coeffs_(),
  gradient_length_(),
  minrow_(),
  row_(),
  coeffs_interior_(),
  prem_apps_(),
  weights_crs_(),
  weights_cbs_(),
  mim_bndy_(),
  gradient_(),
  mimetic_threshold_(mtk::kDefaultMimeticThreshold) {}

mtk::Grad1D::Grad1D(const Grad1D &grad):
  order_accuracy_(grad.order_accuracy_),
  dim_null_(grad.dim_null_),
  num_bndy_approxs_(grad.num_bndy_approxs_),
  num_bndy_coeffs_(grad.num_bndy_coeffs_),
  gradient_length_(grad.gradient_length_),
  minrow_(grad.minrow_),
  row_(grad.row_),
  coeffs_interior_(grad.coeffs_interior_),
  prem_apps_(grad.prem_apps_),
  weights_crs_(grad.weights_crs_),
  weights_cbs_(grad.weights_cbs_),
  mim_bndy_(grad.mim_bndy_),
  gradient_(grad.gradient_),
  mimetic_threshold_(grad.mimetic_threshold_) {}

mtk::Grad1D::~Grad1D() {

  delete[] coeffs_interior_;
  coeffs_interior_ = nullptr;

  delete[] prem_apps_;
  prem_apps_ = nullptr;
```

```

  delete[] weights_crs_;
  weights_crs_ = nullptr;

  delete[] weights_cbs_;
  weights_cbs_ = nullptr;

  delete[] mim_bndy_;
  mim_bndy_ = nullptr;

  delete[] gradient_;
  gradient_ = nullptr;
}

bool mtk::Grad1D::ConstructGrad1D(int order_accuracy,
Real mimetic_threshold) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(order_accuracy < 2, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent((order_accuracy%2) != 0, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent(mimetic_threshold <= mtk::kZero,
                      __FILE__, __LINE__, __func__);

  if (order_accuracy >= mtk::kCriticalOrderAccuracyGrad) {
    std::cout << "WARNING: Numerical accuracy is high." << std::endl;;
  }

  std::cout << "order_accuracy_ = " << order_accuracy << std::endl;;
  std::cout << "mimetic_threshold_ = " << mimetic_threshold << std::endl;;
  #endif

  order_accuracy_ = order_accuracy;
  mimetic_threshold_ = mimetic_threshold;


  bool abort_construction = ComputeStencilInteriorGrid();

  #if MTK_DEBUG_LEVEL > 0
  if (!abort_construction) {
    std::cerr << "Could NOT complete stage 1." << std::endl;;
    std::cerr << "Exiting..." << std::endl;;
    return false;
  }
  #endif

  // At this point, we already have the values for the interior stencil stored
  // in the coeffs_interior_ array.

  dim_null_ = order_accuracy_/2 - 1;

  num_bndy_approxs_ = dim_null_ + 1;

  #ifdef MTK_PRECISION_DOUBLE
  num_bndy_coeffs_ = (int) (3.0*((mtk::Real) order_accuracy_)/2.0);
  #else
  num_bndy_coeffs_ = (int) (3.0f*((mtk::Real) order_accuracy_)/2.0f);
  #endif


  // For this we will follow recommendations given in:
  //
  // http://icl.cs.utk.edu/lapack-forum/viewtopic.php?f=5&t=4506
  //
  // We will compute the QR Factorization of the transpose, as in the
  // following (MATLAB) pseudo-code:
  //
  // [Q,R] = qr(V'); % Full QR as defined in
  // % http://www.stanford.edu/class/ee263/notes/qr_matlab.pdf
  //
  // null-space = Q(:, last (order_accuracy_/2 - 1) columns of Q );
  //
  // However, given the nature of the Vandermonde matrices we've just
  // computed, they all posses the same null-space. Therefore, we impose the
  // convention of computing the null-space of the first Vandermonde matrix
  // (west boundary).

  // In the case of the gradient, the first Vandermonde system has a unique
  // solution for the case of second-order-accuracy. Ergo, the Vandermonde
  // matrix used to assemble said system, will have an empty null-space.

  // Therefore, we only compute a rational basis for the case of order higher
```

```
  // than second.

  if (dim_null_ > 0) {

    abort_construction = ComputeRationalBasisNullSpace();

    #if MTK_DEBUG_LEVEL > 0
    if (!abort_construction) {
      std::cerr << "Could NOT complete stage 2.1." << std::endl;;
      std::cerr << "Exiting..." << std::endl;;
      return false;
    }
    #endif
  }


  abort_construction = ComputePreliminaryApproximations();

  #if MTK_DEBUG_LEVEL > 0
  if (!abort_construction) {
    std::cerr << "Could NOT complete stage 2.2." << std::endl;;
    std::cerr << "Exiting..." << std::endl;;
    return false;
  }
  #endif


  abort_construction = ComputeWeights();

  #if MTK_DEBUG_LEVEL > 0
  if (!abort_construction) {
    std::cerr << "Could NOT complete stage 2.3." << std::endl;;
    std::cerr << "Exiting..." << std::endl;;
    return false;
  }
  #endif


  if (dim_null_ > 0) {

    abort_construction = ComputeStencilBoundaryGrid();

    #if MTK_DEBUG_LEVEL > 0
    if (!abort_construction) {
      std::cerr << "Could NOT complete stage 2.4." << std::endl;;
      std::cerr << "Exiting..." << std::endl;;
      return false;
    }
    #endif
  }


  // Once we have the following three collections of data:
  //   (a) the coefficients for the interior,
  //   (b) the coefficients for the boundary (if it applies),
  //   (c) and the weights (if it applies),
  // we will store everything in the output array:

  abort_construction = AssembleOperator();

  #if MTK_DEBUG_LEVEL > 0
  if (!abort_construction) {
    std::cerr << "Could NOT complete stage 3." << std::endl;;
    std::cerr << "Exiting..." << std::endl;;
    return false;
  }
  #endif

  return true;
}

int mtk::Grad1D::num_bndy_coeffs() {

  return num_bndy_coeffs_;
}

mtk::Real *mtk::Grad1D::weights_crs() {

  return weights_crs_;
}

```

```
mtk::Real *mtk::Grad1D::weights_cbs() {

  return weights_cbs_;
}

mtk::DenseMatrix mtk::Grad1D::ReturnAsDenseMatrix(const UniStgGrid1D &grid) {

  int nn{grid.num_cells_x()}; // Number of cells on the grid.

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(order_accuracy_ <= 0, __FILE__, __LINE__, __func__);


  mtk::Tools::Prevent(nn < 3*order_accuracy_ - 2, __FILE__, __LINE__, __func__);
  #endif

  mtk::Real inv_delta_x{mtk::kOne/grid.delta_x()};

  int gg_num_rows = nn + 1;
  int gg_num_cols = nn + 2;
  int elements_per_row = num_bndy_coeffs_;
  int num_extra_rows = order_accuracy_/2;

  // Output matrix featuring sizes for gradient operators.
  mtk::DenseMatrix out(gg_num_rows, gg_num_cols);


  auto ee_index = 0;
  for (auto ii = 0; ii < num_extra_rows; ii++) {
    auto cc = 0;
    for(auto jj = 0 ; jj < gg_num_cols; jj++) {
      if(cc >= elements_per_row) {
        out.SetValue(ii, jj, mtk::kZero);
      } else {
        out.SetValue(ii,jj,
                     gradient_[2*order_accuracy_ + 1 + ee_index++]*inv_delta_x);
        cc++;
      }
    }
  }


  for (auto ii = num_extra_rows; ii < gg_num_rows - num_extra_rows; ii++) {
    auto jj = ii - num_extra_rows + 1;
    for (auto cc = 0; cc < order_accuracy_; cc++, jj++) {
      out.SetValue(ii, jj, coeffs_interior_[cc]*inv_delta_x);
    }
  }


  ee_index = 0;
  for (auto ii = gg_num_rows - 1; ii >= gg_num_rows - num_extra_rows; ii--) {
    auto cc = 0;
    for (auto jj = gg_num_cols - 1; jj >= 0; jj--) {
      if(cc >= elements_per_row) {
        out.SetValue(ii,jj,mtk::kZero);
      } else {
        out.SetValue(ii,jj,
                     -gradient_[2*order_accuracy_ + 1 + ee_index++]*inv_delta_x);
        cc++;
      }
     }
  }

  return out;
}

bool mtk::Grad1D::ComputeStencilInteriorGrid() {


  mtk::Real* pp{}; // Spatial coordinates to create interior stencil.

  try {
    pp = new mtk::Real[order_accuracy_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(pp, mtk::kZero, sizeof(pp[0])*order_accuracy_);
```

```
#ifdef MTK_PRECISION_DOUBLE
pp[0] = 1.0/2.0 - ((mtk::Real) order_accuracy_)/2.0;
#else
pp[0] = 1.0f/2.0f - ((mtk::Real) order_accuracy_)/2.0f;
#endif

for (auto ii = 1; ii < order_accuracy_; ++ii) {
  pp[ii] = pp[ii - 1] + mtk::kOne;
}

#if MTK_DEBUG_LEVEL > 0
std::cout << "pp =" << std::endl;
for (auto ii = 0; ii < order_accuracy_; ++ii) {
  std::cout << std::setw(12) << pp[ii];
}
std::cout << std::endl << std::endl;
#endif


bool transpose{false};

mtk::DenseMatrix vander_matrix(pp,order_accuracy_,order_accuracy_,transpose);

#if MTK_DEBUG_LEVEL > 0
std::cout << "vander_matrix = " << std::endl;
std::cout << vander_matrix << std::endl << std::endl;
#endif


try {
  coeffs_interior_ = new mtk::Real[order_accuracy_];
} catch (std::bad_alloc &memory_allocation_exception) {
  std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
    std::endl;
  std::cerr << memory_allocation_exception.what() << std::endl;
}
memset(coeffs_interior_, mtk::kZero, sizeof(coeffs_interior_[0])*order_accuracy_);

coeffs_interior_[1] = mtk::kOne;

#if MTK_DEBUG_LEVEL > 0
std::cout << "oo =" << std::endl;
for (auto ii = 0; ii < order_accuracy_; ++ii) {
  std::cout << std::setw(12) << coeffs_interior_[ii] << std::endl;
}
std::cout << std::endl;
#endif


int info{mtk::LAPACKAdapter::SolveDenseSystem(vander_matrix,
                                              coeffs_interior_)};

#if MTK_DEBUG_LEVEL > 0
if (!info) {
  std::cout << "System solved! Interior stencil attained!" << std::endl;
  std::cout << std::endl;
}
else {
  std::cerr << "Something wrong solving system! info = " << info << std::endl;
  std::cerr << "Exiting..." << std::endl;
  return false;
}
#endif

#if MTK_DEBUG_LEVEL > 0
std::cout << "coeffs_interior_ =" << std::endl;
for (auto ii = 0; ii < order_accuracy_; ++ii) {
  std::cout << std::setw(12) << coeffs_interior_[ii];
}
std::cout << std::endl << std::endl;
#endif

delete [] pp;
pp = nullptr;

return true;
}

bool mtk::Grad1D::ComputeRationalBasisNullSpace(void) {

```

```
  mtk::Real* gg{}; // Generator vector for the first Vandermonde matrix.

  try {
    gg = new mtk::Real[num_bndy_coeffs_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(gg, mtk::kZero, sizeof(gg[0])*num_bndy_coeffs_);

  #ifdef MTK_PRECISION_DOUBLE
  gg[1] = 1.0/2.0;
  #else
  gg[1] = 1.0f/2.0f;
  #endif
  for (auto ii = 2; ii < num_bndy_coeffs_; ++ii) {
    gg[ii] = gg[ii - 1] + mtk::kOne;
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "gg =" << std::endl;
  for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
    std::cout << std::setw(12) << gg[ii];
  }
  std::cout << std::endl << std::endl;
  #endif


  bool tran{true}; // Should I transpose the Vandermonde matrix.

  mtk::DenseMatrix aa_west_t(gg, num_bndy_coeffs_, order_accuracy_ + 1, tran);

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "aa_west_t =" << std::endl;
  std::cout << aa_west_t << std::endl;
  #endif


  mtk::DenseMatrix qq_t(mtk::LAPACKAdapter::QRFactorDenseMatrix
(aa_west_t));

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "qq_t = " << std::endl;
  std::cout << qq_t << std::endl;
  #endif


  int kk_num_rows{num_bndy_coeffs_};
  int kk_num_cols{dim_null_};

  mtk::DenseMatrix kk(kk_num_rows, kk_num_cols);

  // In the case of the gradient, even though we must solve for a null-space
  // of dimension 2, we must only extract ONE basis for the kernel.
  // We perform this extraction here:

  int aux_{kk_num_rows - kk_num_cols};
  for (auto ii = kk_num_rows - kk_num_cols; ii < kk_num_rows; ii++) {
    aux_--;
    for (auto jj = 0; jj < kk_num_rows; jj++) {
      kk.data()[jj*kk_num_cols + (kk_num_rows - kk_num_cols - aux_ - 1)] =
        qq_t.data()[ii*num_bndy_coeffs_ + jj];
    }
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "kk =" << std::endl;
  std::cout << kk << std::endl;
  std::cout << "kk.num_rows() = " << kk.num_rows() << std::endl;
  std::cout << "kk.num_cols() = " << kk.num_cols() << std::endl;
  std::cout << std::endl;
  #endif


  // Scale thus requesting that the last entries of the attained basis for the
  // null-space, adopt the pattern we require.
  // Essentially we will implement the following MATLAB pseudo-code:
  //  scalers = kk(num_bndy_approxs - (dim_null - 1):num_bndy_approxs,:)\B
  //  SK = kk*scalers
```

```
// where SK is the scaled null-space.

// In this point, we almost have all the data we need correctly allocated
// in memory. We will create the matrix iden_, and elements we wish to scale in
// the kk array. Using the concept of the leading dimension, we could just
// use kk, with the correct leading dimension and that is it. BUT I DO NOT
// GET how does it work. So I will just create a matrix with the content of
// this array that we need, solve for the scalers and then scale the
// whole kk:

// We will then create memory for that sub-matrix of kk (subk).

mtk::DenseMatrix subk(dim_null_, dim_null_);

auto zz = 0;
for (auto ii = order_accuracy_ + 1; ii < num_bndy_coeffs_; ii++) {
  for (auto jj = 0; jj < dim_null_; jj++) {
    subk.data()[zz*(dim_null_) + jj] = kk.data()[ii*(dim_null_) + jj];
  }
  zz++;
}

#if MTK_DEBUG_LEVEL > 0
std::cout << "subk =" << std::endl;
std::cout << subk << std::endl;
#endif

subk.Transpose();

#if MTK_DEBUG_LEVEL > 0
std::cout << "subk_t =" << std::endl;
std::cout << subk << std::endl;
#endif

bool padded{false};
tran = false;

mtk::DenseMatrix iden(dim_null_, padded, tran);

#if MTK_DEBUG_LEVEL > 0
std::cout << "iden =" << std::endl;
std::cout << iden << std::endl;
#endif

// Solve the system to compute the scalers.
// An example of the system to solve, for k = 8, is:
//
// subk*scalers = iden or
//
// |  0.386018  -0.0339244   -0.129478 |            | 1 0 0 |
// | -0.119774   0.0199423   0.0558632 |*scalers =  | 0 1 0 |
// | 0.0155708 -0.00349546 -0.00853182 |            | 0 0 1 |
//
// Notice this is a nrhs = 3 system.
// Noteworthy: we do NOT ACTUALLY ALLOCATE space for the scalers... they
// will be stored in the created identity matrix.
// Let us first transpose subk (because of LAPACK):

int info{mtk::LAPACKAdapter::SolveDenseSystem(subk, iden)};

#if MTK_DEBUG_LEVEL > 0
if (!info) {
  std::cout << "System successfully solved!" <<
    std::endl;
} else {
  std::cerr << "Something went wrong solving system! info = " << info <<
    std::endl;
  std::cerr << "Exiting..." << std::endl;
  return false;
}
std::cout << std::endl;
#endif

#if MTK_DEBUG_LEVEL > 0
std::cout << "Computed scalers:" << std::endl;
std::cout << iden << std::endl;
#endif

// Multiply the two matrices to attain a scaled basis for null-space.

rat_basis_null_space_ = mtk::BLASAdapter::RealDenseMM(kk, iden);
```

```
  #if MTK_DEBUG_LEVEL > 0
  std::cout << "Rational basis for the null-space:" << std::endl;
  std::cout << rat_basis_null_space_ << std::endl;
  #endif

  // At this point, we have a rational basis for the null-space, with the
  // pattern we need! :)

  delete [] gg;
  gg = nullptr;

  return true;
}

bool mtk::Grad1D::ComputePreliminaryApproximations() {


  mtk::Real *gg{}; // Generator vector for the first approximation.

  try {
    gg = new mtk::Real[num_bndy_coeffs_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(gg, mtk::kZero, sizeof(gg[0])*num_bndy_coeffs_);

  #ifdef MTK_PRECISION_DOUBLE
  gg[1] = 1.0/2.0;
  #else
  gg[1] = 1.0f/2.0f;
  #endif
  for (auto ii = 2; ii < num_bndy_coeffs_; ++ii) {
    gg[ii] = gg[ii - 1] + mtk::kOne;
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "gg0 =" << std::endl;
  for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
    std::cout << std::setw(12) << gg[ii];
  }
  std::cout << std::endl << std::endl;
  #endif

  // Allocate 2D array to store the collection of preliminary approximations.
  try {
    prem_apps_ = new mtk::Real[num_bndy_coeffs_*num_bndy_approxs_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(prem_apps_,
         mtk::kZero,
         sizeof(prem_apps_[0])*num_bndy_coeffs_*num_bndy_approxs_);


  for (auto ll = 0; ll < num_bndy_approxs_; ++ll) {

    // Re-check new generator vector for every iteration except for the first.
    #if MTK_DEBUG_LEVEL > 0
    if (ll > 0) {
      std::cout << "gg" << ll << " =" << std::endl;
      for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
        std::cout << std::setw(12) << gg[ii];
      }
      std::cout << std::endl << std::endl;
    }
    #endif


    bool transpose{false};

    mtk::DenseMatrix aa(gg,
                         num_bndy_coeffs_, order_accuracy_ + 1,
                         transpose);

    #if MTK_DEBUG_LEVEL > 0
    std::cout << "aa_" << ll << " =" << std::endl;
```

```
std::cout << aa << std::endl;
#endif


mtk::Real *ob{};

auto ob_ld = num_bndy_coeffs_;

try {
  ob = new mtk::Real[ob_ld];
} catch (std::bad_alloc &memory_allocation_exception) {
  std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
    std::endl;
  std::cerr << memory_allocation_exception.what() << std::endl;
}
memset(ob, mtk::kZero, sizeof(ob[0])*ob_ld);

ob[1] = mtk::kOne;

#if MTK_DEBUG_LEVEL > 0
std::cout << "ob = " << std::endl << std::endl;
for (auto ii = 0; ii < ob_ld; ++ii) {
  std::cout << std::setw(12) << ob[ii] << std::endl;
}
std::cout << std::endl;
#endif


// However, this is an under-determined system of equations. So we can not
// use the same LAPACK routine (dgesv_). We will instead use dgels_, through
// our LAPACKAdapter class.

int info_{
  mtk::LAPACKAdapter::SolveRectangularDenseSystem(aa, ob
, ob_ld)};

#if MTK_DEBUG_LEVEL > 0
if (!info_) {
  std::cout << "System successfully solved!" << std::endl << std::endl;
} else {
  std::cerr << "Error solving system! info = " << info_ << std::endl;
}
#endif

#if MTK_DEBUG_LEVEL > 0
std::cout << "ob =" << std::endl;
for (auto ii = 0; ii < ob_ld; ++ii) {
  std::cout << std::setw(12) << ob[ii] << std::endl;
}
std::cout << std::endl;
#endif


// This implies a DAXPY operation. However, we must construct the arguments
// for this operation.

// Save them into the ob_bottom array:

Real *ob_bottom{}; // Bottom part of the attained kernel used to scale it.

try {
  ob_bottom = new mtk::Real[dim_null_];
} catch (std::bad_alloc &memory_allocation_exception) {
  std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
    std::endl;
  std::cerr << memory_allocation_exception.what() << std::endl;
}
memset(ob_bottom, mtk::kZero, sizeof(ob_bottom[0])*dim_null_);

for (auto ii = 0; ii < dim_null_; ++ii) {
  ob_bottom[(dim_null_ - 1) - ii] = ob[num_bndy_coeffs_ - ii - 1];
}

#if MTK_DEBUG_LEVEL > 0
std::cout << "ob_bottom =" << std::endl;
for (auto ii = 0; ii < dim_null_; ++ii) {
  std::cout << std::setw(12) << ob_bottom[ii] << std::endl;
}
std::cout << std::endl;
#endif

```

```

    // We must computed an scaled ob, sob, using the scaled null-space in
    // rat_basis_null_space_.
    // Such operation is: sob = ob - rat_basis_null_space_*ob_bottom
    // or:                 ob = -1.0*rat_basis_null_space_*ob_bottom + 1.0*ob
    // thus:                Y =    a*A    *x         +   b*Y (DAXPY).

    #if MTK_DEBUG_LEVEL > 0
    std::cout << "Rational basis for the null-space:" << std::endl;
    std::cout << rat_basis_null_space_ << std::endl;
    #endif

    mtk::Real alpha{-mtk::kOne};
    mtk::Real beta{mtk::kOne};

    mtk::BLASAdapter::RealDenseMV(alpha, rat_basis_null_space_,
                                  ob_bottom, beta, ob);

    #if MTK_DEBUG_LEVEL > 0
    std::cout << "scaled ob:" << std::endl;
    for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
      std::cout << std::setw(12) << ob[ii] << std::endl;
    }
    std::cout << std::endl;
    #endif

    // We save the recently scaled solution, into an array containing these.
    // We can NOT start building the pi matrix, simply because I want that part
    // to be separated since its construction depends on the algorithm we want
    // to implement.

    for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
      prem_apps_[ii*num_bndy_approxs_ + ll] = ob[ii];
    }

    // After the first iteration, simply shift the entries of the last
    // generator vector used:
    for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
      gg[ii]--;
    }

    // Garbage collection for this loop:
    delete[] ob;
    ob = nullptr;

    delete[] ob_bottom;
    ob_bottom = nullptr;
  } // End of: for (ll = 0; ll < dim_null; ll++);

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "Matrix post-scaled preliminary apps: " << std::endl;
  for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
    for (auto jj = 0; jj < num_bndy_approxs_; ++jj) {
      std::cout << std::setw(12) << prem_apps_[ii*num_bndy_approxs_ + jj];
    }
    std::cout << std::endl;
  }
  std::cout << std::endl;
  #endif

  delete[] gg;
  gg = nullptr;

  return true;
}

bool mtk::Grad1D::ComputeWeights() {

  // Matrix to copmpute the weights as in the CRSA.
  mtk::DenseMatrix pi(num_bndy_coeffs_, num_bndy_coeffs_ - 1);


  // Assemble the pi matrix using:
  // 1. The collection of scaled preliminary approximations.
  // 2. The collection of coefficients approximating at the interior.
  // 3. The scaled basis for the null-space.

  // 1.1. Process array of scaled preliminary approximations.

  // These are queued in scaled_solutions. Each one of these, will be a column
  // of the pi matrix:
```

```
  for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
    for (auto jj = 0; jj < num_bndy_approxs_; ++jj) {
      pi.data()[ii*(2*(num_bndy_approxs_ - 1) + (order_accuracy_/2 + 1)) + jj] =
        prem_apps_[ii*num_bndy_approxs_ + jj];
    }
  }

  // 1.2. Add columns from known stencil approximating at the interior.

  // However, these must be padded by zeros, according to their position in the
  // final pi matrix:
  auto mm = 1;
  for (auto jj = num_bndy_approxs_; jj < order_accuracy_; ++jj) {
    for (auto ii = 0; ii < order_accuracy_; ++ii) {
      auto de = (ii + mm)*(2*(num_bndy_approxs_ - 1) +
        (order_accuracy_/2 + 1)) + jj;
      pi.data()[de] = coeffs_interior_[ii];
    }
    ++mm;
  }

  rat_basis_null_space_.OrderColMajor();

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "Rational basis for the null-space (col. major):" << std::endl;
  std::cout << rat_basis_null_space_ << std::endl;
  #endif

  // 1.3. Add final set of columns: rational basis for null-space.
  for (auto jj = dim_null_ + (order_accuracy_/2 + 1);
       jj < num_bndy_coeffs_ - 1; ++jj) {
    for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
      auto og =
        (jj - (dim_null_ + (order_accuracy_/2 + 1)))*num_bndy_coeffs_ + ii;
      auto de = ii*(2*dim_null_ + (order_accuracy_/2 + 1)) + jj;
      pi.data()[de] = rat_basis_null_space_.data()[og];
    }
  }

  #if MTK_DEBUG_LEVEL >0
  std::cout << "coeffs_interior_ =" << std::endl;
  for (auto ii = 0; ii < order_accuracy_; ++ii) {
    std::cout << std::setw(12) << coeffs_interior_[ii];
  }
  std::cout << std::endl << std::endl;
  #endif

  #if MTK_DEBUG_LEVEL >0
  std::cout << "Constructed pi matrix for CRS Algorithm: " << std::endl;
  std::cout << pi << std::endl;
  #endif


  // This imposes the mimetic condition.

  mtk::Real *hh{};  // Right-hand side to compute weights in the C{R,B}SA.

  try {
    hh = new mtk::Real[num_bndy_coeffs_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(hh, mtk::kZero, sizeof(hh[0])*num_bndy_coeffs_);

  hh[0] = -mtk::kOne;
  for (auto ii = (order_accuracy_/2 + 2 - 1); ii < num_bndy_coeffs_; ++ii) {
    auto aux_xx = mtk::kZero;
    for (auto jj = 0; jj < ((ii - (order_accuracy_/2 - 1)) - 1); ++jj) {
      aux_xx += coeffs_interior_[jj];
    }
    hh[ii] = -mtk::kOne*aux_xx;
  }


  // That is, we construct a system, to solve for the weights.

  // Once again we face the challenge of solving with LAPACK. However, for the
  // CRSA, this matrix PI is over-determined, since it has more rows than
  // unknowns. However, according to the theory, the solution to this system is
```

```
  // unique. We will use dgels_.

  try {
    weights_cbs_ = new mtk::Real[num_bndy_coeffs_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(weights_cbs_, mtk::kZero, sizeof(weights_cbs_[0])*num_bndy_coeffs_);

  int weights_ld{pi.num_cols() + 1};

  // Preserve hh.
  std::copy(hh, hh + weights_ld, weights_cbs_);

  pi.Transpose();

  int info{
    mtk::LAPACKAdapter::SolveRectangularDenseSystem(pi,
                                                    weights_cbs_, weights_ld)
  };

  #if MTK_DEBUG_LEVEL > 0
  if (!info) {
    std::cout << "System successfully solved!" << std::endl << std::endl;
  } else {
    std::cerr << "Error solving system! info = " << info << std::endl;
  }
  #endif

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "hh =" << std::endl;
  for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
    std::cout << std::setw(11) << hh[ii] << std::endl;
  }
  std::cout << std::endl;
  #endif

  // Preserve the original weights for research.

  try {
    weights_crs_ = new mtk::Real[num_bndy_coeffs_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(weights_crs_, mtk::kZero, sizeof(weights_crs_[0])*num_bndy_coeffs_);

  std::copy(weights_cbs_, weights_cbs_ + (weights_ld - 1), weights_crs_);

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "weights_CRSA + lambda =" << std::endl;
  for (auto ii = 0; ii < weights_ld - 1; ++ii) {
    std::cout << std::setw(12) << weights_crs_[ii] << std::endl;
  }
  std::cout << std::endl;
  #endif

  if (order_accuracy_ >= mtk::kCriticalOrderAccuracyGrad) {

    int minrow_{std::numeric_limits<int>::infinity()};

    mtk::Real norm{mtk::BLASAdapter::RealNRM2(weights_cbs_,
order_accuracy_)};
    mtk::Real minnorm{std::numeric_limits<mtk::Real>::infinity()};


    mtk::DenseMatrix phi(order_accuracy_ + 1, order_accuracy_);

    // 6.1. Insert preliminary approximations to first set of columns.

    for (auto ii = 0; ii < order_accuracy_ + 1; ++ii) {
      for (auto jj = 0; jj < num_bndy_approxs_; ++jj) {
        phi.data()[ii*(order_accuracy_) + jj] =
          prem_apps_[ii*num_bndy_approxs_ + jj];
      }
    }

    // 6.2. Skip a column and negate preliminary approximations.
```

```
  for (auto jj = 0; jj < order_accuracy_ + 1; jj++) {
    for (auto ii = 1; ii < num_bndy_approxs_; ii++) {
      auto de = (ii+ order_accuracy_ - num_bndy_approxs_+ jj*order_accuracy_);
      auto og = (num_bndy_approxs_ - ii + (jj)*num_bndy_approxs_);
      phi.data()[de] = -prem_apps_[og];
    }
  }

  // 6.3. Flip negative columns up-down.

  for (auto ii = 0; ii < order_accuracy_/2; ii++) {
    for (auto jj = num_bndy_approxs_ + 1; jj < order_accuracy_; jj++) {
      auto aux = phi.data()[ii*order_accuracy_ + jj];
      phi.data()[ii*order_accuracy_ + jj] =
        phi.data()[(order_accuracy_ - ii)*order_accuracy_ + jj];
      phi.data()[(order_accuracy_ - ii)*order_accuracy_ + jj] = aux;
    }
  }

  // 6.4. Insert stencil.

  auto mm = 0;
  for (auto jj = num_bndy_approxs_; jj < num_bndy_approxs_ +  1; jj++) {
    for (auto ii = 0; ii < order_accuracy_ + 1; ii++) {
      if (ii == 0) {
        phi.data()[jj] = 0.0;
      } else {
        phi.data()[(ii + mm)*order_accuracy_ + jj] = coeffs_interior_[ii - 1];
      }
    }
    mm++;
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "phi =" << std::endl;
  std::cout << phi << std::endl;
  #endif


  mtk::Real *lamed{};  // Used to build big lambda.

  try {
    lamed = new mtk::Real[num_bndy_approxs_ - 1];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(lamed, mtk::kZero, sizeof(lamed[0])*(num_bndy_approxs_ - 1));

  for (auto ii = 0; ii < num_bndy_approxs_ - 1; ++ii) {
    lamed[ii] = hh[ii + order_accuracy_ + 1] ;
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "lamed =" << std::endl;
  for (auto ii = 0; ii < num_bndy_approxs_ - 1; ++ii) {
    std::cout << std::setw(12) << lamed[ii] << std::endl;
  }
  std::cout << std::endl;
  #endif

  for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
    mtk::Real temp = mtk::kZero;
    for(auto jj = 0; jj < num_bndy_approxs_ - 1; ++jj) {
      temp = temp +
        lamed[jj]*rat_basis_null_space_.data()[jj*num_bndy_coeffs_ + ii];
    }
    hh[ii] = hh[ii] - temp;
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "big_lambda =" << std::endl;
  for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
    std::cout << std::setw(12) << hh[ii] << std::endl;
  }
  std::cout << std::endl;
  #endif

```

```
    int copy_result{};  // Should I replace the solution... not for now.

    mtk::Real normerr_; // Norm of the error for the solution on each row.

    for(auto row_= 0; row_ < order_accuracy_ + 1; ++row_) {
      normerr_ = mtk::GLPKAdapter::SolveSimplexAndCompare(phi.data(),
                                                          order_accuracy_ + 1,
                                                          order_accuracy_,
                                                          order_accuracy_,
                                                          hh,
                                                          weights_cbs_,
                                                          row_,
                                                          mimetic_threshold_,
                                                          copy_result);
      mtk::Real aux{normerr_/norm};

      #if MTK_DEBUG_LEVEL>0
      std::cout << "Relative norm: " << aux << " " << std::endl;
      std::cout << std::endl;
      #endif

      if (aux < minnorm) {
        minnorm = aux;
        minrow_= row_;
      }
    }

    #if MTK_DEBUG_LEVEL > 0
    std::cout << "weights_CBSA + lambda (after brute force search):" <<
      std::endl;
    for (auto ii = 0; ii < num_bndy_coeffs_ - 1; ++ii) {
      std::cout << std::setw(12) << weights_cbs_[ii] << std::endl;
    }
    std::cout << std::endl;
    #endif


    // After we know which row yields the smallest relative norm that row is
    // chosen to be the objective function and the result of the optimizer is
    // chosen to be the new weights_.

    #if MTK_DEBUG_LEVEL > 0
    std::cout << "Minimum Relative Norm " << minnorm << " found at row " <<
      minrow_ + 1 << std::endl;
    std::cout << std::endl;
    #endif

    copy_result = 1;
    normerr_ = mtk::GLPKAdapter::SolveSimplexAndCompare(phi.data(),
                                                        order_accuracy_ + 1,
                                                        order_accuracy_,
                                                        order_accuracy_,
                                                        hh,
                                                        weights_cbs_,
                                                        minrow_,
                                                        mimetic_threshold_,
                                                        copy_result);
    mtk::Real aux_{normerr_/norm};
    #if MTK_DEBUG_LEVEL > 0
    std::cout << "Relative norm: " << aux_ << std::endl;
    std::cout << std::endl;
    #endif

    delete [] lamed;
    lamed = nullptr;
  }

  delete [] hh;
  hh = nullptr;

  return true;
}

bool mtk::Grad1D::ComputeStencilBoundaryGrid(void) {

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "weights_* + lambda =" << std::endl;
  for (auto ii = 0; ii < num_bndy_coeffs_ - 1; ++ii) {
    std::cout << std::setw(12) << weights_cbs_[ii] << std::endl;
```

```
  }
  std::cout << std::endl;
  #endif


  mtk::Real *lambda{}; // Collection of bottom values from weights_.

  try {
    lambda = new mtk::Real[dim_null_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(lambda, mtk::kZero, sizeof(lambda[0])*dim_null_);

  for (auto ii = 0; ii < dim_null_; ++ii) {
    lambda[ii] = weights_cbs_[order_accuracy_ + ii];
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "lambda =" << std::endl;
  for (auto ii = 0; ii < dim_null_; ++ii) {
    std::cout << std::setw(12) << lambda[ii] << std::endl;
  }
  std::cout << std::endl;
  #endif


  mtk::Real *alpha{}; // Collection of alpha values.

  try {
    alpha = new mtk::Real[dim_null_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(alpha, mtk::kZero, sizeof(alpha[0])*dim_null_);

  for (auto ii = 0; ii < dim_null_; ++ii) {
    alpha[ii] = lambda[ii]/weights_cbs_[ii] ;
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "alpha =" << std::endl;
  for (auto ii = 0; ii < dim_null_; ++ii) {
    std::cout << std::setw(12) << alpha[ii] << std::endl;
  }
  std::cout << std::endl;
  #endif


  try {
    mim_bndy_ = new mtk::Real[num_bndy_coeffs_*num_bndy_approxs_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(mim_bndy_,
         mtk::kZero,
         sizeof(mim_bndy_[0])*num_bndy_coeffs_*num_bndy_approxs_);

  for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
    for (auto jj = 0; jj < (num_bndy_approxs_ - 1); ++jj) {
      mim_bndy_[ii*num_bndy_approxs_ + jj] =
        prem_apps_[ii*num_bndy_approxs_ + jj] +
        alpha[jj]*rat_basis_null_space_.data()[jj*num_bndy_coeffs_ + ii];
    }
  }

  for(auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
    mim_bndy_[ii*num_bndy_approxs_ + (num_bndy_approxs_ - 1)] =
      prem_apps_[ii*num_bndy_approxs_ + (num_bndy_approxs_ - 1)];
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "Collection of mimetic approximations:" << std::endl;
  for (auto ii = 0; ii < num_bndy_coeffs_; ++ii) {
    for (auto jj = 0; jj < num_bndy_approxs_; ++jj) {
```

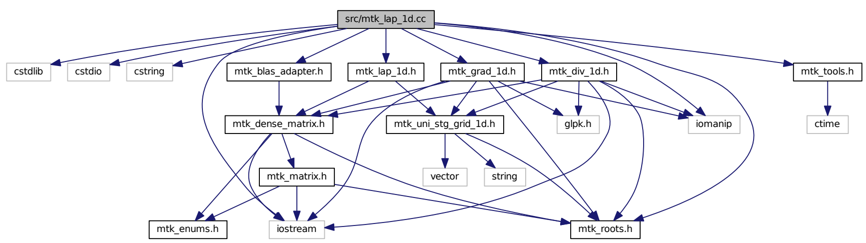
```
      std::cout << std::setw(13) << mim_bndy_[ii*num_bndy_approxs_ + jj];
    }
    std::cout << std::endl;
  }
  std::cout << std::endl;
  #endif

  delete[] lambda;
  lambda = nullptr;

  delete[] alpha;
  alpha = nullptr;

  return true;
}

bool mtk::Grad1D::AssembleOperator(void) {

  // The output array will have this form:
  // 1. The first entry of the array will contain the used order kk.
  // 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 a collection of weights.
  // 4. The next dim_null - 1 entries will contain the collections of
  // approximating coefficients for the west boundary of the grid.

  gradient_length_ = 1 + order_accuracy_ + order_accuracy_ +
    num_bndy_approxs_*num_bndy_coeffs_;

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "gradient_length_ = " << gradient_length_ << std::endl;
  #endif

  try {
    gradient_ = new mtk::Real[gradient_length_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(gradient_, mtk::kZero, sizeof(gradient_[0])*gradient_length_);


  gradient_[0] = order_accuracy_;


  for (auto ii = 0; ii < order_accuracy_; ++ii) {
    gradient_[ii + 1] = coeffs_interior_[ii];
  }


  for (auto ii = 0; ii < order_accuracy_; ++ii) {
    gradient_[(order_accuracy_ + 1) + ii] = weights_cbs_[ii];
  }


  int offset{2*order_accuracy_ + 1};

  int aux {}; // Auxiliary variable.

  if (order_accuracy_ > mtk::kDefaultOrderAccuracy) {
    for (auto ii = 0; ii < num_bndy_approxs_ ; ii++) {
      for (auto jj = 0; jj < num_bndy_coeffs_; jj++) {
        gradient_[offset + aux] = mim_bndy_[jj*num_bndy_approxs_ + ii];
        aux++;
      }
    }
  } else {
    gradient_[offset + 0] = prem_apps_[0];
    gradient_[offset + 1] = prem_apps_[1];
    gradient_[offset + 2] = prem_apps_[2];
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "1D " << order_accuracy_ << "-order grad built!" << std::endl;
  std::cout << std::endl;
  #endif

  return true;
}
```

17.49 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)

17.49.1 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.

17.50 mtk_lap_1d.cc

```
/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#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"

namespace mtk {

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


  stream << "laplacian_[0] = " << in.laplacian_[0] << std::endl << std::endl;



  stream << "laplacian_[1:" << 2*in.order_accuracy_ - 1 << "] = " <<
    std::endl << std::endl;
  for (auto ii = 1; ii <= (2*in.order_accuracy_ - 1); ++ii) {
    stream << std::setw(13) << in.laplacian_[ii] << " ";
  }
  stream << std::endl << std::endl;


  auto offset = 1 + (2*in.order_accuracy_ - 1);

  stream << "laplacian_[" << offset << ":" << offset +
    (in.order_accuracy_ - 1)*(2*in.order_accuracy_) - 1 << "] = " <<
    std::endl << std::endl;
```

```

  for (auto ii = 0; ii < in.order_accuracy_ - 1; ++ii) {
    for (auto jj = 0; jj < 2*in.order_accuracy_; ++jj) {
      stream << std::setw(13) <<
        in.laplacian_[offset + ii*(2*in.order_accuracy_) + jj];
    }
    stream << std::endl;
  }

  return stream;
}
}

mtk::Lap1D::Lap1D():
  order_accuracy_(mtk::kDefaultOrderAccuracy),
  laplacian_length_(),
  mimetic_threshold_(mtk::kDefaultMimeticThreshold) {}

mtk::Lap1D::~Lap1D() {

  delete [] laplacian_;
  laplacian_ = nullptr;
}

bool mtk::Lap1D::ConstructLap1D(int order_accuracy,
                                mtk::Real mimetic_threshold) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(order_accuracy < 2, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent((order_accuracy%2) != 0, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent(mimetic_threshold <= mtk::kZero,
                      __FILE__, __LINE__, __func__);

  if (order_accuracy >= mtk::kCriticalOrderAccuracyDiv) {
    std::cout << "WARNING: Numerical accuracy is high." << std::endl;
  }

  std::cout << "order_accuracy_ = " << order_accuracy << std::endl;
  std::cout << "mimetic_threshold_ = " << mimetic_threshold << std::endl;
  #endif

  order_accuracy_ = order_accuracy;
  mimetic_threshold_ = mimetic_threshold;


  mtk::Grad1D grad; // Mimetic gradient.

  bool info = grad.ConstructGrad1D(order_accuracy_, mimetic_threshold_);

  if (!info) {
    std::cerr << "Mimetic grad could not be built." << std::endl;
    return false;
  }


  mtk::Div1D div; // Mimetic divergence.

  info = div.ConstructDiv1D(order_accuracy_, mimetic_threshold_);

  if (!info) {
    std::cerr << "Mimetic div could not be built." << std::endl;
    return false;
  }


  // Since these are mimetic operator, we must multiply the matrices arising
  // from both the divergence and the Laplacian, in order to get the
  // approximating coefficients for the Laplacian operator.

  // However, we must choose a grid that implied a step size of 1, so to get
  // the approximating coefficients, without being affected from the
  // normalization with respect to the grid.

  // Also, the grid must be of the minimum size to support the requested order
  // of accuracy. We must please the divergence.

  mtk::UniStgGrid1D aux(mtk::kZero,
                        (mtk::Real) 3*order_accuracy_ - 1,
                        3*order_accuracy_ - 1);

  #if MTK_DEBUG_LEVEL > 0
```

```
  std::cout << "aux =" << std::endl;
  std::cout << aux << std::endl;
  std::cout <<"aux.delta_x() = " << aux.delta_x() << std::endl;
  std::cout << std::endl;
  #endif

  mtk::DenseMatrix grad_m(grad.ReturnAsDenseMatrix(aux));

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "grad_m =" << std::endl;
  std::cout << grad_m << std::endl;
  #endif

  mtk::DenseMatrix div_m(div.ReturnAsDenseMatrix(aux));

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "div_m =" << std::endl;
  std::cout << div_m << std::endl;
  #endif


  mtk::DenseMatrix lap; // Laplacian matrix to hold to computed coefficients.

  lap = mtk::BLASAdapter::RealDenseMM(div_m, grad_m);

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "lap =" << std::endl;
  std::cout << lap << std::endl;
  #endif



  // The output array will have this form:
  // 1. The first entry of the array will contain the used order kk.
  // 2. The second entry of the array will contain the collection of
  // approximating coefficients for the interior of the grid.
  // 3. The next entries will contain the collections of approximating
  // coefficients for the west boundary of the grid.

  laplacian_length_ = 1 + (2*order_accuracy_ - 1) +
    (order_accuracy_ - 1)*(2*order_accuracy_);

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "laplacian_length_ = " << laplacian_length_ << std::endl;
  std::cout << std::endl;
  #endif

  try {
    laplacian_ = new mtk::Real[laplacian_length_];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(laplacian_, mtk::kZero, sizeof(laplacian_[0])*laplacian_length_);


  laplacian_[0] = order_accuracy_;


  for (auto ii = 0; ii < 2*order_accuracy_ - 1; ++ii) {
    laplacian_[ii + 1] = lap.GetValue(1 + (order_accuracy_ - 1), ii + 1);
  }


  auto offset = 1 + (2*order_accuracy_ - 1);

  for (auto ii = 0; ii < order_accuracy_ - 1; ++ii) {
    for (auto jj = 0; jj < 2*order_accuracy_; ++jj) {
      laplacian_[offset + ii*(2*order_accuracy_) + jj] =
        lap.GetValue(1 + ii,jj);
    }
  }

  return true;
}

mtk::DenseMatrix mtk::Lap1D::ReturnAsDenseMatrix(const
UniStgGrid1D &grid) {

  int nn{grid.num_cells_x()};  // Number of cells on the grid.
```

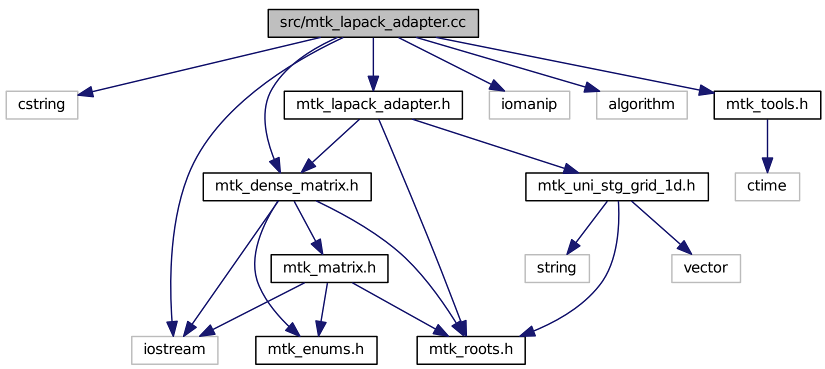
```
  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(nn <= 0, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent(nn < 3*order_accuracy_ - 1, __FILE__, __LINE__, __func__);
  #endif

  mtk::DenseMatrix lap(nn + 2, nn + 2); // Laplacian matrix to be returned.

  mtk::Real idx{mtk::kOne/(grid.delta_x()*grid.delta_x())}; // Inverse of dx^2.


  auto offset = (1 + 2*order_accuracy_ - 1);

  for (auto ii = 0; ii < order_accuracy_ - 1; ++ii) {
    for (auto jj = 0; jj < 2*order_accuracy_; ++jj) {
      lap.SetValue(1 + ii,
                   jj,
                   idx*laplacian_[offset + ii*2*order_accuracy_ + jj]);
    }
  }


  offset = 1 + (order_accuracy_ - 1);

  int kk{1};
  for (auto ii = order_accuracy_; ii <= nn - (order_accuracy_ - 1); ++ii) {
    int mm{1};
    for (auto jj = 0; jj < 2*order_accuracy_ - 1; ++jj) {
      lap.SetValue(ii, jj + kk, idx*laplacian_[mm]);
      mm = mm + 1;
    }
    kk = kk + 1;
  }


  offset = (1 + 2*order_accuracy_ - 1);

  auto aux = order_accuracy_ + (nn - 2*(order_accuracy_ - 1));

  auto ll = 1;
  auto rr = 1;
  for (auto ii = nn; ii > aux - 1; --ii) {
    auto cc = 0;
    for (auto jj = nn + 2 - 1; jj >= (nn + 2) - 2*order_accuracy_; --jj) {
      lap.SetValue(ii, jj, lap.GetValue(rr,cc));
      ++ll;
      ++cc;
    }
    rr++;
  }


  return lap;
}

mtk::Real* mtk::Lap1D::Data(const UniStgGrid1D &grid) {

  mtk::DenseMatrix tmp;

  tmp = ReturnAsDenseMatrix(grid);

  return tmp.data();
}
```

17.51 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.

17.51.1 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** 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.cc.

17.52 mtk_lapack_adapter.cc

```

/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#include <cstring>

#include <iostream>
#include <iomanip>

#include <algorithm>

#include "mtk_tools.h"
#include "mtk_dense_matrix.h"
#include "mtk_lapack_adapter.h"

namespace mtk {

extern "C" {
```

```
#ifdef MTK_PRECISION_DOUBLE

void dgesv_(int* n,
            int* nrhs,
            Real* a,
            int* lda,
            int* ipiv,
            Real* b,
            int* ldb,
            int* info);
#else

void sgesv_(int* n,
            int* nrhs,
            Real* a,
            int* lda,
            int* ipiv,
            Real* b,
            int* ldb,
            int* info);
#endif

#ifdef MTK_PRECISION_DOUBLE

void dgels_(char* trans,
            int* m,
            int* n,
            int* nrhs,
            Real* a,
            int* lda,
            Real* b,
            int* ldb,
            Real* work,
            int* lwork,
            int* info);
#else

void sgels_(char* trans,
            int* m,
            int* n,
            int* nrhs,
            Real* a,
            int* lda,
            Real* b,
            int* ldb,
            Real* work,
            int* lwork,
            int* info);
#endif

#ifdef MTK_PRECISION_DOUBLE

void dgeqrf_(int *m,
             int *n,
             Real *a,
             int *lda,
             Real *tau,
             Real *work,
             int *lwork,
             int *info);
#else

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

#ifdef MTK_PRECISION_DOUBLE

void dormqr_(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);
#else

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);
#endif
}
}

int mtk::LAPACKAdapter::SolveDenseSystem(
mtk::DenseMatrix &mm,
                                          mtk::Real *rhs) {


  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(rhs == nullptr, __FILE__, __LINE__, __func__);
  #endif

  int *ipiv{};                  // Array for pivoting information.
  int nrhs{1};                  // Number of right-hand sides.
  int info{};                   // Status of the solution.
  int mm_rank{mm.num_rows()}; // Rank of the matrix.

  try {
    ipiv = new int[mm_rank];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(ipiv, 0, sizeof(ipiv[0])*mm_rank);

  int ldbb = mm_rank;
  int mm_ld = mm_rank;

  #ifdef MTK_PRECISION_DOUBLE
  dgesv_(&mm_rank, &nrhs, mm.data(), &mm_ld, ipiv, rhs, &ldbb, &info);
  #else
  fgesv_(&mm_rank, &nrhs, mm.data(), &mm_ld, ipiv, rhs, &ldbb, &info);
  #endif

  delete [] ipiv;

  return info;
}

int mtk::LAPACKAdapter::SolveDenseSystem(
mtk::DenseMatrix &mm,
                                          mtk::DenseMatrix &bb) {

  int nrhs{bb.num_rows()};  // Number of right-hand sides.

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(nrhs <= 0, __FILE__, __LINE__, __func__);
  #endif

  int *ipiv{};                  // Array for pivoting information.
  int info{};                   // Status of the solution.
  int mm_rank{mm.num_rows()}; // Rank of the matrix.

  try {
    ipiv = new int[mm_rank];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
```

```
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(ipiv, 0, sizeof(ipiv[0])*mm_rank);

  int ldbb = mm_rank;
  int mm_ld = mm_rank;

  #ifdef MTK_PRECISION_DOUBLE
  dgesv_(&mm_rank, &nrhs, mm.data(), &mm_ld, ipiv, bb.data(), &ldbb, &info);
  #else
  fgesv_(&mm_rank, &nrhs, mm.data(), &mm_ld, ipiv, bb.data(), &ldbb, &info);
  #endif

  delete [] ipiv;

  // After output, the data in the matrix will be column-major ordered.

  bb.SetOrdering(mtk::COL_MAJOR);

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "bb_col_maj_ord =" << std::endl;
  std::cout << bb << std::endl;
  #endif

  bb.OrderRowMajor();

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "bb_row_maj_ord =" << std::endl;
  std::cout << bb << std::endl;
  #endif

  return info;
}

int mtk::LAPACKAdapter::SolveDenseSystem(
mtk::DenseMatrix &mm,
                                         mtk::UniStgGrid1D &rhs) {

  int nrhs{1};  // Number of right-hand sides.

  int *ipiv{};                // Array for pivoting information.
  int info{};                 // Status of the solution.
  int mm_rank{mm.num_rows()}; // Rank of the matrix.

  try {
    ipiv = new int[mm_rank];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(ipiv, 0, sizeof(ipiv[0])*mm_rank);

  int ldbb = mm_rank;
  int mm_ld = mm_rank;

  mm.OrderColMajor();

  #ifdef MTK_PRECISION_DOUBLE
  dgesv_(&mm_rank, &nrhs, mm.data(), &mm_ld, ipiv,
         rhs.discrete_field_u(), &ldbb, &info);
  #else
  fgesv_(&mm_rank, &nrhs, mm.data(), &mm_ld, ipiv,
         rhs.discrete_field_u(), &ldbb, &info);
  #endif

  mm.OrderRowMajor();

  delete [] ipiv;

  return info;
}

mtk::DenseMatrix mtk::LAPACKAdapter::QRFactorDenseMatrix
 (mtk::DenseMatrix &aa) {

  mtk::Real *work{}; // Working array.
  mtk::Real *tau{};  // Array for the Householder scalars.

  // Prepare to factorize: allocate and inquire for the value of lwork.
```

```
  try {
    work = new mtk::Real[1];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(work, mtk::kZero, sizeof(aa.data()[0])*1);

  int lwork{-1};
  int info{};

  int aa_num_cols = aa.num_cols();
  int aaT_num_rows = aa.num_cols();
  int aaT_num_cols = aa.num_rows();

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "Input matrix BEFORE QR factorization:" << std::endl;
  std::cout << aa << std::endl;
  #endif

  #ifdef MTK_PRECISION_DOUBLE
  dgeqrf_(&aaT_num_rows, &aaT_num_cols, aa.data(), &aaT_num_rows,
          tau,
          work, &lwork, &info);
  #else
  fgeqrf_(&aaT_num_rows, &aaT_num_cols, aa.data(), &aaT_num_rows,
          tau,
          work, &lwork, &info);
  #endif

  #if MTK_DEBUG_LEVEL > 0
  if (info == 0) {
    lwork = (int) work[0];
  } else {
    std::cerr << "Could not get value for lwork on line " << __LINE__ - 5 <<
      std::endl;
    std::cerr << "Exiting..." << std::endl;
  }
  #endif

  #if MTK_DEBUG_LEVEL>0
  std::cout << "lwork = " << std::endl << std::setw(12) << lwork << std::endl
    << std::endl;
  #endif

  delete [] work;
  work = nullptr;

  // Once we know lwork, we can actually invoke the factorization:
  try {
    work = new mtk::Real [lwork];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(work, mtk::kZero, sizeof(work[0])*lwork);

  int ltau = std::min(aaT_num_rows,aaT_num_cols);

  try {
    tau = new mtk::Real [ltau];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(tau, mtk::kZero, sizeof(0.0)*ltau);

  #ifdef MTK_PRECISION_DOUBLE
  dgeqrf_(&aaT_num_rows, &aaT_num_cols, aa.data(), &aaT_num_rows,
          tau, work, &lwork, &info);
  #else
  fgeqrf_(&aaT_num_rows, &aaT_num_cols, aa.data(), &aaT_num_rows,
          tau, work, &lwork, &info);
  #endif

  if (!info) {
    #if MTK_DEBUG_LEVEL > 0
    std::cout << "QR factorization completed!" << std::endl << std::endl;
```

```
    #endif
  } else {
    std::cerr << "Error solving system! info = " << info << std::endl;
    std::cerr << "Exiting..." << std::endl;
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "Input matrix AFTER QR factorization:" << std::endl;
  std::cout << aa << std::endl;
  #endif

  // We now generate the real matrix Q with orthonormal columns. This has to
  // be done separately since the actual output of dgeqrf_ (AA_) represents
  // the orthogonal matrix Q as a product of min(aa_num_rows,aa_num_cols)
  // elementary Householder reflectors. Notice that we must re-inquire the new
  // value for lwork that is used.

  bool padded{false};

  bool transpose{false};

  mtk::DenseMatrix QQ_(aa.num_cols(),padded,transpose);

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "Initialized QQ_T: " << std::endl;
  std::cout << QQ_ << std::endl;
  #endif

  // Assemble the QQ_ matrix:
  lwork = -1;

  delete[] work;
  work = nullptr;

  try {
    work = new mtk::Real[1];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
    std::cerr << memory_allocation_exception.what() <<
      std::endl;
  }
  memset(work, mtk::kZero, sizeof(work[0])*1);

  char side_{'L'};
  char trans_{'N'};

  int aux = QQ_.num_rows();

  #ifdef MTK_PRECISION_DOUBLE
  dormqr_(&side_, &trans_,
          &aa_num_cols, &aa_num_cols, &ltau, aa.data(), &aaT_num_rows, tau,
          QQ_.data(), &aux, work, &lwork, &info);
  #else
  formqr_(&side_, &trans_,
          &aa_num_cols, &aa_num_cols, &ltau, aa.data(), &aaT_num_rows, tau,
          QQ_.data(), &aux, work, &lwork, &info);
  #endif

  #if MTK_DEBUG_LEVEL > 0
  if (info == 0) {
    lwork = (int) work[0];
  } else {
    std::cerr << "Could not get lwork on line " << __LINE__ - 5 << std::endl;
    std::cerr << "Exiting..." << std::endl;
  }
  #endif

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "lwork = " << std::endl << std::setw(12) << lwork <<
    std::endl << std::endl;
  #endif

  delete[] work;
  work = nullptr;

  try {
    work = new mtk::Real[lwork];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 <<
      std::endl;
```

```
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(work, mtk::kZero, sizeof(work[0])*lwork);

  #ifdef MTK_PRECISION_DOUBLE
  dormqr_(&side_, &trans_,
          &aa_num_cols, &aa_num_cols, &ltau, aa.data(), &aaT_num_rows, tau,
          QQ_.data(), &aux, work, &lwork, &info);
  #else
  formqr_(&side_, &trans_,
          &aa_num_cols, &aa_num_cols, &ltau, aa.data(), &aaT_num_rows, tau,
          QQ_.data(), &aux, work, &lwork, &info);
  #endif

  if (!info) {
    #if MTK_DEBUG_LEVEL>0
    std::cout << "Q matrix successfully assembled!" << std::endl << std::endl;
    #endif
  } else {
    std::cerr << "Something went wrong solving system! info = " << info <<
      std::endl;
    std::cerr << "Exiting..." << std::endl;
  }

  delete[] work;
  work = nullptr;

  delete[] tau;
  tau = nullptr;

  return QQ_;
}

int mtk::LAPACKAdapter::SolveRectangularDenseSystem(const
mtk::DenseMatrix &aa,
                                                     mtk::Real *ob_,
                                                     int ob_ld_) {

  // We first invoke the solver to query for the value of lwork. For this,
  // we must at least allocate enough space to allow access to WORK(1), or
  // work[0]:

  // If LWORK = -1, then a workspace query is assumed; the routine only
  // calculates the optimal size of the WORK array, returns this value as
  // the first entry of the WORK array, and no error message related to
  // LWORK is issued by XERBLA.

  mtk::Real *work{}; // Work array.

  try {
    work = new mtk::Real[1];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 << std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(work, mtk::kZero, sizeof(work[0])*1);

  char trans_{'N'};
  int nrhs_{1};
  int info{0};
  int lwork{-1};

  int AA_num_rows_  = aa.num_cols();
  int AA_num_cols_  = aa.num_rows();
  int AA_ld_ = std::max(1,aa.num_cols());

  #ifdef MTK_PRECISION_DOUBLE
  dgels_(&trans_, &AA_num_rows_, &AA_num_cols_, &nrhs_, aa.data(), &AA_ld_,
         ob_, &ob_ld_,
         work, &lwork, &info);
  #else
  sgels_(&trans_, &AA_num_rows_, &AA_num_cols_, &nrhs_, aa.data(), &AA_ld_,
         ob_, &ob_ld_,
         work, &lwork, &info);
  #endif

  if (info == 0) {
    lwork = (int) work[0];
  } else {
    std::cerr << "Could not get value for lwork on line " << __LINE__ - 2 <<
      std::endl;
```

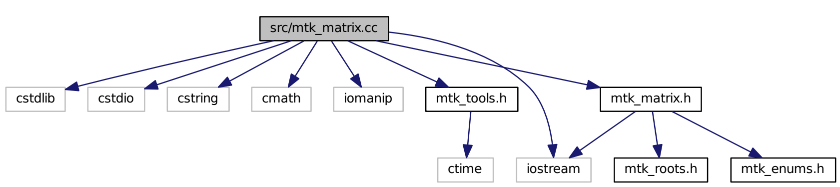
```
    std::cerr << "Exiting..." << std::endl;
    return info;
  }

  #if MTK_DEBUG_LEVEL > 0
  std::cout << "lwork = " << std::endl << std::setw(12)<< lwork <<
    std::endl << std::endl;
  #endif

  // We then use lwork's new value to create the work array:
  delete[] work;
  work = nullptr;

  try {
    work = new mtk::Real[lwork];
  } catch (std::bad_alloc &memory_allocation_exception) {
    std::cerr << "Memory allocation exception on line " << __LINE__ - 3 << std::endl;
    std::cerr << memory_allocation_exception.what() << std::endl;
  }
  memset(work, 0.0, sizeof(work[0])*lwork);

  // We now invoke the solver again:
  #ifdef MTK_PRECISION_DOUBLE
  dgels_(&trans_, &AA_num_rows_, &AA_num_cols_, &nrhs_, aa.data(), &AA_ld_,
         ob_, &ob_ld_,
         work, &lwork, &info);
  #else
  sgels_(&trans_, &AA_num_rows_, &AA_num_cols_, &nrhs_, aa.data(), &AA_ld_,
         ob_, &ob_ld_,
         work, &lwork, &info);
  #endif

  delete [] work;
  work = nullptr;

  return info;
}
```

17.53 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:



17.53.1 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.

17.54 mtk_matrix.cc

```

/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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

1. Modifications to the source code should be reported to:

esanchez at mail dot sdsu dot edu

A copy of the modified files should be reported once modifications are
completed. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#include <cstdlib>
#include <cstdio>
#include <cstring>
#include <cmath>

#include <iomanip>
#include <iostream>

#include "mtk_tools.h"
```

```
#include "mtk_matrix.h"

mtk::Matrix::Matrix():
  storage_(mtk::DENSE),
  ordering_(mtk::ROW_MAJOR),
  num_rows_(),
  num_cols_(),
  num_values_(),
  ld_(),
  num_zero_(),
  num_non_zero_(),
  num_null_(),
  num_non_null_(),
  kl_(),
  ku_(),
  bandwidth_(),
  abs_density_(),
  rel_density_(),
  abs_sparsity_(),
  rel_sparsity_() {}

mtk::Matrix::Matrix(const Matrix &in):
  storage_(in.storage_),
  ordering_(in.ordering_),
  num_rows_(in.num_rows_),
  num_cols_(in.num_cols_),
  num_values_(in.num_values_),
  ld_(in.ld_),
  num_zero_(in.num_zero_),
  num_non_zero_(in.num_non_zero_),
  num_null_(in.num_null_),
  num_non_null_(in.num_non_null_),
  kl_(in.kl_),
  ku_(in.ku_),
  bandwidth_(in.bandwidth_),
  abs_density_(in.abs_density_),
  rel_density_(in.rel_density_),
  abs_sparsity_(in.abs_sparsity_),
  rel_sparsity_(in.rel_sparsity_) {}

mtk::Matrix::~Matrix() {}

mtk::MatrixStorage mtk::Matrix::storage() const {

  return storage_;
}

mtk::MatrixOrdering mtk::Matrix::ordering() const {

  return ordering_;
}

int mtk::Matrix::num_rows() const {

  return num_rows_;
}

int mtk::Matrix::num_cols() const {

  return num_cols_;
}

int mtk::Matrix::num_values() const {

  return num_values_;
}

int mtk::Matrix::ld() const {

  return ld_;
}

int mtk::Matrix::num_zero() const {

  return num_zero_;
}

int mtk::Matrix::num_non_zero() const {

  return num_non_zero_;
}
```

```
int mtk::Matrix::num_null() const {

  return num_null_;
}

int mtk::Matrix::num_non_null() const {

  return num_non_null_;
}

int mtk::Matrix::kl() const {

  return kl_;
}

int mtk::Matrix::ku() const {

  return ku_;
}

int mtk::Matrix::bandwidth() const {

  return bandwidth_;
}

mtk::Real mtk::Matrix::rel_density() const {

  return rel_density_;
}

mtk::Real mtk::Matrix::abs_sparsity() const {

  return abs_sparsity_;
}

mtk::Real mtk::Matrix::rel_sparsity() const {

  return rel_sparsity_;
}

void mtk::Matrix::set_storage(const mtk::MatrixStorage &ss) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(!(ss == mtk::DENSE ||
                        ss == mtk::BANDED ||
                        ss == mtk::CRS),
                      __FILE__, __LINE__, __func__);
  #endif

  storage_ = ss;
}

void mtk::Matrix::set_ordering(const
mtk::MatrixOrdering &oo) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(!(oo == mtk::ROW_MAJOR || oo ==
mtk::COL_MAJOR),
                      __FILE__, __LINE__, __func__);
  #endif

  ordering_ = oo;

  ld_ = (ordering_ == mtk::ROW_MAJOR)?
    std::max(1,num_cols_): std::max(1,num_rows_);
}

void mtk::Matrix::set_num_rows(int in) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(in < 1, __FILE__, __LINE__, __func__);
  #endif

  num_rows_ = in;
  num_values_ = num_rows_*num_cols_;
  ld_ = (ordering_ == mtk::ROW_MAJOR)?
    std::max(1,num_cols_): std::max(1,num_rows_);
}

void mtk::Matrix::set_num_cols(int in) {
```

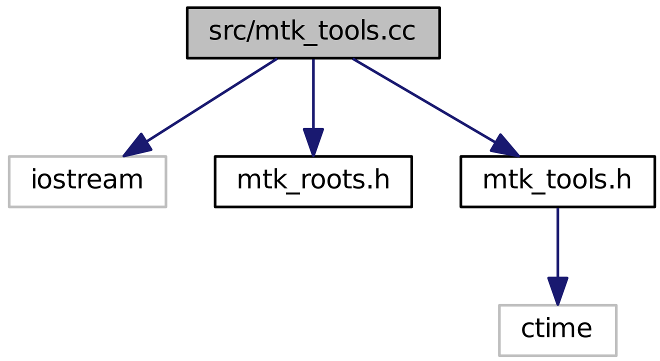
```

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(in < 1, __FILE__, __LINE__, __func__);
  #endif

  num_cols_ = in;
  num_values_ = num_rows_*num_cols_;
  ld_ = (ordering_ == mtk::ROW_MAJOR)?
    std::max(1,num_cols_): std::max(1,num_rows_);
}

void mtk::Matrix::set_num_zero(int in) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(in < 0, __FILE__, __LINE__, __func__);
  #endif

  num_zero_ = in;
  num_non_zero_ = num_values_ - num_zero_;

  rel_density_ = (mtk::Real) num_non_zero_/num_values_;
  rel_sparsity_ = 1.0 - rel_density_;
}

void mtk::Matrix::set_num_null(int in) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(in < 0, __FILE__, __LINE__, __func__);
  #endif

  num_null_ = in;
  num_non_null_ = num_values_ - num_null_;

  abs_density_ = (mtk::Real) num_non_null_/num_values_;
  abs_sparsity_ = 1.0 - abs_density_;
}

void mtk::Matrix::IncreaseNumZero() {


  num_zero_++;
  num_non_zero_ = num_values_ - num_zero_;
  rel_density_ = (mtk::Real) num_non_zero_/num_values_;
  rel_sparsity_ = 1.0 - rel_density_;
}

void mtk::Matrix::IncreaseNumNull() {


  num_null_++;
  num_non_null_ = num_values_ - num_null_;
  abs_density_ = (mtk::Real) num_non_null_/num_values_;
  abs_sparsity_ = 1.0 - abs_density_;
}
```

17.55 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:



17.55.1 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.

17.56 mtk_tools.cc

```

/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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
```

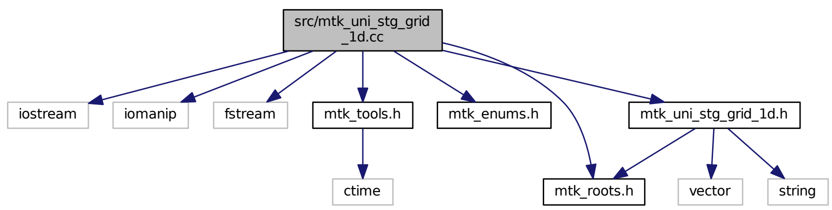
```
specific prior written permission.

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.
*/

#include <iostream>

#include "mtk_roots.h"
#include "mtk_tools.h"

void mtk::Tools::Prevent(const bool condition,
                         const char *fname,
                         int lineno,
                         const char *fxname) {


  #if MTK_DEBUG_LEVEL > 0
  if (lineno < 1) {
    std::cerr << __FILE__ << ": " << "Incorrect parameter at line " <<
    __LINE__ - 2 << " (" << __func__ << ")" << std::endl;
    exit(EXIT_FAILURE);
  }
  #endif

  if (condition) {
    std::cerr << fname << ": " << "Incorrect parameter at line " <<
    lineno << " (" << fxname << ")" << std::endl;
    exit(EXIT_FAILURE);
  }
}


int mtk::Tools::test_number_; // Used to control the correctness of the test.

clock_t mtk::Tools::begin_time_;  // Used to time tests.

void mtk::Tools::BeginTestNo(const int &nn) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(nn <= 0, __FILE__, __LINE__, __func__);
  #endif

  test_number_ = nn;

  std::cout << "Test " << nn << "..." << std::endl << std::endl;
  begin_time_ = clock();
}

void mtk::Tools::EndTestNo(const int &nn) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(test_number_ != nn, __FILE__, __LINE__, __func__);
  #endif

  auto duration = mtk::Real(clock() - begin_time_)/CLOCKS_PER_SEC;
  std::cout << "Test " << test_number_ << " complete! ";
  std::cout << "Elapsed: " << duration << " seconds." << std::endl;
}
```

17.57 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)

17.57.1 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.

17.58 mtk_uni_stg_grid_1d.cc

```

/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#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"

namespace mtk {

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

  stream << '[' << in.west_bndy_x_ << ':' << in.num_cells_x_ << ':' <<
  in.east_bndy_x_ << "] = " << std::endl << std::endl;


  stream << "x:";
  for (unsigned int ii = 0; ii < in.discrete_domain_x_.size(); ++ii) {
    stream << std::setw(10) << in.discrete_domain_x_[ii];
  }
  stream << std::endl;

  if (in.nature_ == mtk::SCALAR) {
    stream << "u:";
  }
  else {
    stream << "v:";
  }
  for (unsigned int ii = 0; ii < in.discrete_field_u_.size(); ++ii) {
    stream << std::setw(10) << in.discrete_field_u_[ii];
  }

  stream << std::endl;

  return stream;
}
}

mtk::UniStgGrid1D::UniStgGrid1D():
```

```
    nature_(),
    discrete_domain_x_(),
    discrete_field_u_(),
    west_bndy_x_(),
    east_bndy_x_(),
    num_cells_x_(),
    delta_x_() {}

mtk::UniStgGrid1D::UniStgGrid1D(const UniStgGrid1D &grid):
    nature_(grid.nature_),
    west_bndy_x_(grid.west_bndy_x_),
    east_bndy_x_(grid.east_bndy_x_),
    num_cells_x_(grid.num_cells_x_),
    delta_x_(grid.delta_x_) {

    std::copy(grid.discrete_domain_x_.begin(),
              grid.discrete_domain_x_.begin() + grid.discrete_domain_x_.size(),
              discrete_domain_x_.begin());

    std::copy(grid.discrete_field_u_.begin(),
              grid.discrete_field_u_.begin() + grid.discrete_field_u_.size(),
              discrete_field_u_.begin());
}

mtk::UniStgGrid1D::UniStgGrid1D(const Real &west_bndy_x,
                                const Real &east_bndy_x,
                                const int &num_cells_x,
                                const mtk::FieldNature &nature) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(west_bndy_x < mtk::kZero, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent(east_bndy_x < mtk::kZero, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent(east_bndy_x <= west_bndy_x, __FILE__, __LINE__, __func__);
  mtk::Tools::Prevent(num_cells_x < 0, __FILE__, __LINE__, __func__);
  #endif

  nature_ = nature;
  west_bndy_x_ = west_bndy_x;
  east_bndy_x_ = east_bndy_x;
  num_cells_x_ = num_cells_x;

  delta_x_ = (east_bndy_x - west_bndy_x)/((mtk::Real) num_cells_x);
}

mtk::UniStgGrid1D::~UniStgGrid1D() {}

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

  return delta_x_;
}

mtk::Real *mtk::UniStgGrid1D::discrete_domain_x() {

  return discrete_domain_x_.data();
}

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

  return discrete_field_u_.data();
}

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

  return num_cells_x_;
}

void mtk::UniStgGrid1D::BindScalarField(
    mtk::Real (*ScalarField)(mtk::Real xx)) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(nature_ == mtk::VECTOR, __FILE__, __LINE__, __func__);
  #endif


  discrete_domain_x_.reserve(num_cells_x_ + 2);

  discrete_domain_x_.push_back(west_bndy_x_);
  #ifdef MTK_PRECISION_DOUBLE
```

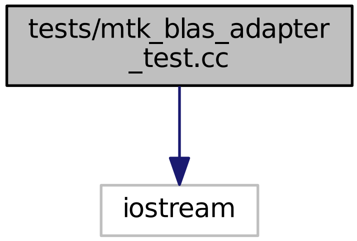
```
  auto first_center = west_bndy_x_ + delta_x_/2.0;
  #else
  auto first_center = west_bndy_x_ + delta_x_/2.0f;
  #endif
  discrete_domain_x_.push_back(first_center);
  for (auto ii = 1; ii < num_cells_x_; ++ii) {
    discrete_domain_x_.push_back(first_center + ii*delta_x_);
  }
  discrete_domain_x_.push_back(east_bndy_x_);


  discrete_field_u_.reserve(num_cells_x_ + 2);

  discrete_field_u_.push_back(ScalarField(west_bndy_x_));

  discrete_field_u_.push_back(ScalarField(first_center));
  for (auto ii = 1; ii < num_cells_x_; ++ii) {
    discrete_field_u_.push_back(ScalarField(first_center + ii*delta_x_));
  }
  discrete_field_u_.push_back(ScalarField(east_bndy_x_));
}

void mtk::UniStgGrid1D::BindVectorField(
    mtk::Real (*VectorField)(mtk::Real xx)) {

  #if MTK_DEBUG_LEVEL > 0
  mtk::Tools::Prevent(nature_ == mtk::SCALAR, __FILE__, __LINE__, __func__);
  #endif


  discrete_domain_x_.reserve(num_cells_x_ + 1);

  discrete_domain_x_.push_back(west_bndy_x_);
  for (auto ii = 1; ii < num_cells_x_; ++ii) {
    discrete_domain_x_.push_back(west_bndy_x_ + ii*delta_x_);
  }
  discrete_domain_x_.push_back(east_bndy_x_);


  discrete_field_u_.reserve(num_cells_x_ + 1);

  discrete_field_u_.push_back(VectorField(west_bndy_x_));
  for (auto ii = 1; ii < num_cells_x_; ++ii) {
    discrete_field_u_.push_back(VectorField(west_bndy_x_ + ii*delta_x_));
  }
  discrete_field_u_.push_back(VectorField(east_bndy_x_));
}

bool mtk::UniStgGrid1D::WriteToFile(std::string filename,
                                    std::string space_name,
                                    std::string field_name) {

  std::ofstream output_dat_file;  // Output file.

  output_dat_file.open(filename);

  if (!output_dat_file.is_open()) {
    return false;
  }

  output_dat_file << "# " << space_name << ' ' << field_name << std::endl;
  for (unsigned int ii = 0; ii < discrete_domain_x_.size(); ++ii) {
    output_dat_file << discrete_domain_x_[ii] << ' ' << discrete_field_u_[ii] <<
      std::endl;
  }

  output_dat_file.close();

  return true;
}
```

17.59 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 ()

17.59.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_blas_adapter_test.cc.

17.59.2 Function Documentation

17.59.2.1 int main ( )

Definition at line 107 of file mtk_blas_adapter_test.cc.

17.60 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. Documentation related to said modifications should be included.
00018
00019 2. Redistributions of source code must be done through direct
00020 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00021
00022 3. Redistributions of source code must retain the above copyright notice, this
00023 list of conditions and the following disclaimer.
```

```

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#if __cplusplus == 201103L

#include <iostream>

#include "mtk.h"

void Test1() {

  mtk::Tools::BeginTestNo(1);

  int rr = 2;
  int cc = 3;

  mtk::DenseMatrix aa(rr,cc);

  aa.SetValue(0,0,1.0);
  aa.SetValue(0,1,2.0);
  aa.SetValue(0,2,3.0);
  aa.SetValue(1,0,4.0);
  aa.SetValue(1,1,5.0);
  aa.SetValue(1,2,6.0);

  std::cout << aa << std::endl;

  mtk::DenseMatrix bb(cc,rr);

  bb.SetValue(0,0,7.0);
  bb.SetValue(0,1,8.0);
  bb.SetValue(1,0,9.0);
  bb.SetValue(1,1,10.0);
  bb.SetValue(2,0,11.0);
  bb.SetValue(2,1,12.0);

  std::cout << bb << std::endl;

  mtk::DenseMatrix pp = mtk::BLASAdapter::RealDenseMM(aa,bb);

  std::cout << pp << std::endl;

  mtk::Tools::EndTestNo(1);
}

int main () {

  std::cout << "Testing mtk::BLASAdapter class." << std::endl;

  Test1();
}

#else
#include <iostream>
```

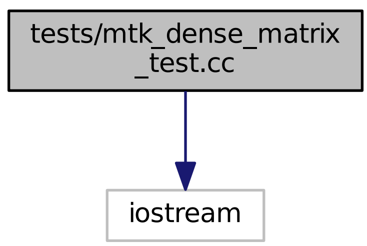
```
using std::cout;
using std::endl;
int main () {
  cout << "This code HAS to be compiled with support for C++11." << endl;
  cout << "Exiting..." << endl;
}
#endif
```

17.61 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 ()

17.61.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_dense_matrix_test.cc.

17.61.2 Function Documentation

17.61.2.1 int main ( )

Definition at line 285 of file mtk_dense_matrix_test.cc.

17.62 mtk_dense_matrix_test.cc

```
/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#if __cplusplus == 201103L

#include <iostream>
#include <ctime>

#include "mtk.h"

void Test1() {

  mtk::Tools::BeginTestNo(1);

  mtk::DenseMatrix m1;

  std::cout << m1 << std::endl;

  mtk::Tools::EndTestNo(1);
}

void Test2() {

  mtk::Tools::BeginTestNo(2);

  int rr = 4;
  int cc = 7;

  mtk::DenseMatrix m2(rr,cc);

  std::cout << m2 << std::endl;

  mtk::Tools::EndTestNo(2);
}

void Test3() {

  mtk::Tools::BeginTestNo(3);
```

```

  int rank = 5;
  bool padded = true;
  bool transpose = false;

  mtk::DenseMatrix m3(rank,padded,transpose);

  std::cout << m3 << std::endl;

  mtk::Tools::EndTestNo(3);
}

void Test4() {

  mtk::Tools::BeginTestNo(4);

  int rank = 5;
  bool padded = false;
  bool transpose = false;

  mtk::DenseMatrix m4(rank,padded,transpose);

  std::cout << m4 << std::endl;

  mtk::Tools::EndTestNo(4);
}

void Test5() {

  mtk::Tools::BeginTestNo(5);

  int rr = 4;
  int cc = 7;

  mtk::DenseMatrix m5(rr,cc);

  for (auto ii = 0; ii < rr; ++ii) {
    for (auto jj = 0; jj < cc; ++jj) {
      m5.SetValue(ii,jj,(mtk::Real) ii + jj);
    }
  }

  std::cout << m5 << std::endl;

  mtk::Real *vals = m5.data();

  for (auto ii = 0; ii < rr; ++ii) {
    for (auto jj = 0; jj < cc; ++jj) {
      std::cout << " " << vals[ii*cc + jj];
    }
    std::cout << std::endl;
  }
  std::cout << std::endl;

  for (auto ii = 0; ii < rr; ++ii) {
    for (auto jj = 0; jj < cc; ++jj) {
      std::cout << " " << m5.GetValue(ii,jj);
    }
    std::cout << std::endl;
  }
  std::cout << std::endl;

  mtk::Tools::EndTestNo(5);
}

void Test6() {

  mtk::Tools::BeginTestNo(6);

  bool transpose = false;
  int generator_length = 3;
  int progression_length = 4;

  mtk::Real generator[] = {-0.5, 0.5, 1.5};

  mtk::DenseMatrix m6(generator,generator_length,progression_length,transpose);

  std::cout << m6 << std::endl;

  transpose = true;

```

```
  mtk::DenseMatrix m7(generator,generator_length,progression_length,transpose);

  std::cout << m7 << std::endl;


  mtk::Tools::EndTestNo(6);
}

void Test7() {

  mtk::Tools::BeginTestNo(7);

  bool padded = false;
  bool transpose = false;
  int lots_of_rows = 2;
  int lots_of_cols = 5;
  mtk::DenseMatrix m8(lots_of_rows,padded,transpose);

  std::cout << m8 << std::endl;

  mtk::DenseMatrix m9(lots_of_rows,lots_of_cols);

  for (auto ii = 0; ii < lots_of_rows; ++ii) {
    for (auto jj = 0; jj < lots_of_cols; ++jj) {
      m9.SetValue(ii,jj,(mtk::Real) ii*lots_of_cols + jj + 1);
    }
  }

  std::cout << m9 << std::endl;

  mtk::DenseMatrix m10 = mtk::DenseMatrix::Kron(m8,m9);

  std::cout << m10 << std::endl;

  mtk::Tools::EndTestNo(7);
}

void Test8() {

  mtk::Tools::BeginTestNo(8);

  int lots_of_rows = 4;
  int lots_of_cols = 3;
  mtk::DenseMatrix m11(lots_of_rows,lots_of_cols);

  for (auto ii = 0; ii < lots_of_rows; ++ii) {
    for (auto jj = 0; jj < lots_of_cols; ++jj) {
      m11.SetValue(ii,jj,(mtk::Real) ii*lots_of_cols + jj + 1);
    }
  }

  std::cout << m11 << std::endl;

  m11.Transpose();

  std::cout << m11 << std::endl;

  mtk::DenseMatrix m12;

  m12 = m11;

  std::cout << m12 << std::endl;

  mtk::Tools::EndTestNo(8);
}

void Test9() {

  mtk::Tools::BeginTestNo(9);

  bool transpose = false;
  int gg_l = 3;
  int progression_length = 4;
  mtk::Real gg[] = {-0.5, 0.5, 1.5};

  mtk::DenseMatrix m13(gg, gg_l ,progression_length, transpose);

  std::cout << m13 << std::endl;

  mtk::DenseMatrix m14;

```

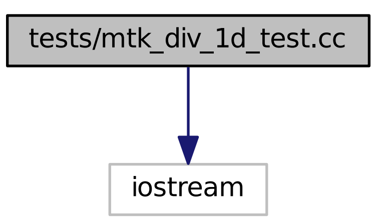
```
  m14 = m13;

  std::cout << m14 << std::endl;

  m13.Transpose();

  std::cout << m13 << std::endl;

  m14 = m13;

  std::cout << m14 << std::endl;

  mtk::Tools::EndTestNo(9);
}

int main () {

  std::cout << "Testing mtk::DenseMatrix class." << std::endl;

  Test1();
  Test2();
  Test3();
  Test4();
  Test5();
  Test6();
  Test7();
  Test8();
  Test9();
}

#else
#include <iostream>
using std::cout;
using std::endl;
int main () {
  cout << "This code HAS to be compiled with support for C++11." << endl;
  cout << "Exiting..." << endl;
}
#endif
```

17.63 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 ()

17.63.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_div_1d_test.cc.

17.63.2 Function Documentation

17.63.2.1 int main ( )

Definition at line 248 of file mtk_div_1d_test.cc.

17.64 mtk_div_1d_test.cc

```
/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#if __cplusplus == 201103L

#include <iostream>

#include "mtk.h"

void Test1() {
```

```

  mtk::Tools::BeginTestNo(1);

  mtk::Div1D div2;

  bool info = div2.ConstructDiv1D();

  if (!info) {
    std::cerr << "Mimetic div (2nd order) could not be built." << std::endl;
  }

  std::cout << div2 << std::endl;

  mtk::Tools::EndTestNo(1);
}

void Test2() {

  mtk::Tools::BeginTestNo(2);

  mtk::Div1D div4;

  bool info = div4.ConstructDiv1D(4);

  if (!info) {
    std::cerr << "Mimetic div (4th order) could not be built." << std::endl;
  }

  std::cout << div4 << std::endl;

  mtk::Tools::EndTestNo(2);
}

void Test3() {

  mtk::Tools::BeginTestNo(3);

  mtk::Div1D div6;

  bool info = div6.ConstructDiv1D(6);

  if (!info) {
    std::cerr << "Mimetic div (6th order) could not be built." << std::endl;
  }

  std::cout << div6 << std::endl;

  mtk::Tools::EndTestNo(3);
}

void Test4() {

  mtk::Tools::BeginTestNo(4);

  mtk::Div1D div8;

  bool info = div8.ConstructDiv1D(8);

  if (!info) {
    std::cerr << "Mimetic div (8th order) could not be built." << std::endl;
  }

  std::cout << div8 << std::endl;

  mtk::Tools::EndTestNo(4);
}

void Test5() {

  mtk::Tools::BeginTestNo(5);

  mtk::Div1D div10;

  bool info = div10.ConstructDiv1D(10);

  if (!info) {
    std::cerr << "Mimetic div (10th order) could not be built." << std::endl;
  }

  std::cout << div10 << std::endl;

```

```
  mtk::Tools::EndTestNo(5);
}

void Test6() {

  mtk::Tools::BeginTestNo(6);

  mtk::Div1D div12;

  bool info = div12.ConstructDiv1D(12);

  if (!info) {
    std::cerr << "Mimetic div (12th order) could not be built." << std::endl;
  }

  std::cout << div12 << std::endl;

  mtk::Tools::EndTestNo(6);
}

void Test7() {

  mtk::Tools::BeginTestNo(7);

  mtk::Div1D div14;

  bool info = div14.ConstructDiv1D(14);

  if (!info) {
    std::cerr << "Mimetic div (14th order) could not be built." << std::endl;
  }

  std::cout << div14 << std::endl;

  mtk::Tools::EndTestNo(7);
}

void Test8() {

  mtk::Tools::BeginTestNo(8);

  mtk::Div1D div2;

  bool info = div2.ConstructDiv1D();

  if (!info) {
    std::cerr << "Mimetic div (2nd order) could not be built." << std::endl;
  }

  std::cout << div2 << std::endl;

  mtk::UniStgGrid1D grid(0.0, 1.0, 5);

  std::cout << grid << std::endl;

  mtk::DenseMatrix div2m(div2.ReturnAsDenseMatrix(grid));

  std::cout << div2m << std::endl;

  mtk::Tools::EndTestNo(8);
}

void Test9() {

  mtk::Tools::BeginTestNo(9);

  mtk::Div1D div4;

  bool info = div4.ConstructDiv1D(4);

  if (!info) {
    std::cerr << "Mimetic div (4th order) could not be built." << std::endl;
  }

  std::cout << div4 << std::endl;

  mtk::UniStgGrid1D grid(0.0, 1.0, 11);

  std::cout << grid << std::endl;

  mtk::DenseMatrix div4m(div4.ReturnAsDenseMatrix(grid));
```

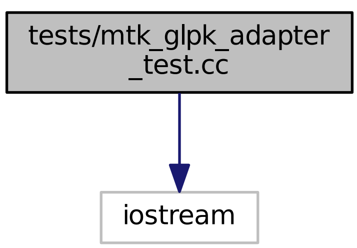
```

  std::cout << div4m << std::endl;

  mtk::Tools::EndTestNo(9);
}

int main () {

  std::cout << "Testing mtk::Div1D class." << std::endl;

  Test1();
  Test2();
  Test3();
  Test4();
  Test5();
  Test6();
  Test7();
  Test8();
  Test9();
}

#else
#include <iostream>
using std::cout;
using std::endl;
int main () {
  cout << "This code HAS to be compiled with support for C++11." << endl;
  cout << "Exiting..." << endl;
}
#endif
```

17.65 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 ()

17.65.1 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.

17.65.2 Function Documentation

17.65.2.1 int main ( )

Definition at line 81 of file mtk_glpk_adapter_test.cc.

17.66 mtk_glpk_adapter_test.cc

```

/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#if __cplusplus == 201103L

#include <iostream>
#include <ctime>

#include "mtk.h"

```

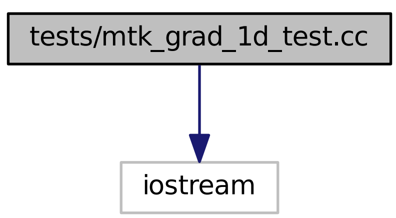
```
void Test1() {

  mtk::Tools::BeginTestNo(1);

  mtk::Tools::EndTestNo(1);
}

int main () {

  std::cout << "Testing mtk::GLPKAdapter class." << std::endl;

  Test1();
}

#else
#include <iostream>
using std::cout;
using std::endl;
int main () {
  cout << "This code HAS to be compiled with support for C++11." << endl;
  cout << "Exiting..." << endl;
}
#endif
```

17.67 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 ()

17.67.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_grad_1d_test.cc.

17.67.2 Function Documentation

17.67.2.1 int main ( )

Definition at line 186 of file mtk_grad_1d_test.cc.

17.68 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. Documentation related to said modifications should be included.
00018
00019 2. Redistributions of source code must be done through direct
00020 downloads from the project's GitHub page: http://www.csrc.sdsu.edu/mtk
00021
00022 3. Redistributions of source code must retain the above copyright notice, this
00023 list of conditions and the following disclaimer.
00024
00025 4. Redistributions in binary form must reproduce the above copyright notice,
00026 this list of conditions and the following disclaimer in the documentation and/or
00027 other materials provided with the distribution.
00028
00029 5. Usage of the binary form on proprietary applications shall require explicit
00030 prior written permission from the the copyright holders.
00031
00032 6. 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 Test1() {
00061
00062   mtk::Tools::BeginTestNo(1);
00063
00064   mtk::Grad1D grad2;
00065
00066   bool info = grad2.ConstructGrad1D();
00067
00068   if (!info) {
00069     std::cerr << "Mimetic grad (2nd order) could not be built." << std::endl;
00070   }
00071
00072   std::cout << grad2 << std::endl;
00073
```

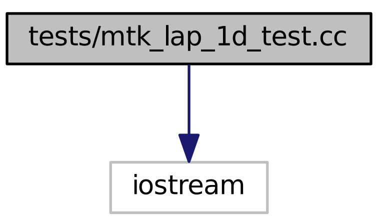
```
  mtk::Tools::EndTestNo(1);
}

void Test2() {

  mtk::Tools::BeginTestNo(2);

  mtk::Grad1D grad4;

  bool info = grad4.ConstructGrad1D(4);

  if (!info) {
    std::cerr << "Mimetic grad (4th order) could not be built." << std::endl;
  }

  std::cout << grad4 << std::endl;

  mtk::Tools::EndTestNo(2);
}

void Test3() {

  mtk::Tools::BeginTestNo(3);

  mtk::Grad1D grad6;

  bool info = grad6.ConstructGrad1D(6);

  if (!info) {
    std::cerr << "Mimetic grad (6th order) could not be built." << std::endl;
  }

  std::cout << grad6 << std::endl;

  mtk::Tools::EndTestNo(3);
}

void Test4() {

  mtk::Tools::BeginTestNo(4);

  mtk::Grad1D grad8;

  bool info = grad8.ConstructGrad1D(8);

  if (!info) {
    std::cerr << "Mimetic grad (8th order) could not be built." << std::endl;
  }

  std::cout << grad8 << std::endl;

  mtk::Tools::EndTestNo(4);
}

void Test5() {

  mtk::Tools::BeginTestNo(5);

  mtk::Grad1D grad10;

  bool info = grad10.ConstructGrad1D(10);

  if (!info) {
    std::cerr << "Mimetic grad (10th order) could not be built." << std::endl;
  }

  std::cout << grad10 << std::endl;

  mtk::Tools::EndTestNo(5);
}

void Test6() {

  mtk::Tools::BeginTestNo(6);

  mtk::Grad1D grad2;

  bool info = grad2.ConstructGrad1D();

  if (!info) {
    std::cerr << "Mimetic grad (2nd order) could not be built." << std::endl;
```

```
  }

  std::cout << grad2 << std::endl;

  mtk::UniStgGrid1D grid(0.0, 1.0, 5);

  std::cout << grid << std::endl;

  mtk::DenseMatrix grad2m(grad2.ReturnAsDenseMatrix(grid));

  std::cout << grad2m << std::endl;

  mtk::Tools::EndTestNo(6);
}

int main () {

  std::cout << "Testing mtk::Grad1D class." << std::endl;

  Test1();
  Test2();
  Test3();
  Test4();
  Test5();
  Test6();
}

#else
#include <iostream>
using std::cout;
using std::endl;
int main () {
  cout << "This code HAS to be compiled with support for C++11." << endl;
  cout << "Exiting..." << endl;
}
#endif
```

17.69 tests/mtk_lap_1d_test.cc File Reference

`#include <iostream>`

Include dependency graph for mtk_lap_1d_test.cc:



Functions

- int main ()

17.69.1 Function Documentation

17.69.1.1 int main ( )

Definition at line 156 of file mtk_lap_1d_test.cc.

17.70 mtk_lap_1d_test.cc

```
#if __cplusplus == 201103L

#include <iostream>

#include "mtk.h"

void Test1() {

  mtk::Tools::BeginTestNo(1);

  mtk::Lap1D lap2;

  bool info = lap2.ConstructLap1D();

  if (!info) {
    std::cerr << "Mimetic lap (2nd order) could not be built." << std::endl;
  }

  mtk::Tools::EndTestNo(1);
}

void Test2() {

  mtk::Tools::BeginTestNo(2);

  mtk::Lap1D lap4;

  bool info = lap4.ConstructLap1D(4);

  if (!info) {
    std::cerr << "Mimetic lap (4th order) could not be built." << std::endl;
  }

  mtk::Tools::EndTestNo(2);
}

void Test3() {

  mtk::Tools::BeginTestNo(3);

  mtk::Lap1D lap6;

  bool info = lap6.ConstructLap1D(6);

  if (!info) {
    std::cerr << "Mimetic lap (6th order) could not be built." << std::endl;
  }

  mtk::Tools::EndTestNo(3);
}

void Test4() {

  mtk::Tools::BeginTestNo(4);

  mtk::Lap1D lap8;

  bool info = lap8.ConstructLap1D(8);

  if (!info) {
    std::cerr << "Mimetic lap (8th order) could not be built." << std::endl;
  }

  mtk::Tools::EndTestNo(4);
}

void Test5() {

  mtk::Tools::BeginTestNo(5);

```

```
  mtk::Lap1D lap10;

  bool info = lap10.ConstructLap1D(10);

  if (!info) {
    std::cerr << "Mimetic lap (10th order) could not be built." << std::endl;
  }

  mtk::Tools::EndTestNo(5);
}

void Test6() {

  mtk::Tools::BeginTestNo(6);

  mtk::Lap1D lap12;

  bool info = lap12.ConstructLap1D(12);

  if (!info) {
    std::cerr << "Mimetic lap (12th order) could not be built." << std::endl;
  }

  mtk::Tools::EndTestNo(6);
}

void Test7() {

  mtk::Tools::BeginTestNo(7);

  mtk::Lap1D lap4;

  bool info = lap4.ConstructLap1D(4);

  if (!info) {
    std::cerr << "Mimetic lap (4th order) could not be built." << std::endl;
  }

  std::cout << lap4 << std::endl;
  std::cout << std::endl;

  mtk::Tools::EndTestNo(7);
}

void Test8() {

  mtk::Tools::BeginTestNo(8);

  mtk::Lap1D lap4;

  bool info = lap4.ConstructLap1D(4);

  if (!info) {
    std::cerr << "Mimetic lap (4th order) could not be built." << std::endl;
  }

  std::cout << lap4 << std::endl;
  std::cout << std::endl;

  mtk::UniStgGrid1D aux(0.0, 1.0, 11);

  mtk::DenseMatrix lap4_m(lap4.ReturnAsDenseMatrix(aux));

  std::cout << lap4_m << std::endl;
  std::cout << std::endl;

  mtk::Tools::EndTestNo(8);
}

int main () {

  std::cout << "Testing MTK 1D Laplacian" << std::endl;

  Test1();
  Test2();
  Test3();
  Test4();
  Test5();
  Test6();
  Test7();
  Test8();
```

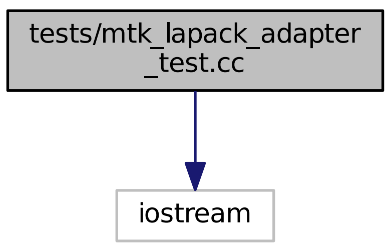
```
}

#else
#include <iostream>
int main () {
  std::cout << "This code HAS to be compiled to support C++11." << std::endl;
  std::cout << "Exiting..." << std::endl;
}
#endif
```

17.71 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 ()

17.71.1 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.

17.71.2 Function Documentation

17.71.2.1 int main ( )

Definition at line 81 of file mtk_lapack_adapter_test.cc.

17.72 mtk_lapack_adapter_test.cc

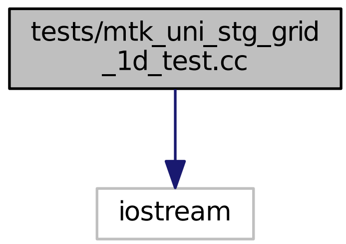
```
/*
Copyright (C) 2015, Computational Science Research Center, San Diego State
University. All rights reserved.

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. Documentation related to said modifications should be included.

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#if __cplusplus == 201103L

#include <iostream>
#include <ctime>

#include "mtk.h"

void Test1() {

  mtk::Tools::BeginTestNo(1);

  mtk::Tools::EndTestNo(1);
}

int main () {

  std::cout << "Testing mtk::LAPACKAdapter class." << std::endl;

  Test1();
}

#else
#include <iostream>
using std::cout;
using std::endl;
int main () {
  cout << "This code HAS to be compiled with support for C++11." << endl;
  cout << "Exiting..." << endl;
}
#endif
```

17.73 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 ()

17.73.1 Detailed Description

Author

: Eduardo J. Sanchez (ejspeiro) - esanchez at mail dot sdsu dot edu

Definition in file mtk_uni_stg_grid_1d_test.cc.

17.73.2 Function Documentation

17.73.2.1 int main ( )

Definition at line 164 of file mtk_uni_stg_grid_1d_test.cc.

17.74 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. Documentation related to said modifications should be included.
```

```

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 of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

4. 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.

5. Usage of the binary form on proprietary applications shall require explicit
prior written permission from the the copyright holders.

6. 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.
*/

#if __cplusplus == 201103L

#include <iostream>
#include <ctime>

#include "mtk.h"

void Test1() {

  mtk::Tools::BeginTestNo(1);

  mtk::UniStgGrid1D gg;

  std::cout << gg << std::endl;

  mtk::Tools::EndTestNo(1);
}

mtk::Real ScalarFieldOne(mtk::Real xx) {

  return 2.0*xx;
}

void Test2() {

  mtk::Tools::BeginTestNo(2);

  mtk::Real aa = 0.0;
  mtk::Real bb = 1.0;

  int nn = 5;

  mtk::UniStgGrid1D gg(aa, bb, nn);

  std::cout << gg << std::endl;

  gg.BindScalarField(ScalarFieldOne);

  std::cout << gg << std::endl;

  mtk::Tools::EndTestNo(2);
}

void Test3() {

```

```
  mtk::Tools::BeginTestNo(3);

  mtk::Real aa = 0.0;
  mtk::Real bb = 1.0;

  int nn = 5;

  mtk::UniStgGrid1D gg(aa, bb, nn);

  std::cout << gg << std::endl;

  gg.BindScalarField(ScalarFieldOne);

  std::cout << gg << std::endl;

  if(!gg.WriteToFile("mtk_uni_stg_grid_1d_test_03.dat", "x", "u(x)")) {
    std::cerr << "Error writing to file." << std::endl;
  }

  mtk::Tools::EndTestNo(3);
}

mtk::Real VectorFieldXComponentOne(mtk::Real xx) {

  return xx*xx;
}

void Test4() {

  mtk::Tools::BeginTestNo(4);

  mtk::Real aa = 0.0;
  mtk::Real bb = 1.0;

  int nn = 20;

  mtk::UniStgGrid1D gg(aa, bb, nn, mtk::VECTOR);

  std::cout << gg << std::endl;

  gg.BindVectorField(VectorFieldXComponentOne);

  std::cout << gg << std::endl;

  if(!gg.WriteToFile("mtk_uni_stg_grid_1d_test_04.dat", "x", "v(x)")) {
    std::cerr << "Error writing to file." << std::endl;
  }

  mtk::Tools::EndTestNo(4);
}

int main () {

  std::cout << "Testing mtk::UniStgGrid1D class." << std::endl;

  Test1();
  Test2();
  Test3();
  Test4();
}

#else
#include <iostream>
using std::cout;
using std::endl;
int main () {
  cout << "This code HAS to be compiled with support for C++11." << endl;
  cout << "Exiting..." << endl;
}
#endif
```

Index