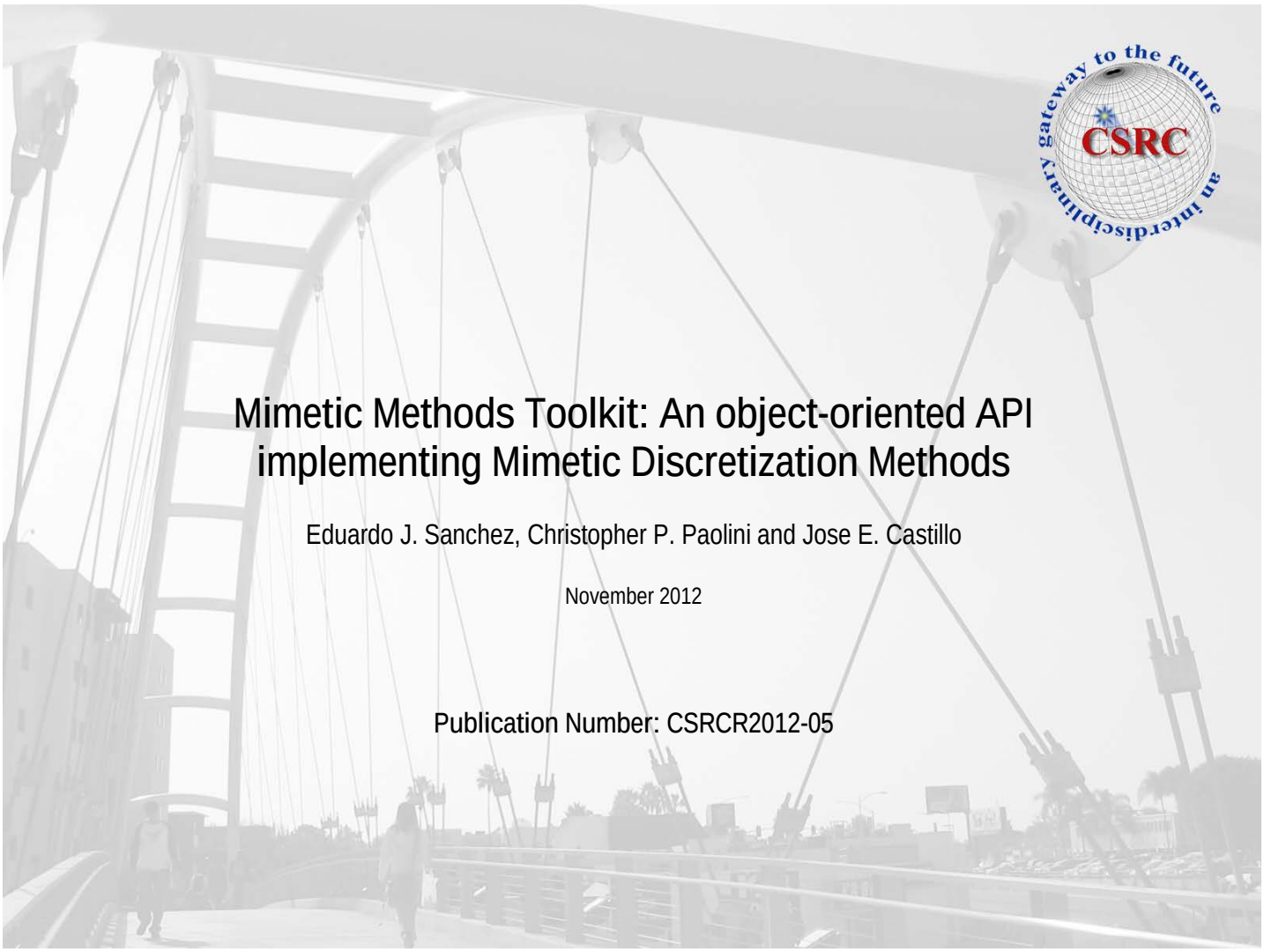


# Mimetic Methods Toolkit: An object-oriented API implementing Mimetic Discretization Methods

Eduardo J. Sanchez, Christopher P. Paolini and Jose E. Castillo

November 2012

Publication Number: CSRCR2012-05

Computational Science & Engineering Faculty and Students Research Articles

Database Powered by the Computational Science Research Center Computing Group & Visualization Lab

COMPUTATIONAL SCIENCE & ENGINEERING

SAN DIEGO STATE UNIVERSITY

Computational Science Research Center
College of Sciences
5500 Campanile Drive
San Diego, CA 92182-1245
(619) 594-3430

# Research Advances for Fall 2012

# Mimetic Methods Toolkit: An object-oriented API implementing Mimetic Discretization Methods

Eduardo J. Sanchez*, Christopher P. Paolini† and Jose E. Castillo‡

November 26, 2012

**Abstract**

In this work, we introduce the Mimetic Methods Toolkit (MTK), an object-oriented Application Programming Interface for the implementation of Mimetic Discretization Methods in developing computer applications of a scientific nature, where the numerical solution of Partial Differential Equations may be required. The MTK was designed based on the Castillo–Grone Method for the construction of discrete differential operators that mimic important properties of their continuous counterparts. The MTK is built as a collection of abstract and concrete classes, thus allowing for an extensible framework, which fosters code reutilization, while intuitively educating the user about the important theoretical aspects of the Mimetic Discretization Methods. In this work, we present an introduction to Mimetic Discretization Methods, and we discuss the computational modeling of the related concepts; in this way, we explain how does the MTK implement these methods. By means of examples, we illustrate the MTK's usage philosophy and, finally, by means of comparing the attained result against previously studied reference solutions, we conclude the correctitude of the implementation efforts in the MTK.

*Computational Science Research Center, 5500 Campanile Dr, San Diego State University, College of Sciences, San Diego, California, USA, 92182-1245.

†Computational Science Research Center, 5500 Campanile Dr, San Diego State University, College of Sciences, San Diego, California, USA, 92182-1245.

‡Computational Science Research Center, 5500 Campanile Dr, San Diego State University, College of Sciences, San Diego, California, USA, 92182-1245.

This page has been intentionally left blank for printing purposes.

# Contents



# List of Figures

# List of Tables

# List of Algorithms

This page has been intentionally left blank for printing purposes.

## Notational Conventions

In this work, we shall take notational conventions very seriously. Notation is our only interface with the already complex world of the abstract theories we will be dealing with, so why not give it its importance?

1. We shall denote continuous scalar-valued quantities, say temperature or pressure, with the default math pseudo-italicized font, using both lower and uppercase Latin letters and lowercase Greek letters: $a, ..., z, A, ...Z, \alpha, ..., \omega$. Discretized instances shall be identified with a tilde accent, and will be assumed to be implemented as row-wise-defined arrays.

2. We shall denote continuous vector-valued quantities using boldfaced lowercase Latin letters: $\mathbf{a}, ..., \mathbf{z}$. Discretized instances shall be identified with a tilde accent.

3. We shall denote matrices using boldfaced uppercase Latin and Greek letters: $\mathbf{A}, ..., \mathbf{Z}, \mathbf{\Gamma}, ..., \mathbf{\Omega}$.

4. We shall denote continuous tensor-valued quantities using scripture-styled uppercase Latin letters: $\mathscr{A}, ..., \mathscr{Z}$. Discretized instances shall be identified with a tilde accent.

5. We shall denote continuous differential operators using standard notation from Calculus. When it comes to their discrete matrix analog operators, we will use boldfaced uppercase Latin Letters with a tilde accent, thus emphasizing the approximation to a continuous operator they intent to. However, those operators built by means of the Castillo–Grone Method shall be identified with a breve accent. As a side note, this notational convention is supported by the fact that, on German cartography, a breve accent placed over two letters is often used in abbreviated place names that end in "$\breve{bg}$", as a short for "*burg*", a common suffix originally meaning "Castle", which is English for "Castillo". This prevents misinterpretation since "*berg*" is another common suffix in place names, which means "mountain". Thus, for example, "*Freib̆g*" stands for "*Freiburg*", not "*Freiberg*". Furthermore, its is also mnemonic, since it resembles a letter 'C'.

6. We shall denote sets using italic uppercase Greek letters: $A, ..., \Omega$. Discretized instances shall be identified with a tilde accent. Numerical sets will be denoted with blackboard boldfaced Latin uppercase letters: $\mathbb{A}, ..., \mathbb{Z}$.

Table 1 summarizes our notational conventions. An important thing to notice is that, when accessed by means of indexing the elements they may contain, the objects shall not conserve their typographical style, thus yielding a default math pseudo-italicized font. For example, notice that tensors loose their typographical style, in this case, their scripture style, thus yielding a default math pseudo-italicized font uppercase Latin letter. This can be depicted in columns three and four of Table 1. However, when objects are indexed as being part of a enumerable set, they will preserve their typographical style.

| Object | Continuous domain | Discrete domain | Indexed |
|---|---|---|---|
| Scalar | $a, ..., z, A, ...Z, \alpha, ..., \omega$ | $\tilde{a}, ..., \tilde{z}, \tilde{A}, ..., \tilde{Z}, \tilde{\alpha}, ..., \tilde{\omega}$ | $a_i, ..., z_i, A_i, ...Z_i, \alpha_i, ..., \omega_i$ |
| Vector | $\mathbf{a}, ..., \mathbf{z}$ | $\tilde{\mathbf{a}}, ..., \tilde{\mathbf{z}}$ | $a_i, ..., z_i$ |
| Matrix | $\mathbf{A}, ..., \mathbf{Z}, \mathbf{\Gamma}, ..., \mathbf{\Omega}$ | $\mathbf{A}, ..., \mathbf{Z}, \mathbf{\Gamma}, ..., \mathbf{\Omega}$ | $a_{ij}, ..., z_{ij}, \alpha_{ij}, ..., \omega_{ij}$ |
| Tensor | $\mathscr{A}, ..., \mathscr{Z}$ | $\tilde{\mathscr{A}}, ..., \tilde{\mathscr{Z}}$ | $A_{ij}, ..., Z_{ij}$ |
| Operator | $\nabla$, $\nabla\cdot$,... | $\tilde{\mathbf{A}}, ..., \tilde{\mathbf{Z}}$ or $\breve{\mathbf{A}}, ..., \breve{\mathbf{Z}}$ | $\tilde{A}_{ij}, ..., \tilde{Z}_{ij}$ or $\breve{A}_{ij}, ..., \breve{Z}_{ij}$ |
| Set | $A, ..., \Omega$ or $\mathbb{A}, ..., \mathbb{Z}$ | $\tilde{A}, ..., \tilde{\Omega}$ | $\tilde{A}_i, ..., \tilde{\Omega}_i$ |

Table 1: Summary of the notational conventions adopted in this work.

This page has been intentionally left blank for printing purposes.

# 1 Introduction

Computational Science has emerged as a cutting edge field when it comes to advancing the knowledge of diverse sciences. Its interdisciplinary nature has proven to be an efficient bridge towards the understanding of the diverse physical phenomena that comprise nature. These physical phenomena are typically modeled as a set of partial (or ordinary) differential (or integral) equations, which usually correspond to a particular conservation law. Therefore, numerical methods used to discretize, solve, and study these equations are of vital importance in the current paradigm of interdisciplinary science. Particularly, in this work, we will be concerned with the use of **Mimetic Discretization Methods (MDMs)** [1].

The field of MDMs has undergone research for a long time. An important work is [2], in where the construction of high-order discrete differential operators is studied. Particularly, in [2], attention is given to the importance of having these operators satisfying important properties that come from their analogous continuous differential operators. Furthermore, in [2], the authors define difference approximations that retain the properties of the continuum operators to be called **mimetic**. Also, in [2], the authors state that Partial Differential Equations solved with these mimetic difference approximations (or Mimetic Discretization Methods) often automatically satisfy discrete versions of conservation laws and analogies to Stokes'[1] Theorem, that are true in the continuum, and therefore are more likely to produce physically valid numerical results.

In 2003, the work of Castillo and Grone [3] studied and solved an important drawback of the MDMs at the time. In [3], the authors construct one-dimensional mimetic differential operators; however, the authors state that creating second-order approximations away from the boundary is simple, but obtaining appropriate behavior near the boundary is difficult, even in a one-dimensional uniform grid. Considering this, in [3], the authors introduce the **Castillo–Grone Method (CGM)** that allows for the construction of mimetic differential operators that yield approximations with the same order of accuracy at the boundary, as well as in the interior of the discretized domain.

Subsequently, diverse works focusing on several theoretical and computational aspects of the CGM-based MDMs have been published. Specifically, in [4], the authors implement the CGM in constructing a second-order discretization, and compare the results with other second-order discretization methods, by applying the CGM to an elliptic boundary value problem in one dimension.

The extension to nonuniform staggered grids of the CGM is presented in a work [5] from the *Facultad Experimental de Ciencias y Tecnología (FACYT)* at the *Universidad de Carabobo*, in Carabobo, Venezuela. In [5], the authors extent the CGM and present this extension on second-order mimetic operators, although the method works for any order of accuracy. Later, the work was extended in [6], where the authors propose a technique for implementing second- and fourth-order mimetic operators over nonuniform, structured one-dimensional meshes. Analogously, researchers from the *Centro Multidisciplinario de Visualización y Computo Científico (CEMVICC)*, a research laboratory within the *FACYT*, at the *Universidad de Carabobo*, in Carabobo, Venezuela, studied [7] the computational implications of solving the large sparse linear systems arising from CGM-based MDMs. Specifically, in [7], the authors perform an experimental study of iterative methods for solving large sparse linear systems arising from second-order mimetic discretizations.

Given the diverse nature of the research work that has been conducted in the field of MDMs, Castillo and Miranda authored a book [1] that summarizes most of the research efforts, while contributing with new ideas in the field of CGM-based MDMs.

In this work, we introduce the Mimetic Methods Toolkit, as an effort to provide a robust computational framework to assist in the intuitive and successful application of the MDMs. The Mimetic Methods Toolkit is written as an Application Programming Interface (API). Because of this, a process of computational modeling had to be performed. Such a process assisted in achieving uniformity and consistency within the extensive and diverse pool of research work that the MDMs have been undergoing.

[1] Sir George Gabriel Stokes, 1st Baronet - Born in Skreen, County Sligo, Ireland on August 13, 1819 - Perished in Cambridge, England on February 1, 1903 (aged 83).

## 1.1 Application Programming Interfaces (APIs)

Several works aim towards the effort of creating numerical Application Programming Interfaces, which allow users to implement the solution to a particular problem in an intuitive way. API development is an ubiquitous discipline in modern software development and in fact, it is quite likely that for every programmer, a reference to an API has been his or her first line of written code. An **Application Programming Interface (API)** provides an abstraction for a problem and specifies how its clients should interact with software components that implement a solution to that problem. Therefore, it can be said that the purpose of an API is to provide a logical interface to the functionality of a software component, while also hiding any of the component's implementation details [8].

Well-known examples of API development projects are given in [9, 10, 11, 12], and [13]; the latter being an important step toward the understanding of portable computational performance and empirical tuning. Furthermore, as computational frameworks are developed to explore new boundaries in High-Performance Computing, the accompanying efforts in API development go along, as exemplified in both [8] and [14].

In the field of numerically solving Ordinary and/or Partial Differential Equations (ODEs and/or PDEs), important theoretical work has been done, ranging from the study of data structures [15], [16], and the development of algorithms [17], to the construction of APIs assisting in the implementation of numerical schemes to write computer applications of a scientific nature [18].

Generally speaking, if we consider a particular programming paradigm, implemented through a specific programming language, as for example, C++, an API will generally include the following elements [8]:

- **Headers**: A collection of `.h` header files that define the interface and allow client code to be compiled against that interface. Open source APIs also include the source code (`.cpp` or `.cc` files) for the API implementation.
- **Libraries**: One or more static (or dynamic) library files that provide an implementation for the API. Clients can link their code against these library files in order to add any required functionality the API will provide to their applications. The reader should be aware of the important (and sometimes neglected) difference between the terms "API" and "library".
- **Documentation**: Overview information that describes how to use the API, often including automatically generated documentation for all classes and functions in the API.

A correct design of the APIs to be used in Computational Science is important, since a balance should be attained between achieving a satisfactory computational performance, and intuitively educating the user; not only in utilizing the API, but also in the theoretical aspects that underlie its design. Notice though that educating the user with respect to the underlying theory that sustains the API, is not the same as unveiling the details regarding the implementation of such theoretical framework, which is not desirable. Particularly, in the field of MDMs, such a balance is necessary, because no library has yet been developed to assist in using these methods, when constructing scientific and industrial computer applications, for which the simulation of some physical phenomena is of interest. For this purpose, we present the "Mimetic Methods Toolkit".

The **Mimetic Methods Toolkit (MTK)** is an API which allows for an intuitive implementation of CGM-based MDMs for the resolution of PDEs, yielding numerical solutions that guarantee uniform order of accuracy, all along the modeled physical domain, while ensuring the satisfaction of conservation laws, thus remaining faithful to the underlying physics of the problem. The MTK is fully developed in C++; therefore, it exploits all the well-known advantages of both object-oriented application models, and the extensive collection of data structure capabilities of this language. Examples of APIs fully developed in C++ are [18, 19, 20], and [21], whereas [22] describes how to write efficient and portable (serial and/or parallel) C++ programs to solve PDEs on a single, or in multiple curvilinear grids, that form an overlapping grid.

## 2 Organization of the Article

This work is organized as follows: We present a summarized introduction to Mimetic Discretization Methods (§3), and we discuss the computational modeling of the related concepts (§4); in this way, we explain how does the MTK implement these methods.

In order to depict how can the MTK be used to solve arising problems in developing scientific computational models or applications, as for example, [23, 24] and [25], we will present usage examples in the context of general elliptic PDEs in one dimension, in order to remain consistent with the extensive amount of work that has been done with these. When analyzing the results, we will compare the attained numerical solutions with those attained by a reference solution already known, in order to conclude the correctitude of the implementation presented in the MTK (§5). Finally, in §6, we present concluding remarks, as well as important directions of future work.

## 3 Mimetic Discretization Methods

**Mimetic Discretization Methods** are methods for the numerical solution of differential equations that begin by discretizing the underlying continuum theory of the problem of interest, instead of discretizing the proposed equation or system of equations directly [4]. With "discretizing the underlying continuum theory of a problem", we mean that they start by constructing matrix operators, which are discrete analogs to the well-known differential operators for gradient, divergence, Laplacian and curl. This is done in such a way that these operators satisfy important properties of their continuous counterparts, as for example $\nabla^2 f = \nabla \cdot (\nabla f)$, for a given scalar-valued quantity[2] $f$. This allows for the discretization scheme to replicate (or mimic) much of the behavior found in the actual continuous problem [3].

Consider an arbitrary solid[3] $\Omega$ of surface $\Sigma$, with a given normal component[4] $\mathbf{n}$, through which a given flux $\mathbf{v}(\mathbf{x}, t),\ \mathbf{x} \in \Omega,\ t \in [t_0, t_1] \subset \mathbb{R}$, goes through[5]. We can write an extended version of Gauß'[6] Divergence Theorem, for a given scalar-valued quantity $f(\mathbf{x}, t)$ of interest, as follows [1]:

$$\iiint_{\Omega} \langle \nabla f, \mathbf{v} \rangle d\Omega + \iiint_{\Omega} f(\nabla \cdot \mathbf{v}) d\Omega = \iint_{\Sigma} f \langle \mathbf{v}, \mathbf{n} \rangle d\Sigma. \tag{1}$$

The details concerning the nature of (1) are given in [1]. The creation of discrete differential operators results from considering a discrete version of (1), which for discretized quantities $\tilde{f}$ and $\tilde{\mathbf{v}}$ of interest reads [1]:

$$\Delta x \langle \breve{\mathbf{G}} \tilde{f}, \tilde{\mathbf{v}} \rangle_{\mathbf{P}} + \Delta x \langle \tilde{f}, \breve{\mathbf{D}} \tilde{\mathbf{v}} \rangle_{\mathbf{Q}} = \langle \tilde{f}, \breve{\mathbf{B}} \tilde{\mathbf{v}} \rangle, \tag{2}$$

where $\breve{\mathbf{G}}$[7], $\breve{\mathbf{D}}$ and $\breve{\mathbf{B}}$ stand for the discrete (or mimetic) CGM-based gradient, divergence and boundary operators, respectively, and $\Delta x$ is the chosen step size for the discretization.

[2] We shall denote continuous scalar-valued quantities, say temperature or pressure, with the default math pseudo-italicized font, using both lower and uppercase Latin letters and lowercase Greek letters. Discretized instances shall be identified with a tilde accent, and will be assumed to be implemented as row-wise-defined arrays.

[3] We shall denote sets using italic uppercase Greek letters.

[4] We shall denote continuous vector-valued quantities using boldfaced lowercase Latin letters. Discretized instances shall be identified with a tilde accent.

[5] Numerical sets will be denoted with blackboard boldfaced Latin uppercase letter.

[6] Johann Carl Friedrich Gauß - Born in Braunschweig, Principality of Brunswick-Wolfenbüttel, Holy Roman Empire on April 30, 1777 - Perished in Göttingen, Kingdom of Hanover on February 23, 1855 (aged 77).

[7] We shall denote discrete matrix operators using boldfaced uppercase Latin Letters with a tilde accent, thus emphasizing the approximation to a continuous operator, which they intent. However, those operators built by means of the Castillo–Grone Method shall be identified with a breve accent. As a side note, this notational convention is supported by the fact that, on German cartography, a breve accent placed over two letters is often used in abbreviated place names that end in "*b̆g*", as a short for "*burg*", a common suffix originally meaning "Castle", which is English for "Castillo". This prevents misinterpretation since "*berg*" is another common suffix in place names, which means "mountain". Thus, for example, "*Freib̆g*" stands for "*Freiburg*", not "*Freiberg*". Furthermore, its is also mnemonic, since it resembles a letter 'C'.

The reader should compare both (1) and (2). For example, the chevrons in (1) denote the standard continuous inner product, whereas the chevrons in (2) denote a generalized discrete inner product with weight matrices[8] $\mathbf{Q}$ and $\mathbf{P}$, as described in both [1] and [3].

In this work, we will be concerned with the **Castillo–Grone Method (CGM)** [1], [3], which is the method used to construct the mimetic operators that are implemented in the MTK. One of the advantages of the CGM over other MDMs is that it allows to achieve uniform order of accuracy, not only through the interior of the attained discretized domain, but also at its boundaries, without the necessity of ghost points or any other numerical artifact, as it could be the case with standard Finite Difference Methods (FDMs) [1], [3], [4]. Comparisons of the CGM against other well-known methods as for example the FDM, as well as with another MDMs, is presented in [4].

The readers interested in the theoretical aspects of both the Mimetic Discretization Methods and the CGM, can consult [1]; nonetheless, in order for this to be a self-contained work, in the following sections, we will present a brief introduction to important concepts in MDMs; specifically, we will present an overview of the details concerning the discretization of the physical domain by means of a staggered grid (§3.1) and the details concerning the mimetic differential operators (§3.2).

## 3.1 One-dimensional staggered grids

Approximations (for the one-dimensional case) are performed on a uniform, one-dimensional **mesh**, which yields a **staggered grid**, thus binding the numerical results from the mimetic operators to the discretized domain, as depicted in Figures 1, 2 and 3.

Specifically, Figure 1 shows that, for a given step size, $\Delta x$, separating the nodes within the nodal grid (soon to be staggered), approximations for the discretized vector-valued quantities of interest (as for example, the discrete gradient $\breve{\mathbf{G}}\tilde{f}$ of a discrete scalar-valued quantity $\tilde{f}$) shall be performed on these **nodes**, which are nothing else but projected faces of three-dimensional cells [1]. Given their nature, these nodes are denoted (see Figure 1) with vertical lines. Figure 1 also shows that the value of the scalar-valued quantities under consideration should also be available at the **boundary nodes**, which, in this figure, are denoted as circles.

For every $i$-th node, the **cell** defined by the two nodes $x_i$ and $x_{i+1}$, will be said to have a **center** located at the coordinate $x_{i+1/2}$, as depicted in Figure 2. Therefore, the first center will be located at a distance of $\Delta x/2$ from the west boundary node. Cell centers, on the other hand, are also denoted as circles (because they correspond to the projection of the centers of the three-dimensional cells). Figure 2 indicates that the values for all the scalar-valued quantities of interest (as for example, the discrete divergence $\breve{\mathbf{D}}\tilde{\mathbf{v}}$ of a discrete vector-valued quantity $\tilde{\mathbf{v}}$) will be computed at these centers.

Figure 3 suggests that the boundary nodes are different from the centers, which is true. The reason is that, the centers will hold the divergence of any discretized vector-valued quantity of interest; in particular, the discrete gradient of any discretized scalar-valued quantity of interest. That is, the centers will hold the computation of the discrete Laplacian $\breve{\mathbf{L}}\tilde{f}$ of any discretized scalar-valued quantity $\tilde{f}$. Therefore, Figure 3 suggests that we should now denote the centers with triangles instead. We believe that these notational conventions are quite mnemonic, since, for example, a triangle resembles the Nabla operator. Furthermore, given its geometry, the triangle allows for a circumscribed circumference to lie inside of it, which will represent the center upon which the computation of the discrete Laplacian is being performed. Clearly, this eliminates the apparent ambiguity among the notation for centers and boundary nodes, temporarily introduced in Figure 2.

## 3.2 Mimetic Differential Operators

In this section, we will briefly summarize the most important aspects of the mimetic operators. As it was mentioned in §3, the interested reader in the intrinsic details of the mimetic differential operators, computed by the CGM, can refer to [1]; however, it is worth mentioning that (as in the case of standard FDMs) knowledge on the derivation of the operators is not required for the application of the CGM-based MDMs.

[8] We shall denote matrices using boldfaced uppercase Latin letters.

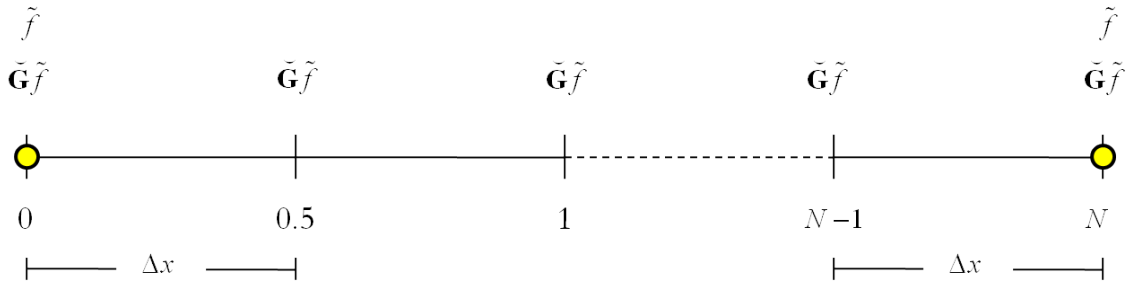



Figure 1: A one-dimensional uniform nodal grid with $N$ cells and step-size $\Delta x = 0.5$. This Figure depicts how are the approximations for the discrete gradient bound to the staggered grid, as well as the importance of the boundary nodes, as explained in §3.1.

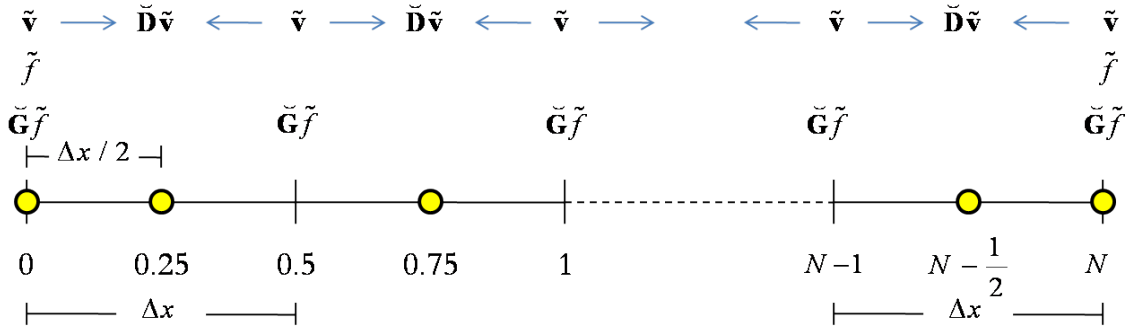



Figure 2: A one-dimensional uniform staggered grid with $N$ cells and step-size $\Delta x = 0.5$. This Figure depicts how are the approximations for the discrete gradient and divergence bound to the staggered grid, as explained in §3.1.

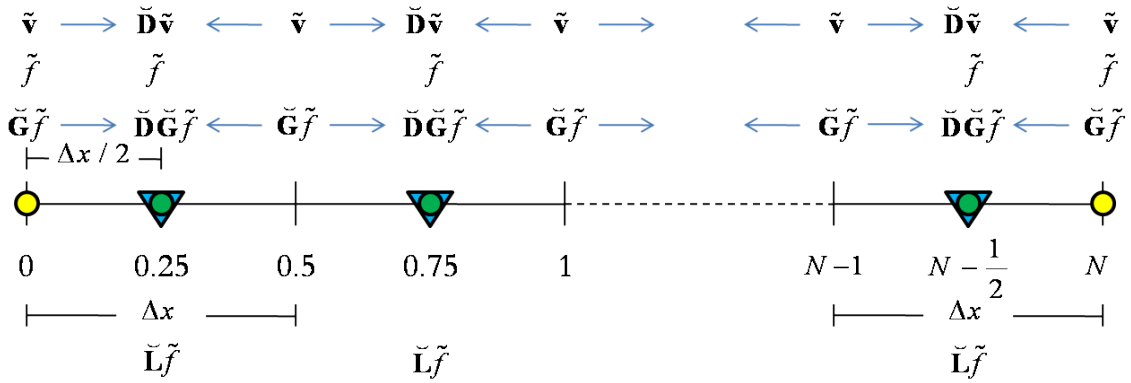



Figure 3: A one-dimensional uniform staggered grid with $N$ cells and step-size $\Delta x = 0.5$. This Figure depicts how are the approximations for the discrete Laplacian bound to the staggered grid, as explained in §3.1. Notice how this binding accords to how the operators mimic the important property: $\breve{\mathbf{L}} = \breve{\mathbf{D}}\breve{\mathbf{G}}$.

Discrete differential operators can be built following different methodologies, which arise from different manipulations of (2). Specifically, if we let $\mathbf{Q}$ be the identity matrix in $\mathbb{R}^{(N+2)\times(N+2)}$, and if we let $\mathbf{P} \in \mathbb{R}^{(N+1)\times(N+1)}$ to satisfy

$$\mathbf{P} = \begin{bmatrix} \frac{1}{2} & 0 & \cdots & \cdots & 0 \\ 0 & 1 & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & 1 & 0 \\ 0 & \cdots & \cdots & 0 & \frac{1}{2} \end{bmatrix}, \tag{3}$$

then, we can obtain the operators as defined by the Support Operators Method [26]. That is, for the case of one-dimensional approximations, we will obtain the following discrete gradient operator:

$$\tilde{\mathbf{G}} = \frac{1}{\Delta x} \begin{bmatrix} -2 & 2 & 0 & \cdots & \cdots & 0 \\ 0 & -1 & 1 & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & & \vdots \\ 0 & \cdots & 0 & -1 & 1 & 0 \\ 0 & \cdots & \cdots & 0 & -2 & 2 \end{bmatrix} \in \mathbb{R}^{(N+1)\times(N+2)}. \tag{4}$$

On the other hand, MTK's operators has been built based upon the CGM, since these yield second-order approximations, in the interior of the grid (discretized with $N$ cells), as well as in the boundaries. For the CGM, we consider (2) one more time. In this case, $\mathbf{Q}$ is the identity, and

$$\mathbf{P} = \begin{bmatrix} \frac{3}{8} & 0 & 0 & \cdots & & \cdots & 0 \\ 0 & \frac{9}{8} & 0 & \cdots & & \cdots & 0 \\ \vdots & \ddots & 1 & \ddots & & & \vdots \\ & & \ddots & \ddots & \ddots & & \\ \vdots & & & \ddots & 1 & \ddots & \vdots \\ 0 & \cdots & & \cdots & 0 & \frac{9}{8} & 0 \\ 0 & \cdots & & \cdots & 0 & 0 & \frac{3}{8} \end{bmatrix}. \tag{5}$$

Similarly, the discrete CGM-based gradient is defined as follows:

$$\breve{\mathbf{G}} = \frac{1}{\Delta x} \begin{bmatrix} -\frac{8}{3} & 3 & -\frac{1}{3} & \cdots & \cdots & 0 \\ 0 & -1 & 1 & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & & \vdots \\ 0 & \cdots & 0 & -1 & 1 & 0 \\ 0 & \cdots & \cdots & \frac{1}{3} & -3 & \frac{8}{3} \end{bmatrix} \in \mathbb{R}^{(N+1)\times(N+2)}. \tag{6}$$

The discrete CGM-based divergence operator is defined as follows:

$$\breve{\mathbf{D}} = \frac{1}{\Delta x} \begin{bmatrix} -1 & 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -1 & 1 \end{bmatrix} \in \mathbb{R}^{N\times(N+1)}. \tag{7}$$

The definitions in (4) and in (6) should be compared. Specifically, the treatment at the boundaries should be pointed out, as depicted in the top and bottom rows of (6).

It is worth mentioning that the definition of the boundary operator, in the case of the GCM, is a consequence of the definition of both the gradient and the divergence operators, as it is described

in [1]. In this train of thought, the CGM-based boundary operator (or, as it will be named later, the CGM-based Neumann operator) has the following form:

$$\breve{\mathbf{B}} = \begin{bmatrix} -1 & 0 & 0 & ... & & & 0 \\ 1/8 & -1/8 & 0 & & & & \vdots \\ -1/8 & 1/8 & 0 & & & & \vdots \\ 0 & 0 & 0 & \ddots & 0 & 0 & 0 \\ \vdots & & & & 0 & -1/8 & 1/8 \\ \vdots & & & & 0 & 1/8 & -1/8 \\ 0 & & & ... & 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{(N+2)\times(N+1)}. \tag{8}$$

Similarly, we define the Dirichlet operator, as follows:

$$\breve{\mathbf{A}} = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & \cdots & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & \cdots & 0 & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \in \mathbb{R}^{(N+2)\times(N+2)}. \tag{9}$$

Considering the nature of the examples to be considered (§5), it is important to understand that, the definition for the CGM-based mimetic Laplacian operator is:

$$\breve{\mathbf{L}} = \breve{\mathbf{D}}\breve{\mathbf{G}}. \tag{10}$$

In §5, we will understand how can these be applied to solve an specific example problem. In the next section (§4), we will present details on the computational modeling that gave rise to the collection of classes within MTK. We will explain the classes modeling the concepts presented in this section.

# 4 Mimetic Methods Toolkit

As it was mentioned in §1.1, the Mimetic Methods Toolkit (MTK) is an object-oriented Application Programming Interface that allows the intuitive implementation of Mimetic Discretization Methods for the resolution of PDEs, yielding numerical solutions that guarantee uniform order of accuracy all along the modeled physical domain, while ensuring the satisfaction of conservation laws, thereby remaining faithful to the underlying physics of the problem.

We have divided the source code according to the designated purpose the classes possess within the API. These divisions (or **concerns**) are grouped by **layers**, and are hierarchically related by the dependence they have among them (see Figure 4). One concern is said to depend on another one, if the classes it includes, rely on the classes the latter includes. Note that this relation can be symmetrical, since two concerns may depend upon each other. Figure 4 depicts the interdependence between these concerns, which deal with the following computational issues:

- In the first layer, the collection of classes containing information about fundamental constants and functioning parameters, as for example, the amount of memory required per each reallocation within dynamics data structures, are grouped in the "**Roots**" concern.

- The second layer contains information on the classes that provide a computational representation of instrumental data structures, as well as the enumerations these rely on. These concerns are called "**Enumerations**" and "**Data structures**". Some basic and common computations, as well as the classes providing basic debugging and profiling, are contained in the "**Execution tools**" concern.

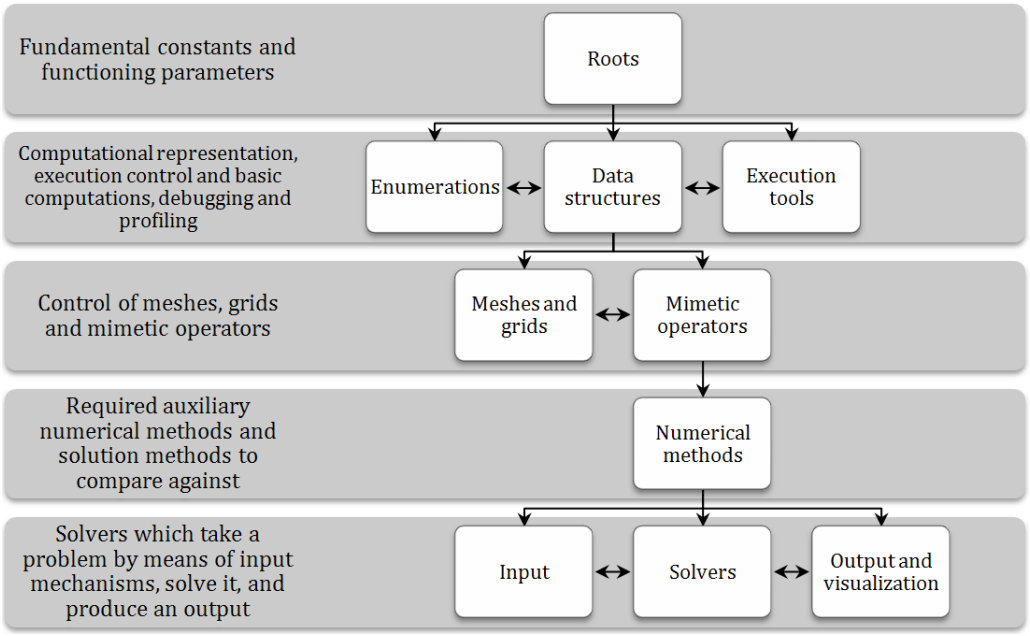



Figure 4: Summary of the MTK Concerns, grouped by layers, showing the existing interdependence among them. These are explained in §4.

- The third layer contains all the core classes of MTK, which implement the most important concepts in MDMs. Specifically, the classes in the "**Meshes and grids**" concerns, deal with the manipulation of the discretization of the physical domains users may need to consider, and the classes in the "**Mimetic operators**" concern, provide the mechanisms for the definition of the mimetic operators to be used in solving any given problem of interest.

- The fourth layer contains classes that provide auxiliary numerical methods, which are necessary for the API to provide the solution to any problem of interest. These are grouped in the "**Numerical methods**" concern. For example, the MTK provides several methods for solving the systems of equation that might arise when solving an specific Boundary Value Problem (BVP). We will mention the available methods in §5.1.2.

- The fifth layer contains all the classes that are necessary for describing problems to be solved by the MTK, as well as for the visualization of the attained solutions. Specifically, the "**Input**" concern, contains classes that are intended to gather data regarding an specific problem so that it can be solved. The "**Solvers**" concern, contains all the classes that implement the CGM (implemented by means of the classes in the previous layers), in order to solve the problem. Finally, the "**Output and visualization**" concern, contains all the classes that interface with visualization frameworks, so that the users can have an intuitive interaction with the attained solution to the problem they are interested in.

In this section, and for the sake of presenting a concise explanation of the computational modeling of the most important concept in MDMs, we will discuss the core concerns of the MTK. We will present, when considered necessary, examples of source code that clearly depict the intended usage of a particular class. In these snippets of code, we will omit most of the technicalities that are inherent of implementing the ideas in a low-level programming language, but this will hopefully increase the readability and the chances of transmitting the correct message. Furthermore, the reader interested in any of the details pertaining any of the classes, for the sake of their application towards solving an specific problem, can refer to the online documentation [27] for the broadest extent of each of the classes.

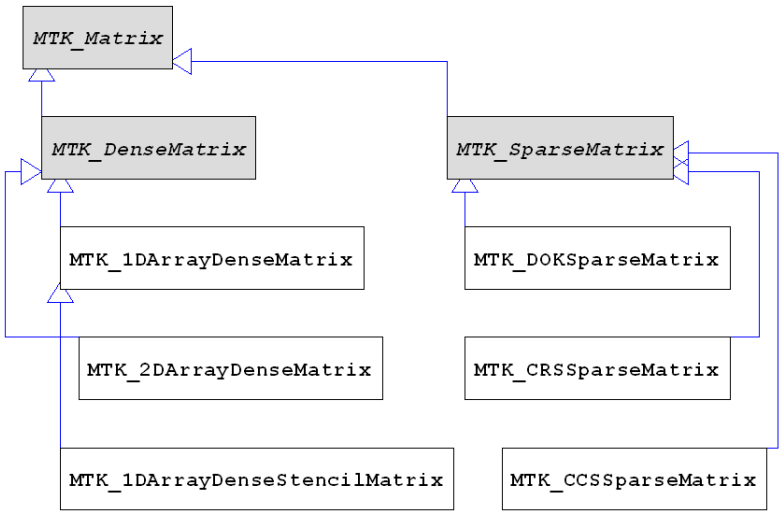



Figure 5: Simplified UML class diagram for the implemented data-structures within the MTK's "Data structures" concern, located in the second layer (see Figure 4). These are explained in §4.1.

## 4.1 Data Structures and Meshes within the MTK

Meshes within the MTK are one of the most important classes, since these contain the information regarding the discretization of the physical domain of interest. As it can be seen in Figure 4, meshes and grids rely strongly in the defined data structures, which posses their own concern. So far, the MTK posses the implementation of the data structures depicted in Figure 5.

We have decided to worry about developing those specific data structures, given the diverse computational tools we want the MTK to be compatible with. For example, the Compressed Row (CRS) and the Compressed Column Storage (CCS) sparse matrix format, are fully compatible with both [11] and [12]. Similarly, the Dictionary of Keys (DOK) Sparse Matrix Format is fully compatible with [28]. Finally, the one-dimensional array dense matrix formats are fully compatible with both [10] and [13].

Figure 6, depicts all the classes modeling meshes and grids, that have been implemented thus far. In the MTK, all the grids are thought as being "Logically Rectangular", following the ideas presented in [29]. Logically rectangular (LR) means that, despite of the fact that they may be implementing different coordinate systems, in the physical domain, these still allow for a rectangular representation within the logical, or computational domain. Similarly, Figure 6 shows that LR one-, two and three-dimensional grids inherit from the general concept of a grid. Particularly, the classes `MTK_2DGrid` and `MTK_3DGrid` have not yet been implemented, thus are depicted without a border line.

For one-dimensional grids, we have done most of the implementation focusing on uniform grids. Specifically, we have implemented nodal grids (see Figure 1), which intent to perform a non-staggered discretization of the physical domain of interest. These are provided solely for the use of auxiliary numerical methods that are provided within the MTK, for the purpose of comparing results (see Figure 4). Similarly, we provide support for staggered grids, grids containing the discretization of the source terms of the equations to solve, as well as grids intended to hold any analytical, or "known" solution, which can be also used for comparison purposes. Algorithm 1, presents an example on how can the user define a one-dimensional uniform nodal and known-solution-holder grids. Analogously, some work is being done to implement non-uniform grids that can represent information in curvilinear coordinates.

Finally, all of the grids contain instances of nodes, which hold the data of interest, modeled as an `MTK_Number`. This generic data type, intents to allow the user for a selective installation of the MTK in single or double precision arithmetic, that is `MTK_Number` $\in$ {`float`, `double`}. In §5 we will present examples of how can these different grids be used in solving an specific problem of interest.

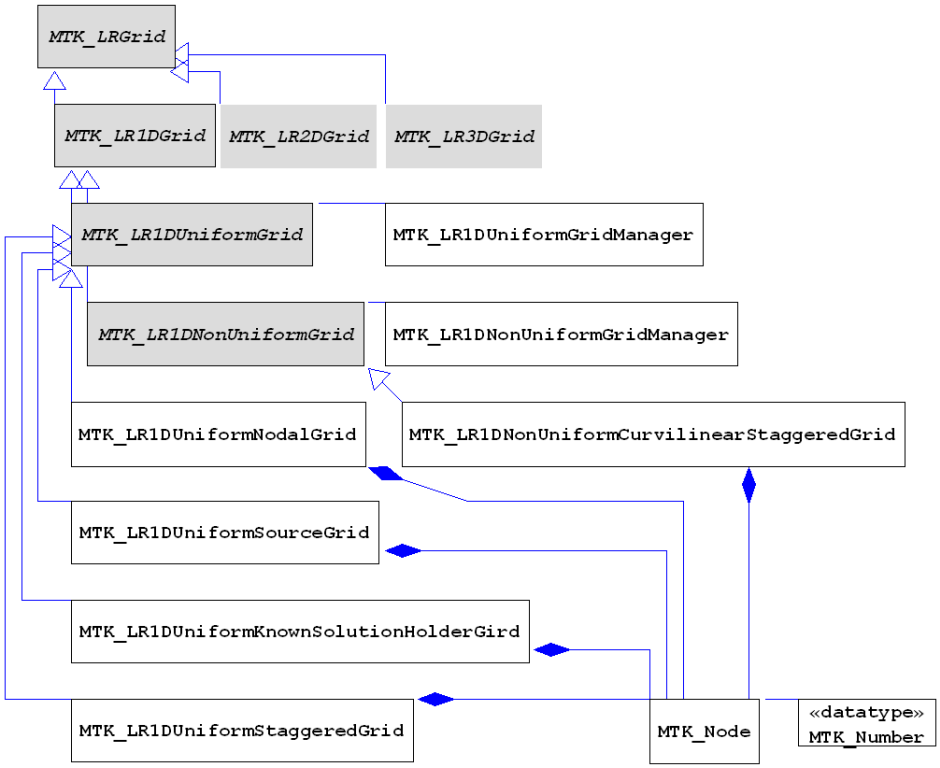



Figure 6: Simplified UML class diagram for the implemented meshes- and grids-related mechanisms within the MTK's "Meshes and grids" concern, located in the third layer (see Figure 4). These are explained in §4.1.

**Algorithm 1** Defining 1-D uniform grids in the MTK, as explained in §4.1.

```
1  MTK_Number aa;  // West boundary.
2  MTK_Number bb;  // East boundary.
3  MTK_Number step_size;
4  MTK_LR1DUniformNodalGridDouble *nodal_grid;
5  MTK_LR1DUniformKnownSolutionHolderGrid *solution_grid;
6
7  aa = 0.0;
8  bb = 5.0;
9  step_size = 0.0001;
10
11 nodal_grid =
12   new MTK_LR1DUniformNodalGrid(aa, bb, step_size);
13
14 cout << "Nodal grid:" << endl;
15 // Print in the format of a one-dimensional array:
16 nodal_grid->Print1DArray();
17
18 solution_grid = new MTK_LR1DUniformKnownSolutionHolderGrid(aa, bb, step_size,
19                                                            &known_sol_function);
20 cout << "Solution holder grid:" << endl;
21 solution_grid->Print1DArray();
```

## 4.2 Mimetic Operators within the MTK

Mimetic operators within the MTK, as it was previously mentioned, are built based on the Castillo–Grone Method [1], [3]. The mimetic operators posses their development concern (Figure 4), and are implemented (for now) as instances of dense matrices, as it can be seen in Figure 7.

Specifically, we have developed instances of the CGM-based gradient, $\breve{\mathbf{G}}$, as well as operators for the boundaries. For the sake of developing a theoretically consistent API, we have differentiated the boundary operators into the Dirichlet[9] and the Neumann[10] boundary operators. If we consider the generalized form of the Robin[11] boundary conditions, in a one-dimensional scenario, for a one-dimensional domain $\Omega = [a, b] \subset \mathbb{R}$, we have:

$$\alpha f(a) - \beta f'(a) = \omega, \tag{11}$$

$$\gamma f(b) + \delta f'(b) = \epsilon, \tag{12}$$

where the values $\alpha$ and $\beta$, are called (within the MTK) the Dirichlet and Neumann coefficients for the west boundary, respectively. Similarly, the value $\omega$ denotes the value of the boundary condition at the west boundary. Analogously, the values $\gamma$ and $\delta$, are called (within the MTK) the Dirichlet and Neumann coefficients for the east boundary, respectively, and the value $\epsilon$ denotes the value of the boundary condition at the east boundary.

The identification of these quantities and their role within the resolution of an specific problem is very important within the MTK, as it can be seen in §4.3. When it comes to defining discrete boundary operators, it is clear that the generalized Robin boundary conditions can be written, in a discrete form, and for a discrete scalar-valued quantity $\tilde{f}$, as follows:

$$(\mathbf{d}\breve{\mathbf{A}} + \mathbf{n}\breve{\mathbf{B}}\breve{\mathbf{G}})\tilde{f}^T = \tilde{z}^T, \tag{13}$$

where $\mathbf{d}$ is the vector of Dirichlet coefficients, defined as $\mathbf{d} \triangleq [\alpha, 0, ..., 0, \gamma]^T$, $\mathbf{n}$ is the vector of Neumann coefficients, defined as $\mathbf{n} \triangleq [\beta, 0, ..., 0, \delta]^T$, $\tilde{z}^T = [\omega, 0, ..., 0, \epsilon]^T$, and where $\breve{\mathbf{A}}$ and $\breve{\mathbf{B}}$ denote the Dirichlet boundary operator and the Neumann operator, respectively. Notice that if $\alpha = \gamma$ or $\beta = \delta$, then we will have that $\mathbf{d} = \alpha$ or $\mathbf{n} = \beta$, respectively. In §5.1.1, we will explain the role of (13) in conjunction with other mimetic operators, in solving an example problem. Similarly, in §5.1, we will pragmatically explain how is this problem description implemented within the MTK, and how should it be manipulated by a client code.

Finally, Figure 7 shows that we support the definition of the CGM-based divergence, $\breve{\mathbf{D}}$, and the CGM-based Laplacian, $\breve{\mathbf{L}}$ operators. Thus far, we have only implemented second-order operators within the MTK, nonetheless, research works, as for example [1] and [3], have successfully achieved definitions for higher-order operators.

## 4.3 Input, Problem Solvers and Output within the MTK

As we have mentioned in the previous section, an important layer of development concerns within the MTK, is the fifth layer, which contains all the classes that model data input, problem solvers and data output. So far, we have implemented input methods for solving general elliptic equations of the form:

$$-\nabla \cdot (\mathscr{K}(\mathbf{x})\nabla f) = F(\mathbf{x}), \tag{14}$$

where

$$\mathscr{K}(\mathbf{x}) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \tag{15}$$

however, the classes can be combined with the user's own client algorithm to allow for the solution of different equations.

[9] Johann Peter Gustav Lejeune Dirichlet - Born in Düren, French Empire on February 13, 1805 - Perished in Göttingen, Kingdom of Hanover on May 5, 1859 (aged 54).

[10] Carl Gottfried Neumann - Born in Königsberg, Prussia on May 7, 1832 - Perished in Leipzig, Saxony, Germany on March 27, 1925 (aged 92).

[11] Victor Gustave Robin - Born in Paris, France on May 17, 1855 - Perished in 1897 (aged 42).

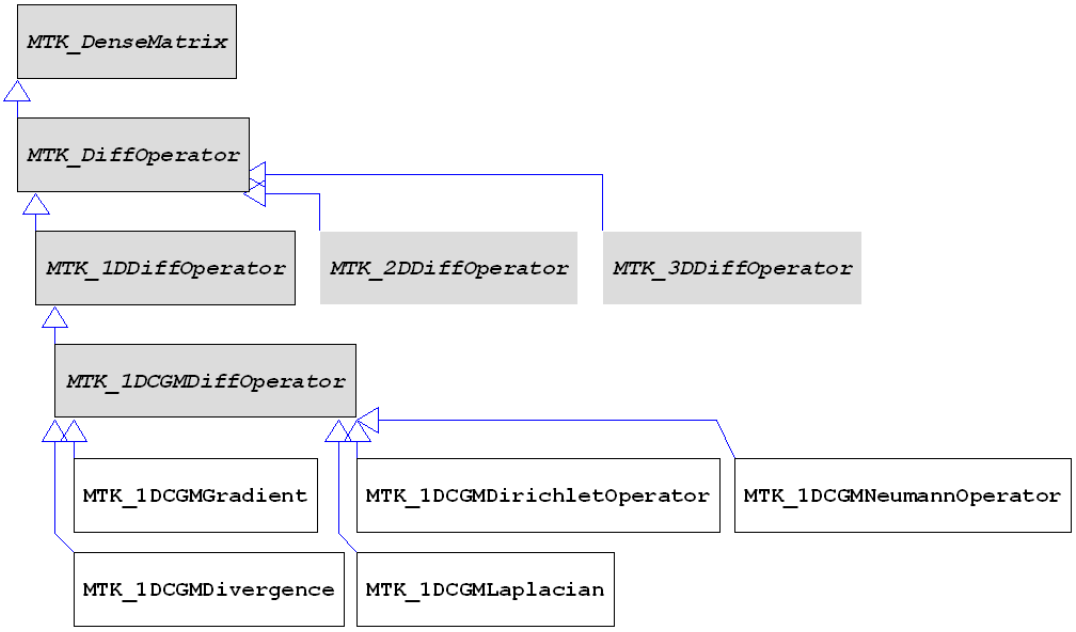



Figure 7: Simplified UML class diagram for the modeling of mimetic operators, as explained in §4.2.

The most important class in terms of data input, is the `MTK_Steady1DProblem`. This class allows to specify all the parameters that describe an steady-state one- dimensional problem to be solved by means of the CGM. Any client code, can specify a problem by instantiating the related class:

```
problem_to_solve = new MTK_Steady1DProblem(alpha, beta, west_bndy_value,
                                           gamma, delta, east_bndy_value);
```

The MTK implements other ways of providing the information for the solution of Initial Value Problems (IVPs), as well, simply for the purpose of comparison. For example, Algorithm 2, shows how can a client code, provide the information for an IVP of interest. Similarly, the MTK interfaces with [30], to provide classes assisting in visualization of the elements being used. For example, in Algorithm 2 the class `MTK_1DPlotter`, is used to graphically depict the solution of the problem under consideration. Section 5 will present the usages of the previously mentioned classes, within the context of implementing the solution based on the CGM, for an specific problem of interest.

## 4.4 Memory Usage in the MTK

In this section, we provide some estimates for the memory usage of the most important classes within the MTK. Our intention is to try to understand the memory requirements of the objects of theses classes, as a function of the required number of cells used in discretizing the physical domain.

We present the tabulated results in Tables 2, 3, and 4, for which we have expressed such results in terms of the `MTK_Number` data type. In the upcoming discussion, we will denote the number of rows and columns in the considered data structures, as $r$ and $c$, respectively. Similarly, we will denote the number of nonzero elements as $\eta$, and the number of cells describing the discretized domain, as $N$.

We also pay attention to the memory requirements of the linear system of equations that arise when solving BVPs by means of the CGM.

### 4.4.1 Estimating memory usage of the `MTK_System` objects

A particularly important class is `MTK_System`. This class, takes an stencil matrix, of size $(N+2)^2$, where $N$ is the number of cells based on which we have chosen to discretize our physical domain, and

**Algorithm 2** Use of an additional provided method to solve a sample problem while customizing a visual output with the MTK, as explained in §4.3.

```
MTK_Number aa = 0.0, bb = 5.0, time_step = 0.1, initial_value = 1.0;

nodal_grid = new MTK_LR1DUniformNodalGrid(aa, bb, time_step);
cout << "Nodal grid:" << endl;
nodal_grid->Print1DArray();

solution_grid =
  new MTK_LR1DUniformKnownSolutionHolderGrid(aa, bb, time_step);
cout << "Solution holder grid:" << endl;
solution_grid->Print1DArray();

converge = solution_grid->SolveInitialValue(MTK_HEUN, initial_value, &rhs);

if (converge) {
  plotter = new MTK_1DPlotter(solution_grid->independent(), solution_grid->
      dependent(), solution_grid->number_of_nodes());
  save_plot = false;
  props = new MTK_1DPlotProperties("t", "f(t)", "Attained solution", "linespoints",
                                   "red");
  plotter->set_plot_properties(props, save_plot, MTK_PNG);
  plotter->See();
} else {
  cout << "No converge was achieved." << endl;
}
```

| Object | Allocated memory (in bytes) |
|---|---|
| MTK_1DArrayDenseMatrix | $r \times c \times$`sizeof(MTK_Number)` |
| MTK_2DArrayDenseMatrix | $r \times c \times$`sizeof(MTK_Number)`$+r \times c \times$`sizeof(int)` |
| MTK_1DArrayDenseStencilMatrix | $(N+2) \times (N+2) \times$`sizeof(MTK_Number)`, with $N = r = c$ |
| MTK_DOKSparseMatrix | $\eta \times$`sizeof(MTK_Number)`$+2 \times \eta \times$`sizeof(int)` |
| MTK_CRSSparseMatrix | $\eta \times$`sizeof(MTK_Number)`$+(\eta + r + 1) \times$`sizeof(int)` |
| MTK_CCSSparseMatrix | $\eta \times$`sizeof(MTK_Number)`$+(\eta + c + 1) \times$`sizeof(int)` |

Table 2: Memory used per instance of the classes in the "Data structures" concern, described in §4.1. See §4.4.

| Object | Allocated memory (in bytes) |
|---|---|
| MTK_Node | $O($`sizeof(MTK_Number)`$)$ |
| MTK_LR1DUniformNodalGrid | $(N+2) \times$`sizeof(MTK_Node)` |
| MTK_LR1DUniformSourceGrid | $(N+2) \times$`sizeof(MTK_Number)` |
| MTK_LR1DUniformKnownSolutionHolderGrid | $(N+2) \times$`sizeof(MTK_Number)` |
| MTK_LR1DUniformStaggeredGrid | $(N+2) \times$`sizeof(MTK_Node)` |
| MTK_LR1DNonUniformCurvilinearStaggeredGrid | $(N+2) \times$`sizeof(MTK_Node)` $+(N+2) \times$`sizeof(MTK_Number)` |

Table 3: Memory used per instance of the classes in the "Meshes and grids" concern, described in §4.1. See §4.4.

| Object | Allocated memory (in bytes) |
|---|---|
| MTK_1DCGMGradient | $(N+1)\times(N+2)\times$`sizeof(MTK_Number)` |
| MTK_1DCGMDirichletOperator | $(N+2)\times(N+2)\times$`sizeof(MTK_Number)` |
| MTK_1DCGMNeumannOperator | $(N+2)\times(N+1)\times$`sizeof(MTK_Number)` |
| MTK_1DCGMDivergence | $(N+2)\times(N+1)\times$`sizeof(MTK_Number)` |
| MTK_1DCGMLaplacian | $(N+2)\times(N+2)\times$`sizeof(MTK_Number)` |

Table 4: Memory used per instance of the classes in the "Mimetic operators" concern, described in §4.2. See §4.4.

| $N$ | Used memory (in kB) |
|---|---|
| 1,000 | 55,364 |
| 1,500 | 123,852 |
| 2,000 | 219,624 |
| 2,500 | 342,832 |
| 3,000 | 493,324 |
| 3,500 | 671,336 |
| 4,000 | 876,520 |
| 4,500 | 1,109,036 |
| 5,000 | 1,368,904 |
| 5,500 | 1,656,148 |
| 6,500 | 2,312,740 |
| 7,500 | 3,078,692 |
| 8,000 | 3,502,536 |
| 10,000 | 5,471,900 |

Table 5: Memory used per instance of the `MTK_System` class. See §4.4.1.

constructed in terms of the mimetic operators, a discrete collection of the source term evaluations (i.e. a `MTK_LR1DUniformSourceGrid` object), and a solution-holder grid (see Figure 6), both implemented as a one- dimensional array of size $(N+2)$. This class computes the final solution to the BVP, using any specified solver algorithm. Considering the input parameters, it is expected that the required amount of memory (in Bytes) should be described by a second-degree polynomial, as follows:

$$\texttt{sizeof(MTK\_Number)} \times ((N+2)^2 + 2(N+2)) = \texttt{sizeof(MTK\_Number)} \times (N^2 + 6N + 8), \quad (16)$$

where $\texttt{MTK\_Number} \in \{\texttt{float}, \texttt{double}\}$ (see Figure 6). We performed an empirical study of memory usage by the `MTK_System` class and the results are given in Table 5 and in Figure 8.

MTK tests were performed in a 1.73 GHz Intel® Pentium® M processor with 0.99 GB of RAM, running Linux Kubuntu 12.04, the Precise Pangolin release, and running Microsoft Windows XP Home Edition, Version 2002, Service Pack 3 using MATLAB R2010b, for the data-fitting study, which reported a fitting polynomial of:

$$p(N) = k(N^2 + 4.991N + 6,848.892), \quad (17)$$

where $k = 0.0547$, and depends on both the definition of the `MTK_Number` and the architecture. In these tests, we used $\texttt{MTK\_Number} = \texttt{double}$.

# 5 Application examples and MTK's usage philosophy

In this section, by means of examples, we present the details on how to use the MTK.

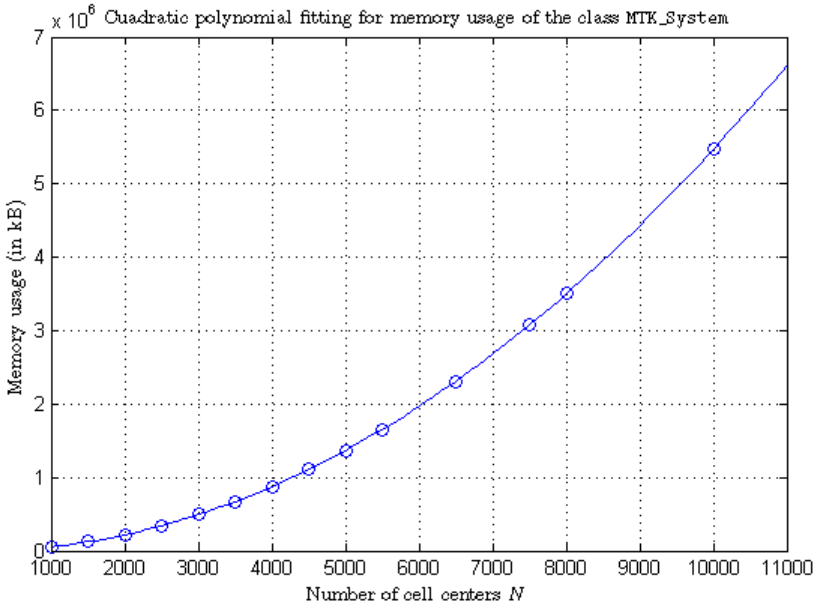



Figure 8: Second-order polynomial fitting for the memory usage estimate for objects of the class `MTK_System`. See §4.4.1.

## 5.1 A one-dimensional example on a uniform staggered mesh with Robin's boundary conditions

Consider:

$$-\nabla \cdot (\mathscr{K}(\mathbf{x}, \lambda)\nabla p) = F(\mathbf{x}, \lambda). \tag{18}$$

Let

$$\mathscr{K}(\mathbf{x}, \lambda) = \mathscr{K}(x, \lambda) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{19}$$

Similarly, let [4]

$$F(\mathbf{x}, \lambda) = F(x, \lambda) = -\frac{\lambda^2 \exp(\lambda x)}{\exp(\lambda) - 1}. \tag{20}$$

Equation (18) now takes the form:

$$-\nabla \cdot \nabla p = -\frac{\lambda^2 \exp(\lambda x)}{\exp(\lambda) - 1} = F(x, \lambda); \tag{21}$$

therefore, yielding the following instance of a one-dimensional Poisson's Equation [4]:

$$-\nabla^2 p(x) = F(x, \lambda), \tag{22}$$

where

$$F(x, \lambda) = -\frac{\lambda \exp(\lambda x)}{\exp(x) - 1}, \tag{23}$$

will stand for our source term. Consider the following BVP, with Robin boundary conditions defined over $\Omega = [a, b]$:

$$\alpha p(a) - \beta p'(a) = \omega \tag{24}$$

$$\alpha p(b) + \beta p'(b) = \epsilon, \tag{25}$$

where $\omega = -1$, $\epsilon = 0$, $\alpha = -\exp(\lambda)$ and $\beta = (\exp(\lambda) - 1)/\lambda$.

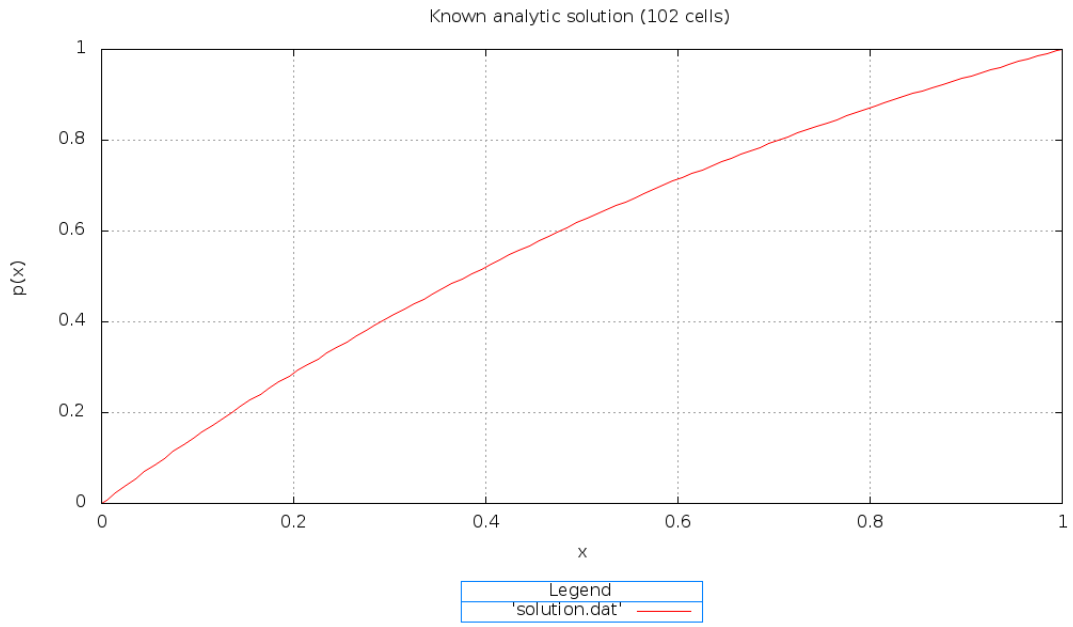



Figure 9: Known analytical solution for example problem number one. A uniform nodal grid with 102 cells was used to generate this plot, be means of the MTK's visualization mechanisms. The context is explained in §5.1.

If we take $\Omega = [0, 1] \subset \mathbb{R}$ and $\lambda = -1$, the problem has known analytical solution [4] given by (See Figure 9):

$$p(x) = \frac{e^{\lambda x} - 1}{e^{\lambda} - 1} = \frac{e^{-x} - 1}{e^{-1} - 1}. \tag{26}$$

### 5.1.1 Reference solution attained by means of the CGM

Before presenting the solution to this problem by means of the MTK, we will present a reference solution, which will also assist in explaining the discretization procedure through a CGM- based MDM. We are interested in solving the following continuous problem:

$$-\nabla^2 p(x) = F(x, \lambda), \tag{27}$$

by means of using CGM-based mimetic operators. Specifically, we are interested in implementing a CGM-based Laplacian operator. If we mimic the continuous problem presented in (27), we will obtain the following mimetic analog:

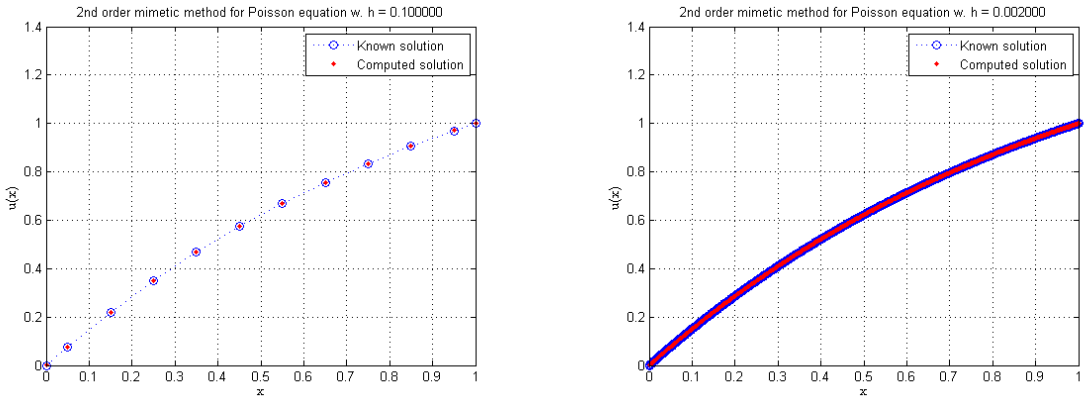
$$-\breve{\mathbf{L}}\tilde{p}^T = \tilde{F}^T. \tag{28}$$

Notice that we have only replaced the continuous operators by their mimetic counterparts. However we are interested in including the information of the discretization at the boundaries, which we can do by means of adapting (13) to our problem of interest. For our problem, letting $\alpha = \gamma$ and $\beta = \delta$, we will have that $\mathbf{d} = \alpha$ and $\mathbf{n} = \beta$, respectively; therefore, (13) will now read:

$$(\alpha\breve{\mathbf{A}} + \beta\breve{\mathbf{B}}\breve{\mathbf{G}})\tilde{p}^T = \tilde{F}^T. \tag{29}$$

In order for us to combine the information in both (28) and (29), given the dimension of the involved discrete operators, we need to define the following augmented form for $\breve{\mathbf{L}}$:

$$\hat{\breve{\mathbf{L}}} = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \breve{\mathbf{L}} & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \in \mathbb{R}^{(N+2)\times(N+2)}, \tag{30}$$




(a) 10 cells.

(b) 500 cells.

Figure 10: MATLAB R2010b reference solution mentioned in §5.1.1, using a second- order CGM-based MDM. In Figure (a) only 10 cells were used, and it shows how is the solution bound to the cell centers, as it is expected from Figure 3. In Figure (b), 500 cells were used, thus yielding a more accurate solution.

where $\breve{\mathbf{L}}$ has been previously defined in (10). Given this augmented operator, dimensions now allow us to define the stencil matrix, for generalized Robin boundary conditions, based on the CGM, $\mathbf{S}$, as follows:

$$\mathbf{S} \triangleq \alpha\breve{\mathbf{A}} + \beta\breve{\mathbf{B}}\breve{\mathbf{G}} - \hat{\breve{\mathbf{L}}} \tag{31}$$

therefore, the solution to our problem lies in solving the following system of linear equations, of rank $(N+2)$:

$$\mathbf{S}\tilde{p}^T = \tilde{F}^T, \tag{32}$$

where $\tilde{F} = [\omega, F(x_{1/2}), ..., F(x_{i+1/2}), ..., F(x_{N-1/2}), \epsilon]$, is the discretized collection of values for our source term. The reader may refer to [1] and [4], for a more detailed description of the arising systems of equation, such as (32).

As a reference solution, we have implemented the scheme in MATLAB R2010b, yielding the reference solution depicted in Figure 10. Figure 11 summarizes the results of a grid refinement study which was performed with the intention of analyzing the actual order of accuracy that we obtained by means of the CGM. We have performed this study not only along the entire grid, but also at the boundaries, thus depicting the uniformity of the achieved accuracy, all along the discretized domain. In each case, we present results in terms of the number of cells, $N$, as well as in terms of the grid step size $\Delta x$.

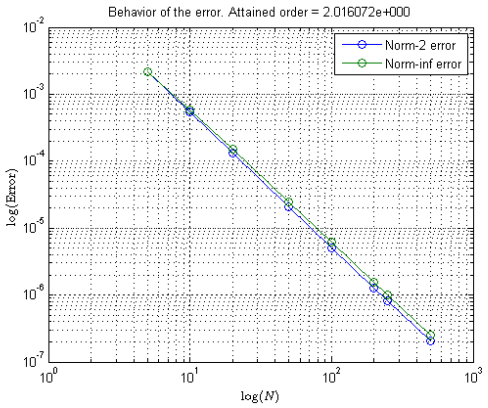
For this study, we have computed the attained order of accuracy by computing the slope of the attained linear relationship defined by the computed relative 2-norm errors in the log-log space. By solving the sample problem for several grid step sizes, we were able to collect the attained relative 2-norm of the error with respect to the known solution:

$$\frac{||\tilde{p}_k - \tilde{p}_c||_2}{||\tilde{p}_k||_2}, \tag{33}$$

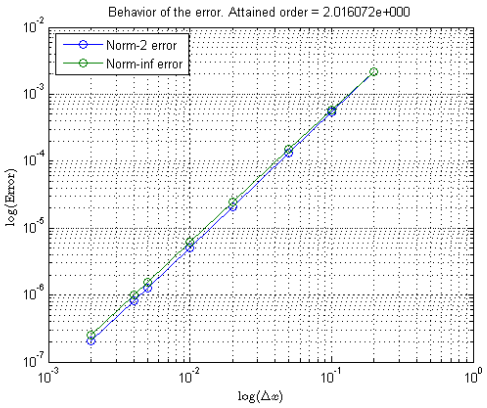
where $\tilde{p}_k$ and $\tilde{p}_c$ denote the discretized known analytical solution and the computed reference solution from the CGM, respectively. We also used th inifinity norm as implemented in MATLAB R2010b. The slopes were computed using the first and the last sample.

### 5.1.2 CGM-based solution implemented with the MTK

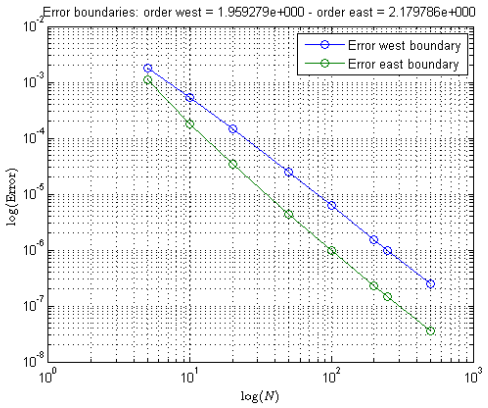
In order to be as concise as possible, we will present the algorithmic approach for solving the first example problem, as we describe the proposed algorithm as implemented by the MTK.



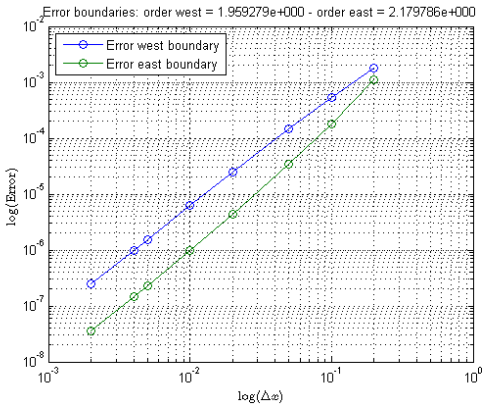

(a) Entire grid, in terms of $N$.




(b) Entire grid, in terms of $\Delta x$.




(c) Boundary, in terms of $N$.




(d) Boundary, in terms of $\Delta x$.

Figure 11: Grid refinement study described in §5.1.1, depicting the attained order achieved by the CGM-based MDM reference solution, computed using MATLAB R2010b. Figures (a) and (b), present the behavior of the error all along the entire grid, whereas Figures (c) and (d) present the behavior of the error at the boundary. As it can be seen, an uniform accuracy prevails at both the interior nodes and the boundary.

The first step is to define the functions implementing the source function of the equation and the analytic solution (if known), for comparison purposes. Then, in cases like the example currently under consideration, we proceed as follows:

```
MTK_Number source_term (MTK_Number xx) {

  MTK_Number lambda;

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

MTK_Number known_solution (MTK_Number xx) {

  MTK_Number lambda;

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

These should be compared with (23) and (26). For the example of interest, the following classes from MTK should be used:

```
  MTK_ToolManager *tools;
  MTK_LR1DUniformStaggeredGridDouble *space;
  MTK_LR1DUniformSourceGrid *source;
  MTK_1DCGDirichletOperator *dir_comp;
  MTK_1DCGNeumannOperator *neu_comp;
  MTK_1DCGGradient *grad;
  MTK_1DCGLaplacian *lap;
  MTK_Steady1DProblem *problem_to_solve;
  MTK_1DStencilMatrix *stencil_matrix;
  MTK_LR1DUniformKnownSolutionHolderGridDouble *solution_grid;
  MTK_1DPlotProperties *props;
  MTK_1DPlotterDouble *plotter;
```

The following auxiliary variables are subsequently considered in order to help define the problem of interest:

```
  number_of_cell_centers = 50;
  desired_order = 2;
  lambda = -1.0;
  west_bndy = 0.0;
  east_bndy = 1.0;
  alpha = -exp(lambda);
  beta = (exp(lambda) - 1.0)/(lambda);
  west_bndy_value = -1.0;
  gamma = alpha;
  delta = beta;
  east_bndy_value = 0.0;
```

We are now able to create the required mimetic operators, as follows:

```
  dir_comp = new MTK_1DCGDirichletOperator(desired_order,
                                           number_of_cell_centers,
                                           MTK_DENSE);
  neu_comp = new MTK_1DCGNeumannOperator(desired_order,
                                         number_of_cell_centers,
                                         MTK_DENSE);
  grad = new MTK_1DCGGradient(desired_order,
                              number_of_cell_centers,
                              MTK_DENSE);
  lap = new MTK_1DCGLaplacian(desired_order,
```

```
                                        number_of_cell_centers,
                                        MTK_DENSE);
```

Similarly, we can define the problem of interest, as follows:

```
problem_to_solve = new MTK_Steady1DProblem(alpha, beta, west_bndy_value,
                                           gamma, delta, east_bndy_value);
```

For the case of $N = 6$, MTK yields the following numerical instance of the problem to solve:

```
Problem of interest:
-0.367879 f(a) - 0.632121 f'(a) = -1.0
-0.367879 f(b) + 0.632121 f'(b) = 0.0
```

We are now able to proceed with the creation of the required grids:

```
space = new MTK_LR1DUniformStaggeredGridDouble(west_bndy, east_bndy,
                                               number_of_cell_centers);
source = new MTK_LR1DUniformSourceGrid(space, west_bndy_value,
                                       east_bndy_value, source_term);
```

Note that the values of the source function (23) are provided, as well as the values for the discretized spatial coordinates. At this point, the MTK can now proceed with the creation of the stencil matrix and the system, which solution represents our desired solution for (22). This is easily done by instructing:

```
stencil_matrix = new MTK_1DStencilMatrix(alpha, dir_comp,
                                         beta, neu_comp,
                                         grad, lap);
MTKSystem system(stencil_matrix, source);
system.Solve();
```

For the case of $N = 6$, MTK allows us to obtain the following numerical instance for the system of equations to solve:

```
Arising system of equation:
|   8.1   -9.5    1.1     -0     -0     -0     -0  |     | -1.00 |
|-66.67 100.00 -33.33  -0.00  -0.00  -0.00  -0.00  |     | 1.43  |
| -0.00 -25.00  50.00 -25.00  -0.00  -0.00  -0.00  |     | 1.17  |
| -0.00  -0.00 -25.00  50.00 -25.00  -0.00  -0.00  | x = | 0.96  |
| -0.00  -0.00  -0.00 -25.00  50.00 -25.00  -0.00  |     | 0.79  |
| -0.00  -0.00  -0.00  -0.00 -33.33 100.00 -66.67  |     | 0.64  |
|  0.00   0.00   0.00   0.00   1.05  -9.48   8.06  |     | 0.00  |
```

The `Solve` member function implements a numerical method for solving the system. Since the operators were created using a dense representation for the storage format, a simple Gaussian elimination with backward substitution was utilized to solve the system [31]. However, full compatibility with known numerical linear algebra libraries is provided by the MTK. In fact, if we analyze the implementation of the constructor for the `MTK_1DStencilMatrix` class, we can see that an ATLAS-optimized CBLAS [13] is used to perform all the required matrix operations, according to the definition of the stencil matrix $\mathbf{S}$, which is a function of the mimetic matrix operators [4]. The user can then visualize the required results graphically:

```
plotter = new MTK_1DPlotterDouble(solution_grid->independent(),
                                  system.Solution(),
                                  solution_grid->number_of_nodes());
save_plot = false;
```

```
loglog = false;
props = new MTK_1DPlotProperties("x",
                                 "u(x)",
                                 "Computed solution",
                                 "points",
                                 "blue");
plotter->set_plot_properties(props, save_plot, loglog, MTK_PNG);
plotter->See();
```

Furthermore, in order to visualize the known solution to this problem (for comparison purposes), we can now create a solution holder grid, as follows (see Figure 9):

```
solution_grid =
new MTK_LR1DUniformKnownSolutionHolderGridDouble(space,
                                                 &known_solution);
plotter = new MTK_1DPlotterDouble(solution_grid->independent(),
                                  solution_grid->dependent(),
                                  solution_grid->number_of_nodes());
save_plot = false;
props =
new MTK_1DPlotProperties("x", "u(x)", "Known solution", "lines", "red");
plotter->set_plot_properties(props,save_plot,MTK_PNG);
plotter->See();
```

Finally, thanks to the tools within the MTK, we are able to numerically compare the error among the two solutions at hand:

```
tools = new MTK_ToolManager();
norm_diff_sol =
tools->RelativeNorm2Difference(solution_grid->dependent(),
                               system.Solution(),
                               solution_grid->number_of_nodes());
cout << "Relative 2-norm of the difference = " << norm_diff_sol << endl;
return EXIT_SUCCESS;
```

Using MTK to solve this problem yields a computed solution depicted in Figure 12, which show that the computed solution is indeed bound to the cell centers, as it is to be expected given the nature of the mimetic Laplacian that was approximated by the operators provided within the MTK. This already shows the correctitude of the implementation of these methods by the MTK. However, we will support this conclusion by means of a grid refinement study, presented in the next section (§5.1.3).

### 5.1.3 The Accuracy of the CGM-based MDM as implemented by the MTK

In this section, we present a grid refinement study to show that the implement solution by means of the MTK, also provides the same order of accuracy as the reference solution, previously studied.

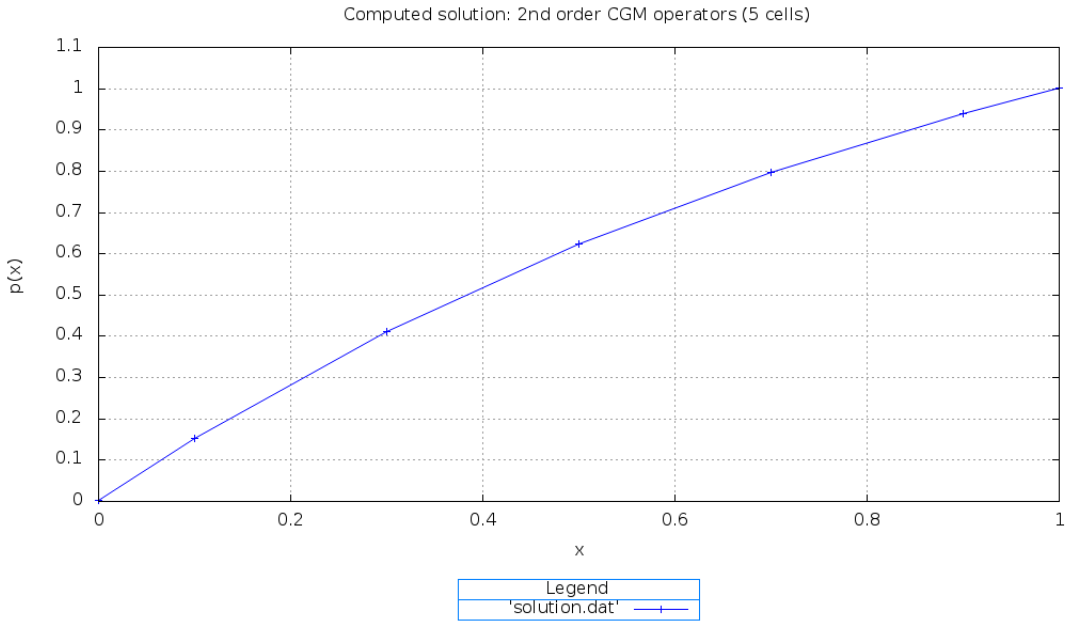
We discretized the known solution using an `MTK_KnownSolutionHolderGrid`. Tables 6 and 7, shows the results of the accuracy study by means of a grid refinement process.

Specifically, Figure 13, depicts the results summarized in Table 6. Specifically, Figure 13 shows the attained accuracy of the numerical solution implemented via the MTK. As in the case of our reference solution, we present these results in terms of both the number of cells, $N$, and the grid step size, $\Delta x$, that were used to compute the solution. Analogously, Figure 14 depicts the results summarized in Table 7.

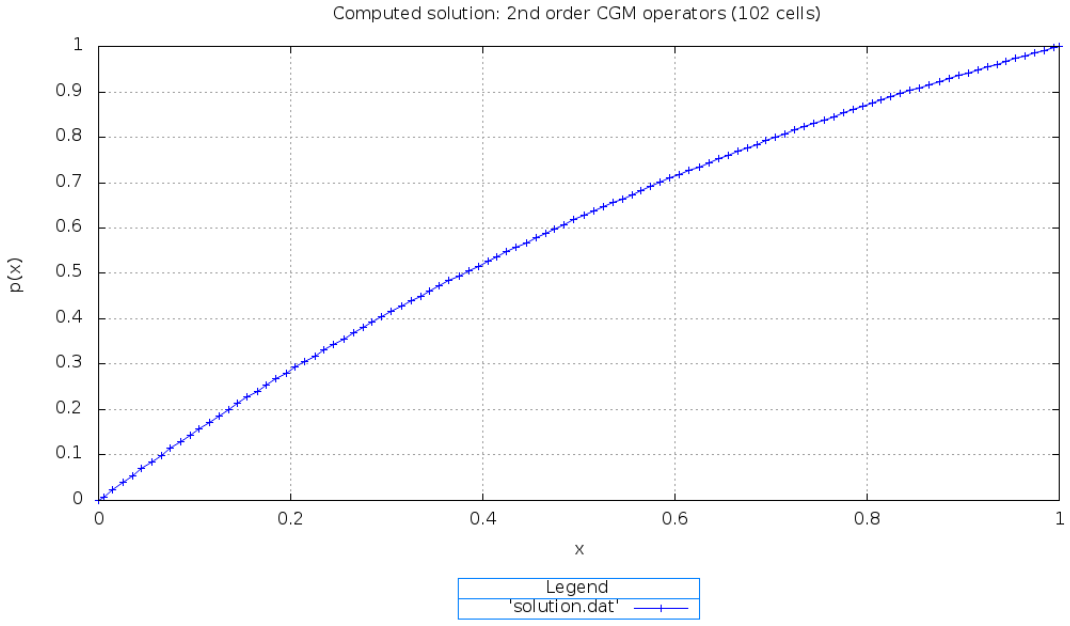
## 5.2 A second example

In this example, we consider:

$$-\nabla^2 f(x) = F(x), \tag{34}$$

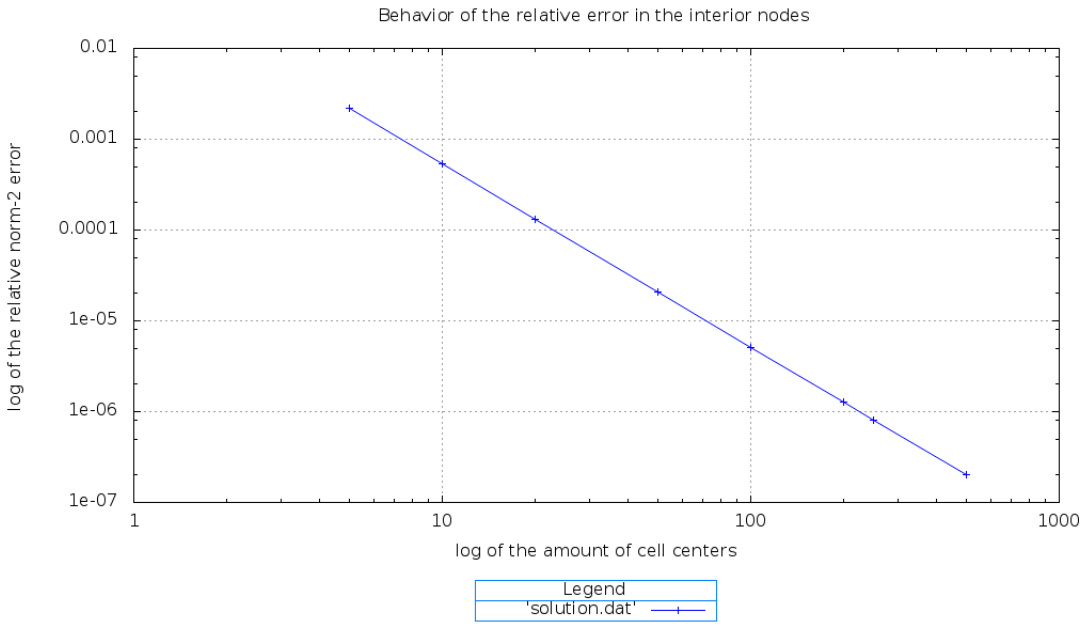



(a) Solution with $N = 5$.

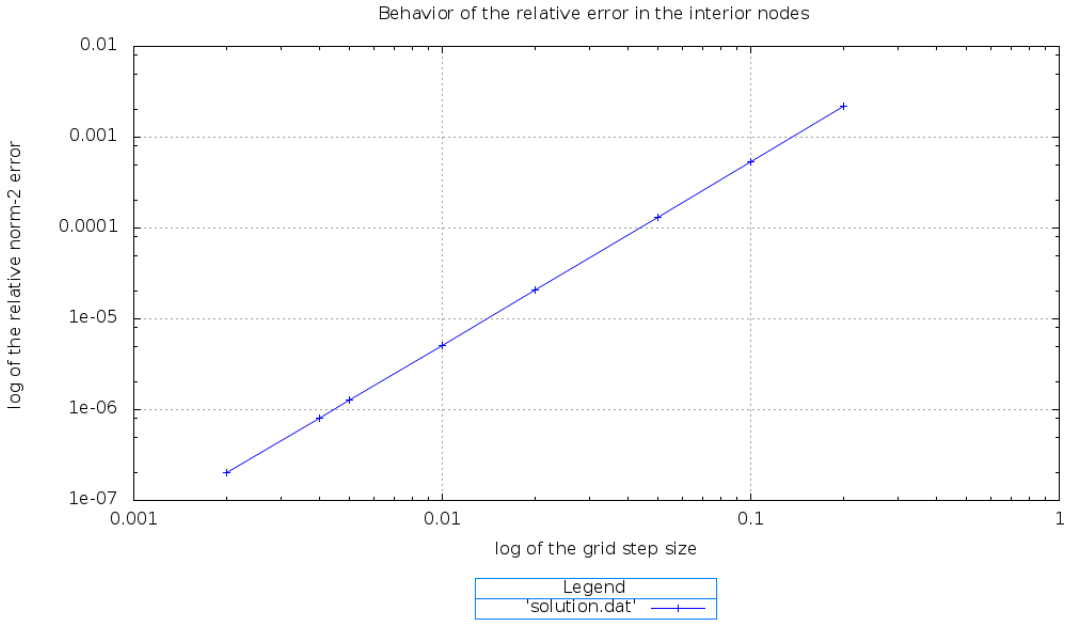



(b) Solution with $N = 102$.

Figure 12: Computed numerical solution for example problem number one, using a one-dimensional uniform staggered grid with only 5 cells (Figure (a)) and 102 cells (Figure (b)), as well as second-order mimetic operators attained by means of the Castillo–Grone Method, as described in §5.1.2. Figure (a) shows how is the Laplacian bound to the centers of the cells in the numerical solution as it can be seen in Figure 3. These plots (as well as the plot shown in Figure 9) were attained by means of MTK's visualization mechanisms.

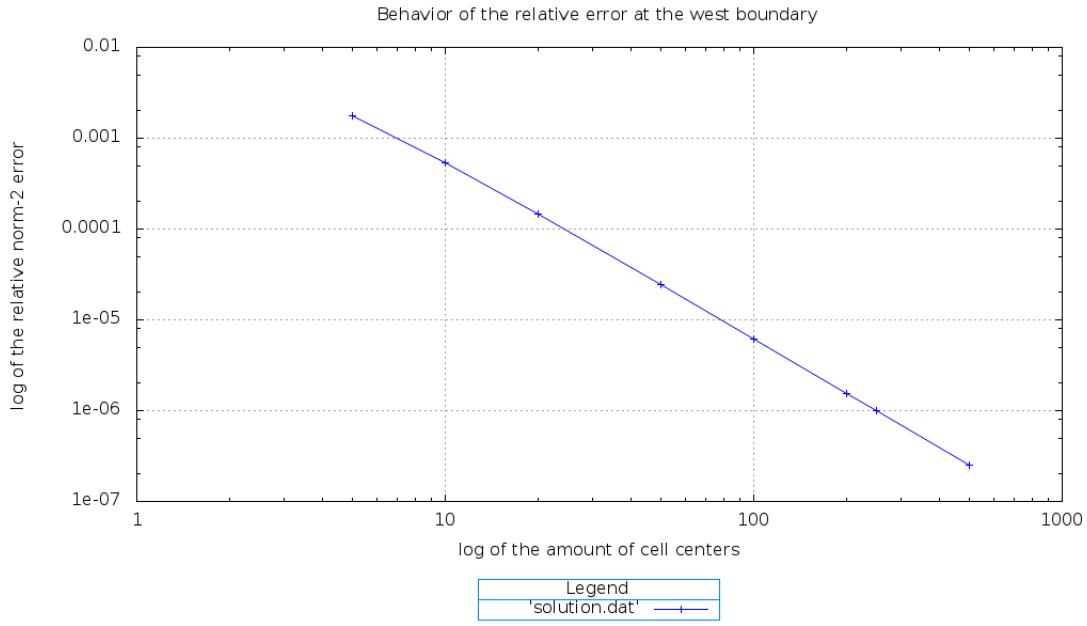



(a) Entire grid, in terms of $N$.

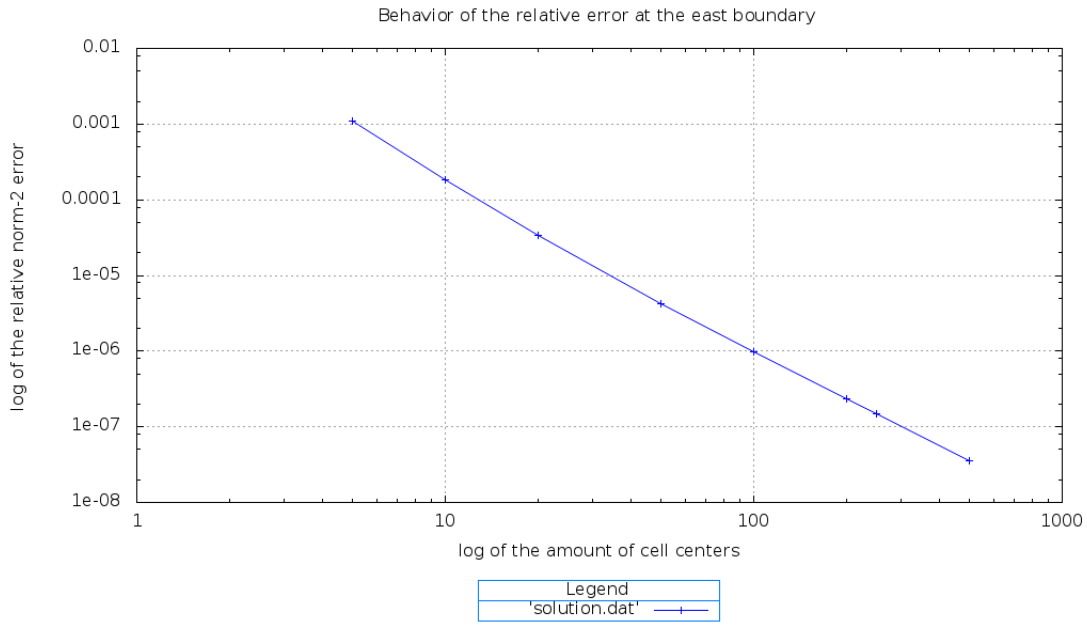



(b) Entire grid, in terms of $\Delta x$.

Figure 13: Grid refinement study all along the entire grid, with emphasis in the interior nodes, depicting the behavior of the error achieved by the MTK-based solution, as explained in §5.1.3. Figure (a) shows the behavior in terms of the number of cells, and Figure (b) shows the behavior in terms of the grid step size.




(a) West boundary




(b) East boundary

Figure 14: Grid refinement study: for example problem number one. Figure (a) shows the behavior of the error at the west boundary node, and Figure (b) does it for the east boundary node. See §5.1.3.

| $N$ | $\Delta x$ | Error interior | Order |
|---|---|---|---|
| 5 | 2.00000e-01 | 2.18379e-03 | - |
| 10 | 1.00000e-01 | 5.40628e-04 | 2.01413e+00 |
| 20 | 5.00000e-02 | 1.32439e-04 | 2.02931e+00 |
| 50 | 2.00000e-02 | 2.07112e-05 | 2.02495e+00 |
| 100 | 1.00000e-02 | 5.12492e-06 | 2.01481e+00 |
| 200 | 5.00000e-03 | 1.27389e-06 | 2.00829e+00 |
| 250 | 4.00000e-03 | 8.14309e-07 | 2.00539e+00 |
| 500 | 2.00000e-03 | 2.03078e-07 | 2.00355e+00 |

Table 6: Calculation of the attained error using MTK objects for the entire grid, as explained in §5.1.3.

| $N$ | $\Delta x$ | Error west | Order west | Error east | Order east |
|---|---|---|---|---|---|
| 5 | 2.00000e-01 | 1.76817e-03 | - | 1.09230e-03 | - |
| 10 | 1.00000e-01 | 5.33654e-04 | 1.72828e+00 | 1.82088e-04 | 2.58466e+00 |
| 20 | 5.00000e-02 | 1.45092e-04 | 1.87894e+00 | 3.38828e-05 | 2.42601e+00 |
| 50 | 2.00000e-02 | 2.43481e-05 | 1.94798e+00 | 4.28961e-06 | 2.25552e+00 |
| 100 | 1.00000e-02 | 6.18200e-06 | 1.97766e+00 | 9.77490e-07 | 2.13369e+00 |
| 200 | 5.00000e-03 | 1.55740e-06 | 1.98894e+00 | 2.32480e-07 | 2.07198e+00 |
| 250 | 4.00000e-03 | 9.98258e-07 | 1.99315e+00 | 1.47263e-07 | 2.04613e+00 |
| 500 | 2.00000e-03 | 2.50327e-07 | 1.99560e+00 | 3.60538e-08 | 2.03017e+00 |

Table 7: Calculation of the attained error using MTK objects for the west and east boundaries, as explained in §5.1.3.

but this time, for $x \in \Omega = [0, 1] \subset \mathbb{R}$, we let [6]:

$$F(x) = \frac{2 \times 10^6 x}{\arctan(100)(1 + 1 \times 10^4 x^2)^2}. \tag{35}$$

We consider generalized Robin boundary conditions of the form:

$$f(0) + f'(0) = -\frac{100}{\arctan(100)}, \tag{36}$$

$$f(1) + f'(1) = 1 + \frac{100}{\arctan(100)(1 + 1 \times 10^4)}, \tag{37}$$

for which, the following analytical solution is known:

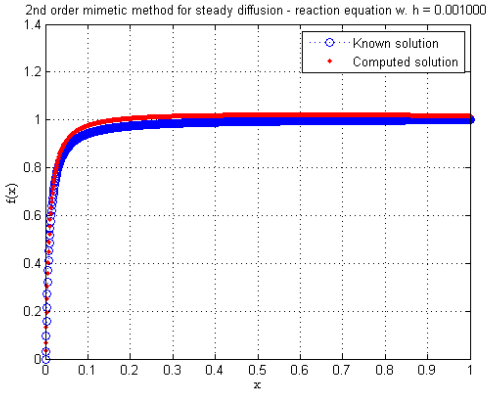
$$f(x) = \frac{\arctan(100x)}{\arctan(100)}. \tag{38}$$

Figure 15 depicts the reference results, which were computed in MATLAB R2010b.

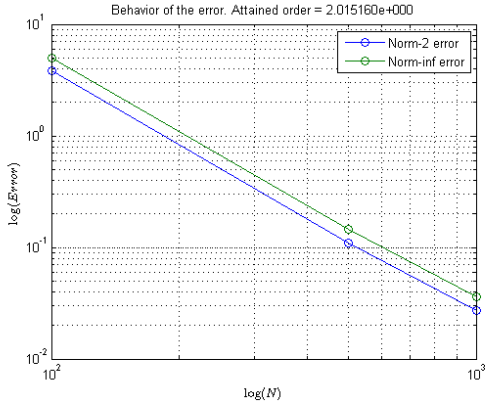
Analogoulsy, Figure 16 contains the solution attained by using the MTK, and Figures 17 and 18 contain the result of the grid refinement study, also using the MTK. Tables 8 and 9 present the related numerical results.

# 6 Concluding remarks and directions of future work

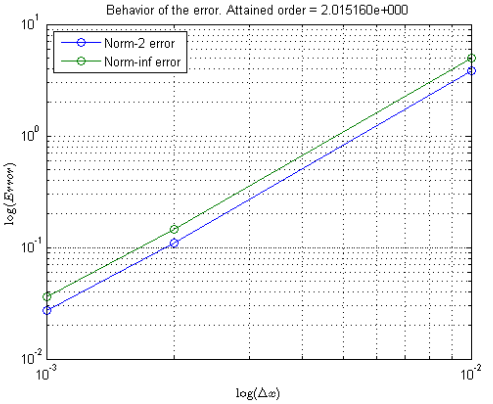
In this work, we introduced the Mimetic Methods Toolkit, or MTK, with the intention of providing an API assisting with the implementation of Mimetic Discretization Methods, when developing computer applications for the simulation and consequent study of any physical phenomenon of interest. We have presented a few important details on Mimetic Discretization Methods; specifically, we introduced the Castillo–Grone Method, which allows for the construction of the Mimetic Differential Operators used to discretize the problems of interest.



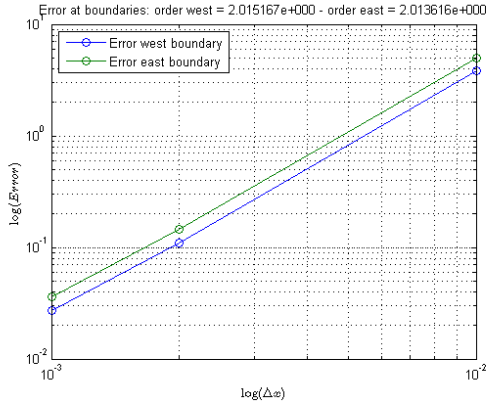

(a) Attained and known solution.




(b) Error in the entire grid, in terms of $N$.




(c) Error in the entire grid, in terms of $\Delta x$.

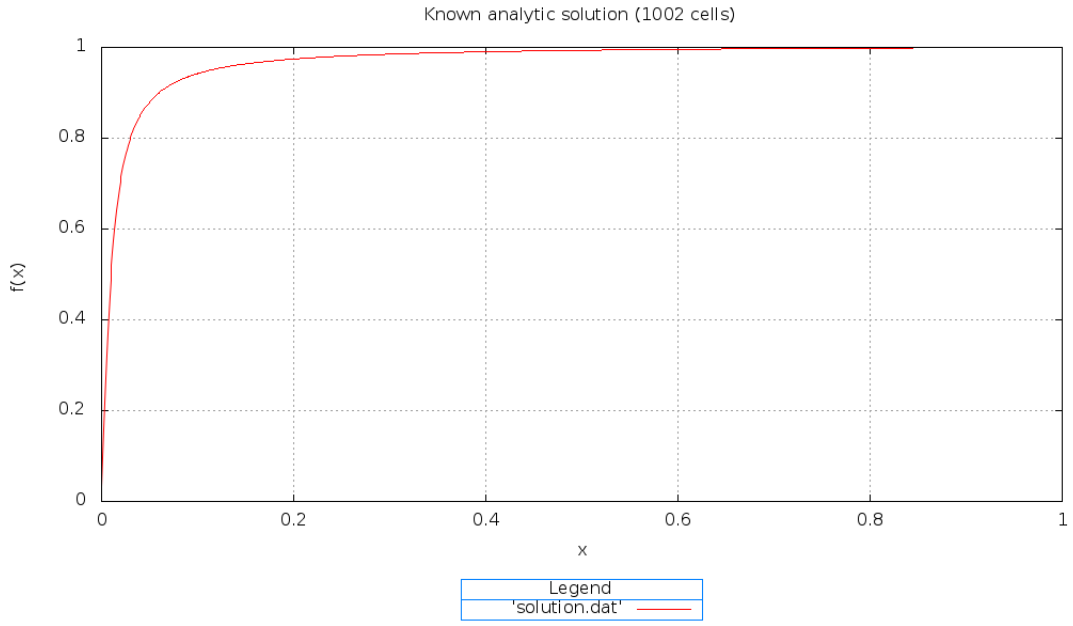



(d) Boundary, in terms of $\Delta x$.

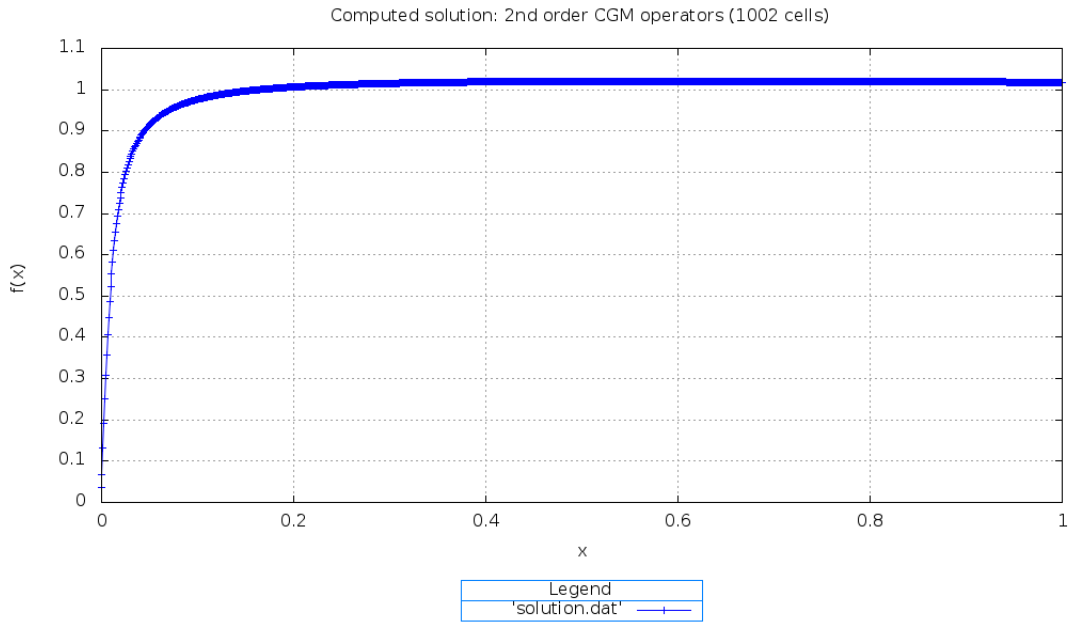
Figure 15: Grid refinement study, depicting the attained order achieved by the CGM-based MDM reference solution for the second example problem (§5.2), computed using MATLAB R2010b. Figure (a) present the attained and the known solution. Figures (b) and (c) present the behavior of the error all along the entire grid, whereas Figure (d) present the behavior of the error at the boundary. As it can be seen, an uniform accuracy prevails at both the interior nodes and the boundary.

| $N$ | $\Delta x$ | Error interior | Order |
|---|---|---|---|
| 100 | 1.00000e-02 | 3.92520e+00 | - |
| 200 | 5.00000e-03 | 7.75892e-01 | 2.33884e+00 |
| 250 | 4.00000e-03 | 4.74483e-01 | 2.20391e+00 |
| 500 | 2.00000e-03 | 1.13039e-01 | 2.06954e+00 |
| 550 | 1.81818e-03 | 9.31962e-02 | 2.02519e+00 |
| 700 | 1.42857e-03 | 5.72903e-02 | 2.01763e+00 |
| 750 | 1.33333e-03 | 4.98634e-02 | 2.01246e+00 |
| 1000 | 1.00000e-03 | 2.79789e-02 | 2.00859e+00 |

Table 8: Calculation of the attained error using MTK objects for the entire grid, as explained in §5.1.3, for example problem number two, presented in §5.2.

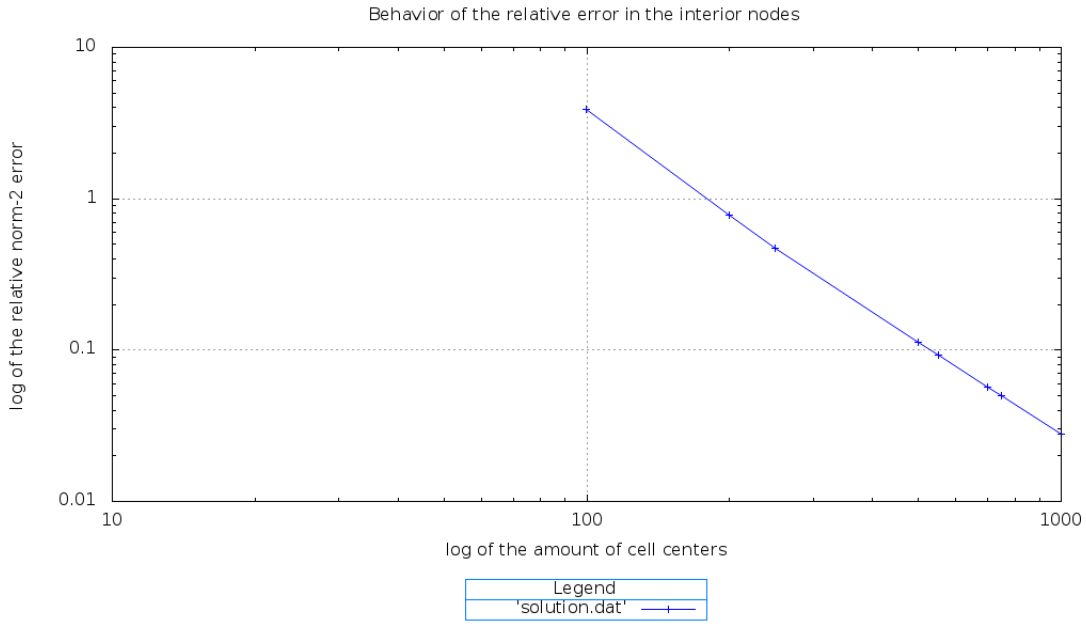



(a) Solution with $N = 1,002$.

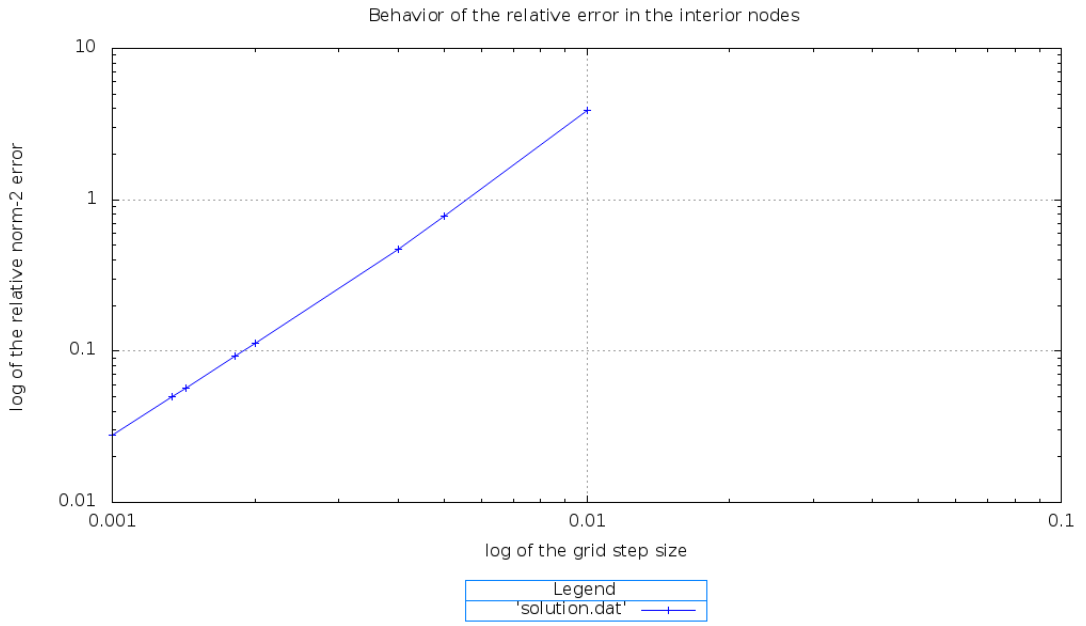



(b) Solution with $N = 1,002$.

Figure 16: Attained solutions for example problem number two, using the MTK. Figure (a) shows a plot of the known solution generated with the MTK, and Figure (b) shows the computed solution. Both plots were generated using 1,002 cells.

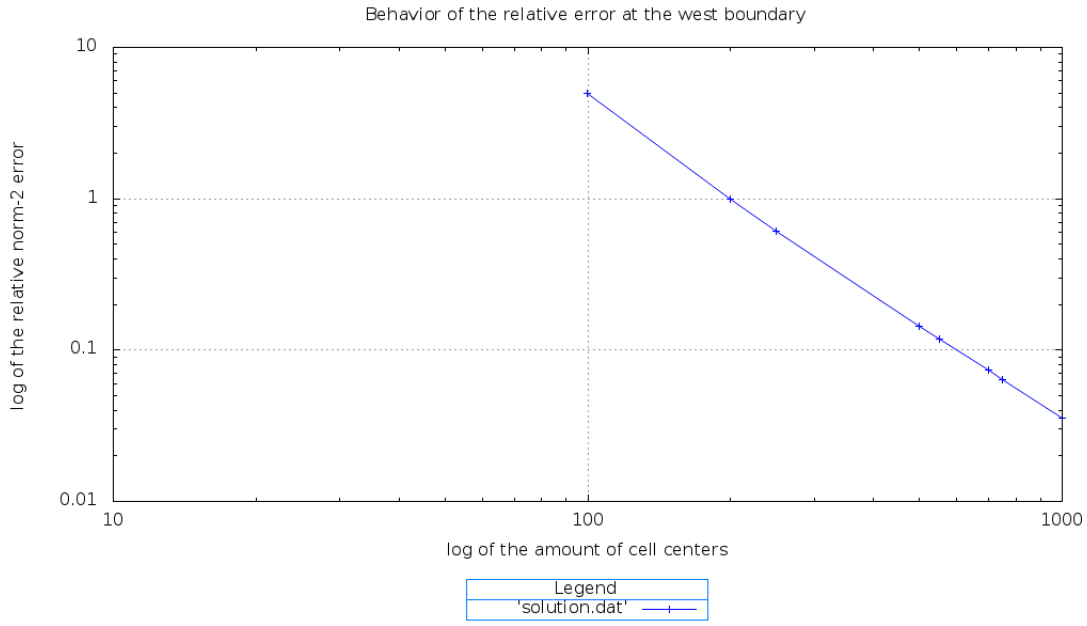



(a) Entire grid, in terms of $N$.

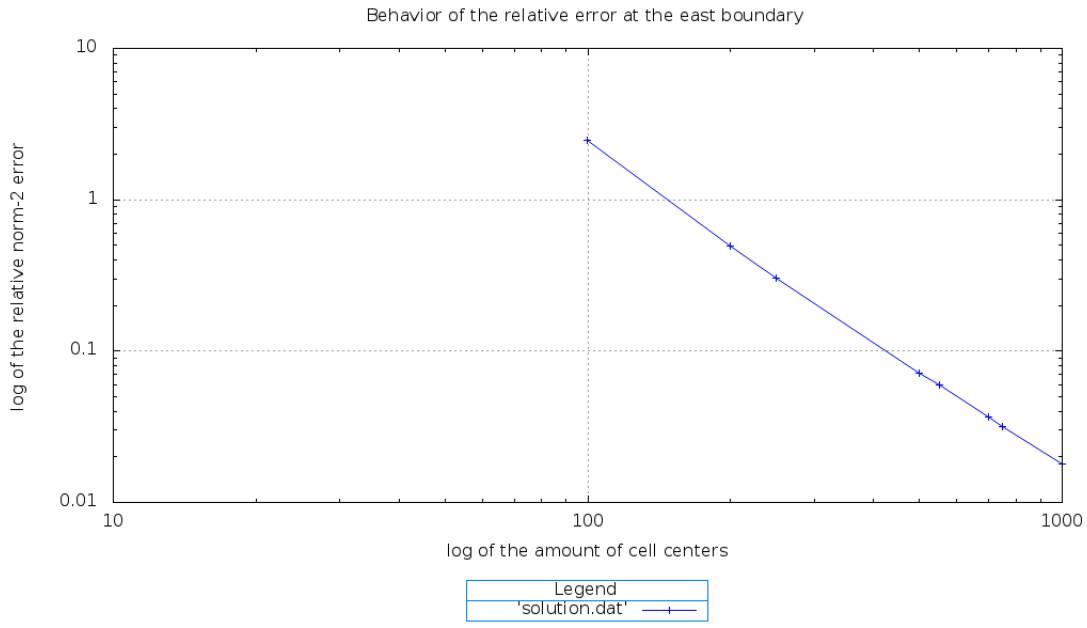



(b) Entire grid, in terms of $\Delta x$.

Figure 17: Grid refinement study for the example problem number two, presented in §5.2. Figure (a) shows the behavior of the error in the entire grid in terms of $N$, and Figure (b) shows it in terms of the step size $\Delta x$.




(a) West boundary




(b) East boundary

Figure 18: Grid refinement study for the example problem number two, presented in §5.2. Figure (a) shows the behavior of the error in the west boundary in terms of $N$, and Figure (b) shows it in the east boundary, also in terms of $N$.

| $N$ | $\Delta x$ | Error west | Order west | Error east | Order east |
|---|---|---|---|---|---|
| 100 | 1.00000e-02 | 5.01087e+00 | - | 2.48970e+00 | - |
| 200 | 5.00000e-03 | 9.95683e-01 | 2.33130e+00 | 4.93448e-01 | 2.33500e+00 |
| 250 | 4.00000e-03 | 6.08587e-01 | 2.20615e+00 | 3.01926e-01 | 2.20143e+00 |
| 500 | 2.00000e-03 | 1.44647e-01 | 2.07293e+00 | 7.20095e-02 | 2.06794e+00 |
| 550 | 1.81818e-03 | 1.19224e-01 | 2.02804e+00 | 5.93753e-02 | 2.02414e+00 |
| 700 | 1.42857e-03 | 7.32461e-02 | 2.02013e+00 | 3.65075e-02 | 2.01674e+00 |
| 750 | 1.33333e-03 | 6.37410e-02 | 2.01466e+00 | 3.17764e-02 | 2.01170e+00 |
| 1000 | 1.00000e-03 | 3.57465e-02 | 2.01047e+00 | 1.78334e-02 | 2.00795e+00 |

Table 9: Calculation of the attained error using MTK objects for the west and east boundaries, as explained in §5.1.3, for example problem number two, presented in §5.2.

We discussed how is the MTK conformed in terms of the classes modeling the most important concepts of the theory of MDMs. Specifically, we discussed the defined data structures and their compatibility with external packages. Similarly, we discussed the mechanisms managing the meshes and grids withing the MTK, and those managing the implementation of the CGM-based mimetic operators.

We presented two examples, which allowed us to pragmatically introduce the MTK's usage philosophy. These examples yielded numerical results which were compared with references solutions. We could see that the solutions attained by means of the MTK not only are correct, but also depict the desired behavior in terms of an uniform order of accuracy all along the discrete domain.

In terms of upcoming work for the MTK, we will focus our attention to several issues within the implementation of each of the concerns. Some directions of upcoming work are:

- Determine the level at which we are interested in making the MTK compatible with external tools, such as for example [13] and [12], so that we can make sure that the concerns are minimally complete, as it is desirable for an API [8].
- To complete an appropriate modeling of meshes and grids, we are interested in modeling the entire context in one, two and three dimensions. Furthermore, we are interested in accounting for different elements that arise when considering higher-dimensional scenarios. For example, in §3.1, we mentioned that the centers of the cells, in one dimension, are nothing but projection of three-dimensional cell centers. Considering a three-dimensional scenario, thus implies the need for differentiating between edges, vertices and centers of a three-dimensional cell, since we intent to develop appropriate visualization mechanisms for three-, two- and one-dimensional grids.
- To implement support for higher-order mimetic operators, as well as for their higher-dimensional counterparts.
- We are also interested in developing a more general system for describing problems, i.e., PDEs of any general type, that can be described in terms of differential operators.

## 6.1 Collaborative development and *MTK Flavors*

Another important aspect of our upcoming work is the collaborative development of MTK. Since MTK's numerical core is written in C++, but the toolkit is intended to keep growing, diverse computational needs have to be taken into account; the result being the following "flavors" or APIs related to MTK. We intent to develop the following related APIs:

1. CMTK: C wrappers collection for MTK; intended for sequential computations.
2. FMTK: Fortran 95 wrappers collection for MTK; intended for sequential computations.
3. MMTK: MATLAB wrappers collection for MTK; intended for sequential computations.

4. RMTK: R wrappers collection for MTK; intended for sequential computations.

5. PyMTK: Python wrappers collection for MTK; intended for sequential computations.

6. DistMTK: Parallel extension for MTK for distributed computing, implemented using MPI and OpenMP.

7. CuMTK: CUDA compatible extension for MTK, implemented using MPI and CUDA.

# 7 Acknowledgments

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 **Dany De Cecchis**, **Mohammad Abouali**, and the following students from "Problems in Computational Science", a research seminar offered in San Diego State University in Fall 2012: **Raul Vargas–Navarro** and **Julia Rossi**. Similarly the authors would like to extent their deepest appreciation to **Dr. Guillermo Miranda**, who assisted in the early modeling of the MTK. His insights and valuable teachings were definitely a necessary conditions for the MTK and for this article.

"*And may the wrong turn to right, in a celestial light, forgive my sacrimony.*"

—T. Karevik et al, *Silverthorn*, 2012.

# References

[1] J.E. Castillo and G.F. Miranda. *Mimetic Discretization Methods*. CRC Press, 1st edition, 2013. In press.

[2] J.E. Castillo, J.M. Hyman, M.J. Shashkov, and S. Steinberg. The Sensitivity and Accuracy of Fourth Order Finite Difference Schemes on Nonuniform Grids in One Dimension. *Computers Math. Applic.*, 30(8):41–55, 1995.

[3] J.E. Castillo and R.D. Grone. A matrix analysis approach to higher-order approximations for divergence and gradients satisfying a global conservation law. *Siam J. Matrix Anal. Appl.*, 25:128–142, 2003.

[4] J.E. Castillo and M. Yasuda. Linear systems arising from second-order mimetic divergence and gradient discretizations. *Journal of Mathematical Modeling and Algorithms*, 4:67–82, 2005.

[5] O. Montilla, C. Cadenas, and J. Castillo. Matrix approach to mimetic discretizations for differential operators on non-uniform grids. *Mathematics and Computer in Simulation*, pages 1–12, 2006.

[6] E.D. Batista and J.E. Castillo. Mimetic schemes on non-uniform structured meshes. *Electronic Transactions on Numerical Analysis*, 34:152–162, 2009.

[7] F.F. Hernández, J.E. Castillo, and G.A. Larrazábal. Large sparse linear systems arising from mimetic discretization. *Computer and Mathematics with Applications*, 53:1–11, 2007.

[8] M. Reddy. *API Design in C++*. Morgan Kauffmann, Massachusetts, 1st edition, 2011.

[9] C.L. Lawson, R.J. Hanson, D. Kincaid, and F.T. Krogh. Basic Linear Algebra Subprograms for FORTRAN usage. *ACM Trans. Math. Soft*, 5:308–323, 1979.

[10] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 3rd edition, 1999.

[11] J.W. Demmel, S.C. Eisenstat, J.R. Gilbert, X.S. Li, and J.W.H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Analysis and Applications.*, 20(3):720–755, 1999.

[12] X.S. Li and J.W. Demmel. SuperLU_DIST: A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems. *ACM Trans. Mathematical Software.*, 29(2):110–140, 2003.

[13] R.C. Whaley, A. Petitet, and J.J. Dongarra. Automated Empirical Optimization of Software and the ATLAS Project. *Parallel Computing*, 27(1–2):3–35, 2001. Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 (`www.netlib.org/lapack/lawns/lawn147.ps`).

[14] M. Naumov. Incomplete LU and Cholesky Preconditioned Iterative Methods Using CUSPARSE and CUBLAS. Technical report, NVIDIA, Santa Clara, California, USA, 2011.

[15] J.A. Chard and V. Shapiro. A multivector data structure for differential forms and equations. *Math. Comput. Simulat.*, 54(1-3):33–64, 2000.

[16] P.W. Gross and P.R. Kotiuga. Data structures for geometric and topological aspects of finite element algorithms. pages 151–169, 2001.

[17] P.W. Gross and P.R. Kotiuga. Finite Element-Based Algorithms to Make Cuts for Magnetic Scalar Potentials: Topological Constraints and Computational Complexity. pages 207–245. 2001.

[18] P. Castillo, R. Rieben, and D. White. FEMSTER: An object-oriented class library of high-order discrete differential forms. *ACM T. Math. Software*, 31(4):425–457, 2005.

[19] Silicon Graphics International. *Introduction to the Standard Template Library.* International, Silicon Graphics, 2012.

[20] B. Karlsson. *Beyond the C++ Standard Library: An Introduction to Boost.* Addison-Wesley, 1st edition, 2005.

[21] J. Blanchette and M. Summerfield. *C++ GUI Programming with Qt 4.* Prentice Hall, 2nd edition, 2008.

[22] W.D. Henshaw. A Primer for Writing PDE Solvers with Overture. Technical report, Lawrence Livermore National Laboratory, Livermore, California, USA, 2011.

[23] O. Rojas, S. Day, J. Castillo, and L.A. Dalguer. Modelling or fupture propagation using high-order mimetic finite differences. *Geophys. J. Int.*, (172):631–650, 2008.

[24] O. Rojas, S. Day, J. Castillo, and L.A. Dalguer. Finite difference modeling of rupture propagation with strong velocity-veakening friction. *Geophys. J. Int.*, 2009.

[25] Eduardo Sanchez, Carlos F. Torres, Pablo Guillen, and German Larrazabal. Modeling and simulation of the production process of electrical energy in a geothermal power plant. *Mathematical and Computer Modelling*, (0):–, 2011.

[26] M. Shashkov. *Conservative Finite Difference Methods on General Grids.* CRC Press, 1st edition, 1996.

[27] E.J. Sanchez et al. *Mimetic Methods Toolkit (MTK) online website.* http://www.csrc.sdsu.edu/mtk/, 2012.

[28] P.R. Amestoy, I.S. Duff, J.Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[29] P. Knupp and S. Steinberg. *Fundamentals of Grid Generation.* CRC Press, 1993.

[30] T. Williams and C. Kelley. *gnuplot 4.4: An Interactive Plotting Program*, 2011.

[31] R. Burden and J.D. Faires. *Numerical Analysis*. Thomson, Belmont, California, 8th edition, 2005.