Tutorial for MOLE

based on commit `e9ad02e`

Carlos Aznarán[†]   Adelaida Otazu[‡]

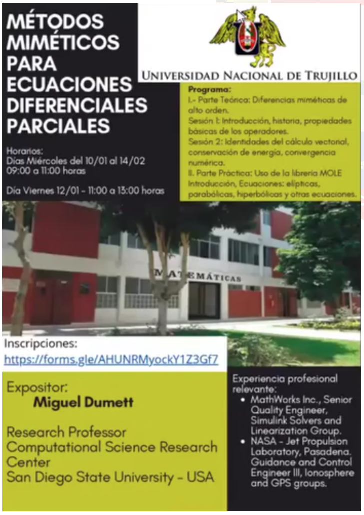
November 2, 2025

[†]Universidad Nacional de Ingeniería
Av. Tupac Amaru, s/n, Lima, Perú
`caznaranl@uni.pe`
0000-0001-8314-2271
[‡]Universidad Nacional del Altiplano Puno
Avenida El Sol, Puno, Perú
`aotazu@unap.edu.pe`
0000-0003-4793-0400

Preface

This manual is for newcomers to Mimetic Operator's Library Enhanced [9] (MOLE) with a foundation in Numerical Analysis for Partial Differential Equations (PDEs). The library is growing with examples implemented with sparse matrix data structures in GNU Octave/ MATLAB, and C++. In addition, most of these examples are provided using a Python module called `pyMOLE` [3]. Therefore, understanding this implementation of mimetic operators will allow users to translate the algorithms into other high-performance scientific programming languages such as Julia, Fortran, Rust and C.

The main goal of this document is to provide clear and complete explanations of these examples to help users understand the core concepts and applications of the library. We would like to thank Professor Miguel Dumett of the Computational Science Research Center at San Diego State University and the National University of Trujillo for organizing MOLE courses.



MÉTODOS MIMÉTICOS PARA ECUACIONES DIFERENCIALES PARCIALES

UNIVERSIDAD NACIONAL DE TRUJILLO

Horarios:
Días Miércoles del 10/01 al 14/02
09:00 a 11:00 horas

Día Viernes 12/01 - 11:00 a 13:00 horas

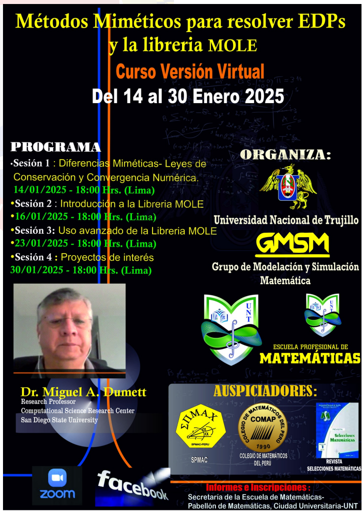
Programa:
I.- Parte Teórica: Diferencias miméticas de alto orden.
Sesión 1: Introducción, historia, propiedades básicas de los operadores.
Sesión 2: Identidades del cálculo vectorial, conservación de energía, convergencia numérica.
II. Parte Práctica: Uso de la librería MOLE
Introducción, Ecuaciones: elípticas, parabólicas, hiperbólicas y otras ecuaciones.

Inscripciones:
https://forms.gle/AHUNRMyockY1Z3Gf7

Expositor:
Miguel Dumett

Research Professor
Computational Science Research Center
San Diego State University - USA

Experiencia profesional relevante:
- MathWorks Inc., Senior Quality Engineer, Simulink Solvers and Linearization Group.
- NASA - Jet Propulsion Laboratory, Pasadena. Guidance and Control Engineer III, Ionosphere and GPS groups.



Métodos Miméticos para resolver EDPs y la librería MOLE

Curso Versión Virtual
Del 14 al 30 Enero 2025

PROGRAMA
- Sesión 1 : Diferencias Miméticas- Leyes de Conservación y Convergencia Numérica.
14/01/2025 - 18:00 Hrs. (Lima)
- Sesión 2 : Introducción a la Libreria MOLE
- 16/01/2025 - 18:00 Hrs. (Lima)
- Sesión 3: Uso avanzado de la Libreria MOLE
- 23/01/2025 - 18:00 Hrs. (Lima)
- Sesión 4 : Proyectos de interés
30/01/2025 - 18:00 Hrs. (Lima)

ORGANIZA:
Universidad Nacional de Trujillo
GMSM
Grupo de Modelación y Simulación Matemática
ESCUELA PROFESIONAL DE MATEMÁTICAS

Dr. Miguel A. Dumett
Research Professor
Computational Science Research Center
San Diego State University

AUSPICIADORES:
SPMAC
COLEGIO DE MATEMÁTICOS DEL PERÚ
REVISTA SELECCIONES MATEMÁTICAS

zoom
facebook

Informes e Inscripciones :
Secretaría de la Escuela de Matemáticas-
Pabellón de Matemáticas, Ciudad Universitaria-UNT

Mimetic Methods courses in January and February 2024 and 2025.

Lima, Puno	Carlos Aznarán
April 2025	Adelaida Otazu

Contents

Octave/MATLAB Scripts

C++ Scripts

Python Scripts

Part I.

Foundations of Mimetic Difference Operators

1. Mathematical and Numerical Foundations

1.1. Introduction and Objectives

[21].

1.2. Notation and Prerequisites

Δx	mesh step size in the direction x
Δy	mesh step size in the direction y
Δz	mesh step size in the direction z
$\mathbf{D}$	mimetic divergence operator
$\mathbf{G}$	mimetic gradient operator
$\mathbf{L}$	mimetic Laplacian operator
$\mathbf{C}$	mimetic rotational operator
$\mathbf{B}$	mimetic boundary operator

1.3. Theoretical Foundations

1.4. Historical Development of Mimetic Methods

MOLE is an open-source library that implements high-order mimetic operators [9]. Let $\Omega = [a, b]$.

$$\mathbf{G} f_d = \vec{0}.$$
$$\mathbf{D} \vec{v}_d = 0.$$
$$\mathbf{C}\mathbf{G} f = 0.$$
$$\mathbf{D}\mathbf{C} \vec{v} = 0.$$
$$\mathbf{D}\mathbf{G} f_d = \mathbf{L} f_d.$$
$$\int_\Omega f \mathbf{D} \vec{v} \, \mathrm{d}V + \int_\Omega \vec{v} \cdot (\mathbf{G} f) \; \mathrm{d}V = \int_{\partial\Omega} f \vec{v} \cdot \vec{n} \, \mathrm{d}S.$$
$$\left\langle \mathbf{D}\vec{v}, f \right\rangle_Q + \left\langle \mathbf{G} f, \vec{v} \right\rangle_P = \left\langle \mathbf{B}\vec{v}, f \right\rangle.$$
$$\left\langle Q\mathbf{D}\vec{v}, f \right\rangle + \left\langle P\mathbf{G} f, \vec{v} \right\rangle = \left\langle \mathbf{B}\vec{v}, f \right\rangle.$$
$$\left\langle Q\mathbf{D}\vec{v} + \mathbf{G}^T P\vec{v}, f \right\rangle = \left\langle \mathbf{B}\vec{v}, f \right\rangle.$$
$$Q\mathbf{D}\vec{v} + \mathbf{G}^T P\vec{v} = \mathbf{B}\vec{v}.$$
$$Q\mathbf{D} + \mathbf{G}^T P = \mathbf{B}.$$

$$\int_0^1 \frac{\mathrm{d}v}{\mathrm{d}x} f \, \mathrm{d}x + \int_0^1 \frac{\mathrm{d}f}{\mathrm{d}x} \, \mathrm{d}x = v(1) \, f(1) - v(0) \, f(0).$$

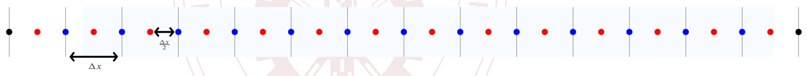


Figure 1.1.: 1D Staggered grid.

$$D^{(2)} = \frac{1}{h} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1c} \\ \vdots & \ddots & \vdots \\ a_{r1} & \cdots & a_{rc} \end{bmatrix}, B = \begin{bmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{p1} & \cdots & a_{pq} \end{bmatrix} A \otimes B := \begin{bmatrix} a_{11}B & \cdots & a_{1c}B \\ \vdots & \ddots & \vdots \\ a_{r1}B & \cdots & a_{rc}B \end{bmatrix}.$$

$$D_{xy}^{(k)} = \begin{bmatrix} \hat{I}_n \otimes D_x^{(k)} & D_y^{(k)} \otimes \hat{I}_m \end{bmatrix}.$$

$$D_{xyz}^{(k)} = \begin{bmatrix} \hat{I}_o \otimes \hat{I}_n \otimes D_x^{(k)} & \hat{I}_o \otimes D_y^{(k)} \otimes \hat{I}_m & D_z^{(k)} \otimes \hat{I}_n \otimes \hat{I}_m & \end{bmatrix}.$$

$$G_{xy}^{(k)} = \begin{bmatrix} \hat{I}_n^T \otimes G_x^{(k)} \\ G_y^{(k)} \otimes \hat{I}_m^T \end{bmatrix}.$$

$$G_{xyz}^{(k)} = \begin{bmatrix} \hat{I}_o^T \otimes \hat{I}_n^T \otimes G_x^{(k)} \\ \hat{I}_o^T \otimes G_y^{(k)} \otimes \hat{I}_m^T \\ G_z^{(k)} \otimes \hat{I}_n^T \otimes \hat{I}_m^T \end{bmatrix}.$$

$$D = DR.$$
$$G = LG$$

Let's consider an uniform mesh

$$x_0 < x_1 < \cdots < x_{n-1} < x_n$$

of $\Omega = [a, b]$.
$i = 0, \ldots, N$.
$x_i = i + i\Delta x$
$\Delta x = \frac{b-a}{n}$
$x_{i+\frac{1}{2}} = \frac{x_i + x_{i+1}}{2}$
$u = \left(u_0, u_{\frac{1}{2}}, u_{\frac{3}{2}}, \ldots, u_{n-\frac{1}{2}}, u_n\right)$. $v = (v_0, v_1, \ldots, v_{n-1}, v_n)$.
$u_i = \frac{u_{i+\frac{1}{2}} - u_{i-\frac{1}{2}}}{h_i}$, $1 \leq i \leq N-1$

$$G\vec{u} = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} = \frac{1}{h}\begin{bmatrix} -\frac{8}{3} & 3 & -\frac{1}{3} & 0 \cdots & 0 \\ 0 & -1 & 1 & 0 & 0 \end{bmatrix}$$

2. Core Mimetic Operators

2.1. Divergence operator

2.1.1. 1D Formulation

Orden 2

Orden 4

Orden 6

Orden 8

2.1.2. 2D Formulation

2.1.3. 3D Formulation

2.2. Gradient Operator

2.2.1. 1D Formulation

Orden 2

Orden 4

Orden 6

Orden 8

2.2.2. 2D Formulation

2.2.3. 3D Formulation

2.3. Laplacian Operator

2.3.1. 1D Formulation

2.3.2. 2D Formulation

2.3.3. 3D Formulation

2.4. Interpolation Operators

2.5. Boundary Condition Implementation

2.5.1. 1D Boundary Handling

2.5.2. 2D/3D Boundary Handling

Let be $f, g\colon \Omega \subset \mathbb{R}^n \to \mathbb{R}$ are scalar fields. Let be $\vec{v}, \vec{w}\colon \Omega \subset \mathbb{R}^n \to \mathbb{R}^m$ are vector fields.

$$\langle f, g \rangle = \int_\Omega f g \,\mathrm{d}V.$$
$$\langle \vec{v}, \vec{w} \rangle = \int_\Omega \vec{v}\vec{w} \,\mathrm{d}V.$$

$$\langle \mathbf{D}\vec{v}, f \rangle + \langle \vec{v}, \mathbf{G} f \rangle = \int_{\partial\Omega} f\vec{v} \cdot \vec{n} \,\mathrm{d}S.$$

$$\begin{aligned} \mathbf{G} \colon \mathbb{R}^{n+2} &\longrightarrow \mathbb{R}^{n+1} \\ f &\longmapsto \mathbf{G} f. \end{aligned}$$

$$\begin{aligned} \mathbf{D} \colon \mathbb{R}^{n+1} &\longrightarrow \mathbb{R}^{n} \\ \vec{v} &\longmapsto \mathbf{D}\vec{v}. \end{aligned}$$

$$\begin{aligned} \mathbf{B} \colon \mathbb{R}^{n+2} &\longrightarrow \mathbb{R}^{n+1} \\ \vec{v} &\longmapsto \mathbf{B}\vec{v}. \end{aligned}$$

Teorema 1

Let be $f = \begin{bmatrix} f_0 & f_{\frac{1}{2}} & f_{\frac{3}{2}} & \cdots & f_{n-\frac{1}{2}} & f_n \end{bmatrix}^T \in \mathbb{R}^{n+2}$ a discretized function defined at the cell centers and at the boundary of the 1D mesh. Let be $v = \begin{bmatrix} v_0 & v_1 & \cdots & v_n \end{bmatrix}^T \in \mathbb{R}^{n+1}$ a discretized function defined on the nodes of the 1D mesh.

- $\mathbf{G} f = 0 \iff f = c.$
- $\mathbf{D}\vec{v} = 0 \iff \vec{v} = c.$

Proof.

- $$\mathbf{D}\vec{v} = \frac{v_{i+1} - v_i}{\Delta x}.$$
  $$0 = \frac{v_{i+1} - v_i}{\Delta x}.$$
- $$\mathbf{G} f = \frac{f_{i+\frac{1}{2}} - f_{i-\frac{1}{2}}}{\Delta x}.$$

■

$$\mathbf{G}_x u = \mathbf{G}u\left(x_i, y_{j+\frac{1}{2}}\right) = \mathbf{G}_{i,j+\frac{1}{2}}.$$

$$\mathbf{G}_y u = \mathbf{G}u\left(x_{i+\frac{1}{2}}, y_j\right) = \mathbf{G}_{i+\frac{1}{2},j}.$$

$$\mathbf{D}_{\vec{v}}\left(x_{i+\frac{1}{2}}, y_{j+\frac{1}{2}}\right) = \mathbf{D}\vec{v}_{i+\frac{1}{2},j+\frac{1}{2}}.$$

3. Code Core Mimetic Operators

3.1. Divergence operator

3.1.1. 1D Formulation

Orden 2

Orden 4

Orden 6

Orden 8

The one-dimensional mimetic divergence operator $div(k, m, k)$, where k: order of accuracy, m: number of cells and dx: step size. The operator div is a matrix of order $m + 2$ by $m + 1$. Where k can take values $2, 4, 6$ and 8; $m > 2 * k + 1$, that is:

if $k = 2$, then the minimum value it takes $m = 5$
if $k = 4$, then the minimum value it takes $m = 9$
if $k = 6$, then the minimum value it takes $m = 13$
if $k = 8$, then the minimum value it takes $m = 17$

Octave script 3.1.: Program `div.m`

Example: If $K = 2$, $m = 5$ and $dx = 1$, then the divergence matrix is of order 7 by 6.

$$D = (\frac{1}{dx}) \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.1}$$

C++ script 3.1.: Program `divergence.h`

Line 1 and 55: In this part the code defines the function $D = div(k, m, dx)$
Line 15: The assert command is the function used to verify the validity

of an expression. In this case, $k \geq 2$. **Line 16:** Verify the validity of an expression. In this case, let k be an even number.
Line 17: Verify that $m \geq 2 * k + 1$.
Line 19: The command $D = sparse(m + 2, m + 1)$ creates a sparse matrix of order $m + 2$ by $m + 1$.
Line 21 and 53: The switch command is used to perform a multiple comparison, k in this case analyzing for the different values that k can take, the maximum being 8.
Line 22: The case when k=2
Line 23 to 24: The main diagonal takes values of -1 from the second column onwards, and below the main diagonal it takes values of 1 from the first column to the last column; the other components are zeros.
Line 27: The case when k=4

Line 28: Define a vector A with 1 row and 5 columns with the indicated fractional values.

Line 29: $D(2, 1 : 5) = A$ refers to row 2 and the first 5 columns of matrix D. Then, vector A is placed at those positions in matrix D.

Line 30: The fliplr(A) command reverses the order of the elements of A, and then multiplies by -1.

Line 32: $D(i, i - 2 : i + 1) = [1/24 - 9/89/8 - 1/24]$ at each iteration of the loop, a 4-valued vector is being assigned to a portion of row i of matrix D.

Line 36: Defines the matrix A of order 2 by 7.

Line 38: In $D(2 : 3, 1 : 7) = A$ assigns matrix A to rows 2 and 3 of matrix D, and columns 1 to 7 of matrix D. That is, the elements of A are placed into that submatrix of D.

Line 39: In $D(m : m + 1, m - 5 : end) = -rot90(A, 2)$ rot90(A, 2) rotates matrix A 180 degrees, which is equivalent to rotating it counterclockwise twice by 90 degrees. It is then multiplied by -1 and assigned to a subsegment of matrix D located in rows m to m+1 and columns from m-5 to the end.
Line 40: In this for loop, index i is traversed from 4 to $m - 1$. At each iteration, a vector of specific values is assigned to a subarray within D. The subarray affects rows i and columns from $i-3$ to $i+2$. The sequence of values assigned is: $[-3/64025/384 - 75/6475/64 - 25/3843/640]$, This is repeated for each value of i between 4 and $m - 1$.
Line 45: The matrix A is defined as $3x93x9$, and its elements are fractions. This matrix will be used to fill specific portions of another matrix D.

Line 48: $D(2 : 4, 1 : 9) = A$ matrix A are assigned to rows 2, 3, and 4, and columns 1 through 9 of matrix D. In other words, rows 2 through 4 of D (3 rows) and the first 9 columns of D are filled with values from matrix A.

Python script 3.1.: Program `div1D.py`

3.1.2. 2D Formulation

Octave script 3.2.: Program `div2D.m`

Line 1 and 29: In this part the code defines the function $D = div2D(k, m, dx, n, dy)$ where k is the order of accuracy, m is the number of points on the x-axis, dx is the step size on the x-axis, n is the number of cells on the y-axis, dy is the step size on the y-axis

Line 16: Calculates the divergence in one dimension on the x-axis $Dx = div(k, m, dx)$ this command is calling divergence in 1D.

Line 17: Calculates the divergence in one dimension on the y-axis $Dy = div(k, n, dy)$ this command is calling divergence in 1D.

Line 19: $Im = sparse(m + 2, m)$ is a sparse matrix of size $m + 2$ by m.
Line 20: $In = sparse(n + 2, n)$ is a sparse matrix of size $n + 2$ by n.
linea 22 Im has size $(m + 2)$ by m as seen on line 19. Then the command $speye(m, m)$ creates a sparse identity matrix of size m by m. and $Im(2 : m + 1, :)$ selects from row 2 to row $m + 1$, all columns. Finally, this puts the identity inside Im but starting at row 2 , leaving row 1 and row $m + 2$ as zeros.

linea 23 In has size $(n + 2)$ by n as seen on line 19. Then the command $speye(n, n)$ creates a sparse identity matrix of size n by n. and $In(2 : n+1, :)$ selects from row 2 to row $n + 1$, all columns. Finally, this puts the identity inside In but starting at row 2 , leaving row 1 and row $n + 2$ as zeros.

linea 25 In this part $Sx = kron(In, Dx)$; This is a sparse $1D$ differential operator, using the Kronecker product using kron. This operator applies Dx in the x direction, repeating for each row in y. Derivatives in x for each row of the $2D$ mesh. Where the size of Sx is: $(n + 2) * m$ by $n * m$, since Dx is m by m and In is $(n + 2)$ by n.

linea 26 In this part $Sy = kron(Dy, Im)$; This is a sparse $1D$ differential operator, using the Kronecker product using kron. This operator applies Dy in the y direction, repeating for each row in y. Derivatives in y for each row of the $2D$ mesh. Where the size of Sy is: $n * (m + 2)$ by $n * m$.

linea 28 D is the horizontal concatenation of the two matrices Sx and Sy, the order of the matrix D being, the number of rows is $(n + 2)$ by n and columns is $2 * n^2$

3.1.3. 3D Formulation

Octave script 3.3.: Program `div3D.m`

Line 1 and 38: In this part the code defines the function $D = div3D(k, m, dx, n, dy, o, dz)$ where k is the order of accuracy, m is the number of points on the x-axis, dx is the step size on the x-axis, n is the number of cells on the y-axis, dy is the step size on the y-axis, o is the number of cells on the z-axis, dz is the step size on the z-axis.

Line 18: $Im = sparse(m + 2, m)$ is equal to the code of div2D line 19.

Line 19: $Im(2 : (m+2)-1, :) = speye(m, m)$ is equal to the code of div2D line 22.

Line 21: $Dx = div(k, m, dx)$ This is calling for divergence in one dimension, on the x-axis.

Line 23: $In = sparse(n + 2, n)$ is equal to the code of div2D line 20.

Line 24: $In(2 : (n + 2) - 1, :) = speye(n, n)$ is equal to the code of div2D line 23.

Line 26: $Dy = div(k, n, dy)$ This is calling for divergence in one dimension, on the y-axis.

Line 28: $Io = sparse(o + 2, o)$ is equal to the code of div2D line 20.

Line 29: $Io(2 : (o + 2) - 1, :) = speye(o, o)$ is equal to the code of div2D line 23.

Line 31: $Dz = div(k, o, dz)$ This is calling for divergence in one dimension, on the z-axis.

Line 33: $Sx = kron(kron(Io, In), Dx)$ applies the derivative at x, keeping y and z fixed using the Kronecker product, the 1D operator is expanded to operate on the entire 3D mesh.

Line 34: $Sy = kron(kron(Io, Dy), Im)$ applies the derivative at y, keeping x and z fixed using the Kronecker product, the 1D operator is expanded to operate on the entire 3D mesh.

Line 35: $Sz = kron(kron(Dz, In), Im)$ applies the derivative at z, keeping x and y fixed using the Kronecker product, the 1D operator is expanded to operate on the entire 3D mesh.

linea 28 D is the horizontal concatenation of the three matrices Sx, Sy and Sz.

3.2. Gradient Operator

3.2.1. 1D Formulation

Orden 2

Orden 4

Orden 6

Orden 8

Octave script 3.4.: Program `grad.m`

C++ script 3.2.: Program `gradient.h`

Python script 3.2.: Program `grad1D.py`

3.2.2. 2D Formulation

Octave script 3.5.: Program `grad2D.m`

3.2.3. 3D Formulation

Octave script 3.6.: Program `grad3D.m`

Octave script 3.7.: Program `lap.m`

C++ script 3.3.: Program `laplacian.h`

Python script 3.3.: Program `lap1D.py`

Octave script 3.8.: Program `lap2D.m`

Octave script 3.9.: Program `lap3D.m`

3.3. Laplacian Operator

3.3.1. 1D Formulation

3.3.2. 2D Formulation

3.3.3. 3D Formulation

3.4. Interpolation Operators

Octave script 3.10.: Program `interpol.m`

Octave script 3.11.: Program `interpol2D.m`

Octave script 3.12.: Program `interpol3D.m`

3.5. Boundary Condition Implementation

3.5.1. 1D Boundary Handling

Octave script 3.13.: Program `addBC1D.m`

3.5.2. 2D/3D Boundary Handling

Let be $f, g \colon \Omega \subset \mathbb{R}^n \to \mathbb{R}$ are scalar fields. Let be $\vec{v}, \vec{w} \colon \Omega \subset \mathbb{R}^n \to \mathbb{R}^m$ are vector fields.

$$\begin{aligned} \langle f, g \rangle &= \int_\Omega f g \,\mathrm{d}V. \\ \langle \vec{v}, \vec{w} \rangle &= \int_\Omega \vec{v}\vec{w} \,\mathrm{d}V. \end{aligned}$$

$$\langle \mathbf{D}\vec{v}, f \rangle + \langle \vec{v}, \mathbf{G} f \rangle = \int_{\partial\Omega} f \vec{v} \cdot \vec{n} \,\mathrm{d}S.$$

$$\begin{aligned} \mathbf{G} \colon \mathbb{R}^{n+2} &\longrightarrow \mathbb{R}^{n+1} \\ f &\longmapsto \mathbf{G} f. \end{aligned}$$

$$\begin{aligned} \mathbf{D} \colon \mathbb{R}^{n+1} &\longrightarrow \mathbb{R}^{n} \\ \vec{v} &\longmapsto \mathbf{D} \vec{v}. \end{aligned}$$

$$\begin{aligned} \mathbf{B} \colon \mathbb{R}^{n+2} &\longrightarrow \mathbb{R}^{n+1} \\ \vec{v} &\longmapsto \mathbf{B} \vec{v}. \end{aligned}$$

Teorema 2

Let be $f = \begin{bmatrix} f_0 & f_{\frac{1}{2}} & f_{\frac{3}{2}} & \cdots & f_{n-\frac{1}{2}} & f_n \end{bmatrix}^T \in \mathbb{R}^{n+2}$ a discretized function defined at the cell centers and at the boundary of the 1D mesh. Let be $v = \begin{bmatrix} v_0 & v_1 & \cdots & v_n \end{bmatrix}^T \in \mathbb{R}^{n+1}$ a discretized function defined on the nodes of the 1D mesh.

- $\mathbf{G} f = 0 \iff f = c.$
- $\mathbf{D} \vec{v} = 0 \iff \vec{v} = c.$

Proof.

-

$$\begin{aligned} \mathbf{D}\vec{v} &= \frac{v_{i+1} - v_i}{\Delta x}. \\ 0 &= \frac{v_{i+1} - v_i}{\Delta x}. \end{aligned}$$

-
$$\mathbf{G}f = \frac{f_{i+\frac{1}{2}} - f_{i-\frac{1}{2}}}{\Delta x}.$$

■

$$\mathbf{G}_x u = \mathbf{G}u\left(x_i, y_{j+\frac{1}{2}}\right) = \mathbf{G}_{i,j+\frac{1}{2}}.$$

$$\mathbf{G}_y u = \mathbf{G}u\left(x_{i+\frac{1}{2}}, y_j\right) = \mathbf{G}_{i+\frac{1}{2},j}.$$

$$\mathbf{D}_{\vec{v}}\left(x_{i+\frac{1}{2}}, y_{j+\frac{1}{2}}\right) = \mathbf{D}\vec{v}_{i+\frac{1}{2},j+\frac{1}{2}}.$$

3.6. Compact operators

C++ script 3.4.: Program `hyperbolic1Dupwind.cpp`

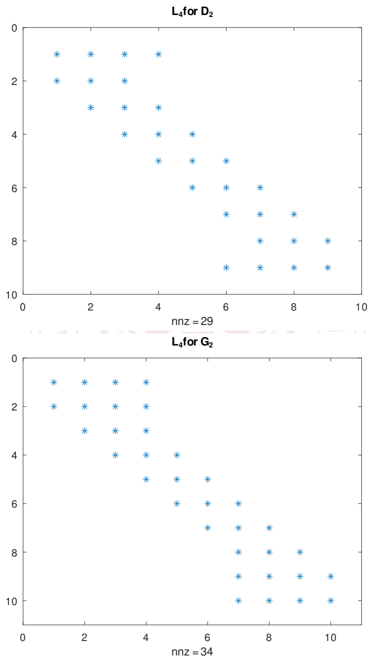


Figure 3.1.: .

4. Numerical Methods for Differential Equations

4.1. Ordinary Differential Equations (ODEs)

4.1.1. Initial Value Problems

Consider the Initial Value Problem

Only the methods used in the tutorials: Verlett, RK4, Explicit Euler

$$\begin{cases} \frac{\mathrm{d}y}{\mathrm{d}t} = f(t, y), & t \in [0, T]. \\ y(0) = y_0. \end{cases}$$

Backward Euler Method

$$f(t_{n+1}, y_{n+1}) = \frac{\mathrm{d}y}{\mathrm{d}t} \approx \frac{y_{n+1} - y_n}{\Delta t} \implies y_{n+1} = y_n + f(t_{n+1}, y_{n+1}) \Delta t.$$

The local truncation error is $\mathcal{O}(\Delta t^2)$. The Butcher table is $\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$.

$$\begin{cases} \frac{\mathrm{d}y}{\mathrm{d}t} = y \sin t^2, & t \in [0, 5]. \\ y(0) = 2. \end{cases}$$

$$S(x) := \int_0^x \sin(t^2)\, \mathrm{d}t = \sum_{n=0}^{\infty} \frac{(-1)^n x^{4n+3}}{(2n+1)!\,(4n+3)}.$$

We integrate and obtain the general solution.

$$\iint \frac{\mathrm{d}^2 u}{\mathrm{d}x^2}\, \mathrm{d}x\, \mathrm{d}x = \iint e^x\, \mathrm{d}x\, \mathrm{d}x.$$
$$\int \frac{\mathrm{d}u}{\mathrm{d}x}\, \mathrm{d}x = \int (e^x + C_1)\, \mathrm{d}x.$$
$$u(x) = e^x + C_1 x + C_2.$$

Now, let's apply the Robin boundary conditions.

$$\begin{cases} 0 = u(0) - \left(\frac{\mathrm{d}u}{\mathrm{d}x}\right)_{x=0} = e^0 + C_1(0) + C_2 - (e^0 + C_1) = C_2 - C_1. \\ 2e = u(1) + \left(\frac{\mathrm{d}u}{\mathrm{d}x}\right)_{x=1} = e^1 + C_1(1) + C_2 + (e^1 + C_1) = 2e + 2C_1 + C_2. \end{cases} \tag{4.1}$$

El sistema (4.1) tiene como solución $C_1 = C_2 = 0$. $\therefore$ la solución de (6.1) es $u(x) = e^x$.

```
#!/usr/bin/env -S octave -qf
% Solves ODE using backward Euler method

h = .05; % Step-size
t = 0:h:5; % Calculates up to y(5)
y = zeros(size(t));
y(1) = 2; % Initial condition
f = @(t, y) sin(t)^2 * y; % f(t, y)

for i = 1:length(t) - 1; % Stages
    old_y = y(i);
    y(i + 1) = fzero(@(y) y - h * f(t(i + 1), y) - old_y, old_y); % Backward Euler
end
```

Octave script 4.1.: Program `backward_euler.m`

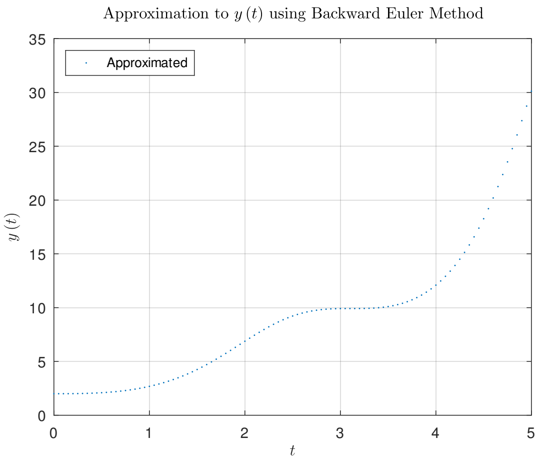


Figure 4.1.: Numerical solution by the Backward Euler Method.

- `https://docs.octave.org/v9.4.0/Ranges.html`
- `https://docs.octave.org/v9.4.0/Solvers.html#XREFfzero`
- `https://docs.octave.org/v9.4.0/Object-Sizes.html#XREFsize`
- `https://docs.octave.org/v9.4.0/Object-Sizes.html#XREFlength`
- `https://docs.octave.org/v9.4.0/Trigonometry.html#XREFsin`
- `https://docs.octave.org/v9.4.0/Special-Utility-Matrices.html#XREFzeros`

Explicit Runge-Kutta 2

$$
\begin{aligned}
\tilde{y}_{n+1} &= y_n + f\left(t_n, y_n\right)\Delta t. \\
y_{n+1} &= y_n + \frac{\Delta t}{2}\left(f\left(t_n, y_n\right) + f\left(\tilde{y}_{n+1}, t_{n+1}\right)\right).
\end{aligned}
$$

The local truncation error is $\mathcal{O}\left(\Delta t^2\right)$. The Butcher table is $\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$.

```
#!/usr/bin/env -S octave -qf
% Solves ODE using explicit RK2 method

h = .05; % Step-size
t = 0:h:5; % Calculates up to y(5)
y = zeros(size(t));
y(1) = 2; % Initial condition
f = @(t, y) sin(t)^2 * y; % f(t, y)

for i = 1:length(t) - 1; % Stages
    k1 = f(t(i), y(i));
    k2 = f(t(i) + h / 2, y(i) + h / 2 * k1);
    y(i + 1) = y(i) + h * k2; % y(i + 1)
end
```

Octave script 4.2.: Program `RK2.m`

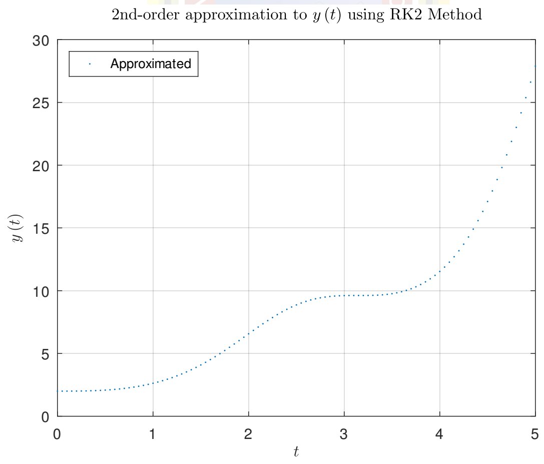


Figure 4.2.: Numerical solution by the Runge-Kutta 2 Method.

Explicit Runge-Kutta 4

$$
\begin{aligned}
k_1 &= f\left(t_n, y_n\right). \\
k_2 &= f\left(t_n + \frac{\Delta t}{2}, y_n + \Delta t \frac{k_1}{2}\right). \\
k_3 &= f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta k_2}{2}\right). \\
k_4 &= f\left(t_n + \Delta t, y_n + \Delta t k_3\right). \\
y_{n+1} &= y_n + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right).
\end{aligned}
$$

The local truncation error is $\mathcal{O}\left(\Delta t^4\right)$. The Butcher table is $\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{4}{6} & \frac{4}{6} & \frac{1}{6} \end{array}$.

```
#!/usr/bin/env -S octave -qf
% Solves ODE using explicit RK4 method

h = .05; % Step-size
t = 0:h:5; % Calculates up to y(5)
y = zeros(size(t));
y(1) = 2; % Initial condition
f = @(t, y) sin(t)^2 * y; % f(t, y)

for i = 1:length(t) - 1% Stages
    k1 = f(t(i), y(i));
    k2 = f(t(i) + h / 2, y(i) + h / 2 * k1);
    k3 = f(t(i) + h / 2, y(i) + h / 2 * k2);
    k4 = f(t(i) + h, y(i) + h * k3);
    y(i + 1) = y(i) + h / 6 * (k1 + 2 * k2 + 2 * k3 + k4); % y(i + 1)
end
```

Octave script 4.3.: Program `RK4.m`

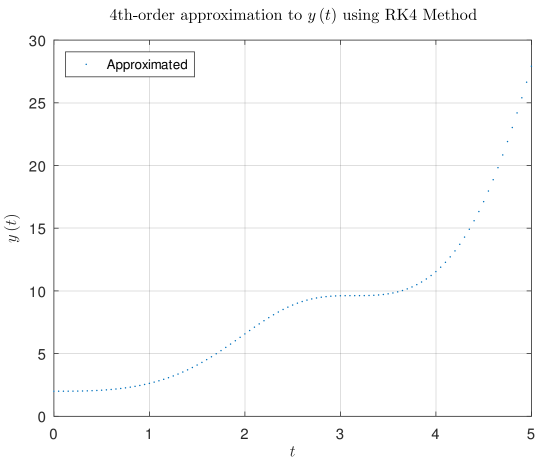


Figure 4.3.: Numerical solution by the Runge-Kutta 4 Method.

Verlett

4.1.2. Boundary Value Problems

4.2. Partial Differential Equations (PDEs)

4.2.1. Hyperbolic Equation (transport)

4.2.2. Parabolic Equation (Diffusion)

4.2.3. Elliptic Equation (Poisson)

von Neumann Stability Analysis

Convergence and Error Estimation

```cpp
//  RK2.cpp

// Description:
// This program solves a first-order ordinary differential equation (ODE) of the
// form:
//     dy/dt = sin^2(t) * y
// using the second-order Runge-Kutta (RK2) method. The solution is computed
// over the time interval [0, 5] with an initial condition y(0) = 2.0.

#include <armadillo>
#include <cmath>   // For sin()
#include <fstream> // For file output
#include <iomanip> // For setprecision
#include <iostream>
#include <sstream> // For string streams

// Function prototype for f(t, y)
double f(double t, double y);

int main()
{
  constexpr double h = 0.1;       // Step size
  constexpr double t_start = 0.0; // Initial time
  constexpr double t_end = 5.0;   // Final time

  int n_steps = static_cast<int>((t_end - t_start) / h) + 1;

  // MOLE's vec type for vectors
  arma::vec t(n_steps); // Time vector
  arma::vec y(n_steps); // Solution vector

  // Initial conditions
  t(0) = t_start;
  y(0) = 2.0;

  // Populate the time vector
  for (int i = 1; i < n_steps; ++i) {
    t(i) = t(i - 1) + h;
  }

  // RK2 Method
  for (int i = 0; i < n_steps - 1; ++i) {
    const double k1 = f(t(i), y(i)); // Slope at the beginning
    const double k2 =
        f(t(i) + h / 2.0, y(i) + h / 2.0 * k1); // Slope at midpoint
    y(i + 1) = y(i) + h * k2;                   // Update solution
  }

  // Set the output stream to fixed-point notation with 6 decimal places
  std::cout << std::fixed << std::setprecision(6);

  // Create a GNUplot script file
  std::ofstream plot_script("plot.gnu");
  if (!plot_script) {
    std::cerr << "Error: Failed to create GNUplot script.\n";
    return 1;
  }
  plot_script << "set title 'RK2 Solution to ODE'\n";
  plot_script << "set xlabel 't'\n";
  plot_script << "set ylabel 'y'\n";
  plot_script << "plot '-' using 1:2 with lines\n";

  // Print the time and solution values to the standard output & gnuplot script
  for (int i = 0; i < n_steps; ++i) {
    // output to stdout
    std::cout << t(i) << " " << y(i) << "\n";
    // AND output to plot_script (plot.gnu)
    plot_script << t(i) << " " << y(i) << "\n";
  }
  plot_script.close();

  // Execute GNUplot using the script
  if (system("gnuplot -persist plot.gnu") != 0) {
    std::cerr << "Error: Failed to execute GNUplot.\n";
    return 1;
  }
  return 0;
}

// Function definition for f(t, y)
double f(double t, double y) { return std::pow(std::sin(t), 2) * y; }
```

C++ script 4.1.: Program `RK2.cpp`

Part II.

Practical Exercises with MOLE

5. Getting Started

MOLE is a high-quality (C++ and GNU Octave) library that implements high-order mimetic methods to solve 1D, 2D and 3D spatial dimensions for PDEs. It provides discrete analogs of the most common vector calculus operators

The official website is `https://csrc-sdsu.github.io/mole`. After skimming the description and reading the papers you will find out that this method never uses a ghost points.

`https://www.csrc.sdsu.edu/research-reports`

5.1. Installation and Setup

5.2. Compiling and running Your First Simulation

6. 1D Case Studies

6.1. Hyperbolic Problems: Transport Equation

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0.$$

```cpp
{

  // Create a sparse matrix of size (m+1) x (m+1)
  arma::sp_mat S(m + 1, m + 1);
  if (type == "backward") {
    // Backward difference
    S.diag(-1) = -arma::ones<vec>(m);   // Sub-diagonal
    S.diag(0) = arma::ones<vec>(m + 1); // Main diagonal
    S(0, m - 1) = -1;                   // Wrap-around for periodic boundary
    S /= dx;
  }
  else if (type == "forward") {
    // Forward difference
    S.diag(0) = -arma::ones<vec>(m + 1); // Main diagonal
    S.diag(1) = arma::ones<vec>(m);      // Super-diagonal
    S(m, 1) = 1;                         // Wrap-around for periodic boundary
    S /= dx;
  }
  else { // "centered"
    // Centered difference
    S.diag(-1) = -arma::ones<vec>(m); // Sub-diagonal
    S.diag(1) = arma::ones<vec>(m);   // Super-diagonal
    S(0, m - 1) = -1;                 // Wrap-around for periodic boundary
    S(m, 1) = 1;                      // Wrap-around for periodic boundary
    S /= (2 * dx);
  }

```

C++ script 6.1.: Program `hyperbolic1Dupwind.cpp`

6.1.1. Animating the Transport Equation

6.2. Elliptic Problems: Poisson Equation

Python script 6.1.: Program `elliptic1D.py`

```
#!/usr/bin/env -S octave -qf
% Solves the 1D Poisson Equation with Robin Boundary Conditions and a
% non-constant forcing right hand side using the Mimetic Method with
% MOLE in Octave / MATLAB.
%
%     Δu = f
%
% u : Vertical Displacement of a membrane
% f : Forcing right hand side
% Δ : Laplace Operator

addpath('/usr/share/mole/matlab/')

west = 0; % Domain's limits
east = 1;

k = 6; % Operator's order of accuracy
m = 2 * k + 1; % Minimum number of cells to attain the desired accuracy
dx = (east - west) / m; % Step length

L = lap(k, m, dx); % 1D Mimetic laplacian operator
L_before_name = sprintf("L_before.h5");
save("-hdf5", L_before_name, "L")
figure('visible', 'off');
spy(L);
saveas(gcf, "elliptic1Dsparsebefore.pdf", 'pdfcrop')

% Impose Robin BC on laplacian operator
a = 1;
b = 1;
L = L + robinBC(k, m, dx, a, b);
L_after_name = sprintf("L_after.h5");
save("-hdf5", L_after_name, "L")
spy(L);
saveas(gcf, "elliptic1Dsparseafter.pdf", 'pdfcrop')

% 1D Staggered grid
grid = [west west + dx / 2:dx:east - dx / 2 east];
save("-hdf5", "grid")

% RHS
U = exp(grid)';
U(1) = 0; % West BC
U(end) = 2 * exp(1); % East BC
tic
U = L \ U; % Solve a linear system of equations
toc
save("-hdf5", "U")
% Plot result
plot(grid, U, 'o')
hold on
plot(grid, exp(grid))
legend('Approximated', 'Analytical', 'Location', 'NorthWest')
```

Octave script 6.1.: Program `elliptic1D.m`

$$\begin{cases} \dfrac{\mathrm{d}^2 u}{\mathrm{d}x^2} = e^x, \text{ para } x \in [0,1]\,. \\ 0 = u\left(0\right) - \left(\dfrac{\mathrm{d}u}{\mathrm{d}x}\right)_{x=0}. \\ 2e = u\left(1\right) + \left(\dfrac{\mathrm{d}u}{\mathrm{d}x}\right)_{x=1}. \end{cases} \tag{6.1}$$

- En la línea 1 encontramos el *shebang*[1], esto permite ejecutar un script de Octave `./elliptic1D.m` con la opción de modo de procesamiento por lotes (batch), para esto se necesita tener permisos de ejecución (por ejemplo, `chmod +x elliptic1D.m`).

- En las líneas 2 al 10 tenemos un comentario sobre el Program de modo que ayude al codificador a obtener un contexto del problema a resolver.

- En la línea 12, la función `addpath` agrega el directorio `"/usr/share/mole/matlab/"` a la ruta de búsqueda de la función. Allí se encuentran el conjunto de scripts Octave / MATLAB de la biblioteca MOLE. Vea el Program **??**.

- En las líneas 14 y 15, se inicializan los identicadores `west` (oeste, izquierda), `east` (este, derecha) con los valores de 0 y 1, respectivamente, estos representan los valores de frontera del dominio espacial en (6.1).

- En la línea 21, llamamos a la función `lap`, este genera un operador Laplaciano discreto extendido que requiere como argumentos obligatorios el orden de precisión `k`, el número de celdas `m` y el tamaño de paso `dx`.

  $$L = L^{(k)} = D^{(k)}G^{(k)} = DG. \qquad (\triangle = \nabla \cdot \nabla)\,,$$

  donde D y G son los operadores miméticos de divergencia y gradiente, respectivamente. Dado que $D \in \mathbb{R}^{(m+2)\times(m+1)}$ y $G \in \mathbb{R}^{(m+1)\times(m+2)}$, entonces $L \in \mathbb{R}^{(m+2)\times(m+2)}$.

- En la línea 22, con la función `figure` desactivamos que se muestre la figura en la pantalla, preferimos solamente guardar la gráfica.

- En la línea 23, con la función `spy` graficamos (no se mostrará) el patrón de dispersidad de L.

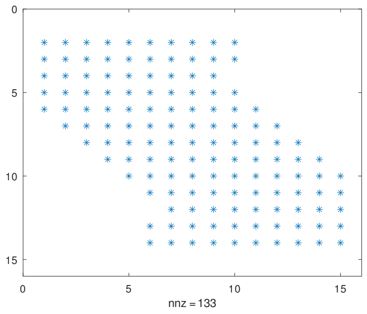
- En la línea 24, con la función `saveas` guardamos esta gráfica en formato PDF y recortado.

- En la línea 29, llamamos a la función `robinBC`, este requiere como argumentos obligatorios el orden de precisión `k`, el número de celdas `m`, el tamaño de paso `dx`, el coeficiente Dirichlet `a` y el coeficiente Neumann `b`. Esta función devuelve una matriz en $\mathbb{R}^{(m+2)\times(m+2)}$. Actualizamos la matriz `L` según el Algoritmo 1.

Algorithm 1: Actualizaciones del operador Laplaciano discreto extendido.

```
1 $A \leftarrow L$;
2 $F \leftarrow f$;
3 $A \leftarrow A + R_G$;
4 $U \leftarrow \text{solve}\,(A, F)$;
```

[1] https://en.wikipedia.org/wiki/Shebang_(Unix)



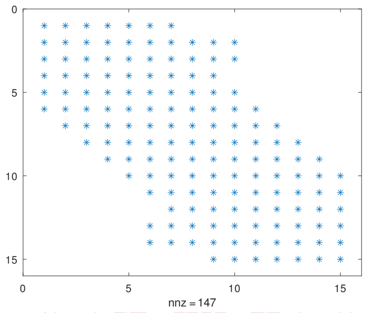


Figure 6.1.: Izquierda: Representación dispersa de L hasta la línea 21. La primera y la última fila son vectores de ceros. Derecha: Representación dispersa de L hasta la línea 29. La matriz $L \in \mathbb{R}^{15\times 15}$.

- En la línea 34, creamos la malla escalonada unidimensional, note que los puntos internos son los centros de las celdas equiespaciados por `dx`. La distancia entre el extremo izquierdo y el posterior punto malla, así como del extremo derecho y el anterior punto malla es `dx/2`.

- En las líneas 35 y 43, guardamos `save` la malla computacional y la solución en el formato HDF5 para posterior post procesamiento.

- En la líneas 39 y 40, aplicamos las condiciones de frontera Robin, empleamos la función `exp`. El signo menos que antecede al coeficiente Neumann `b` se debe a que en el borde izquierdo de la malla el vector normal hacia afuera apunta hacia la izquierda, mientras que en el borde derecho el vector normal hacia afuera apunta hacia la derecha.

- En la línea 42, resolvemos el sistema de ecuaciones lineales disperso con la función `mldivide`.

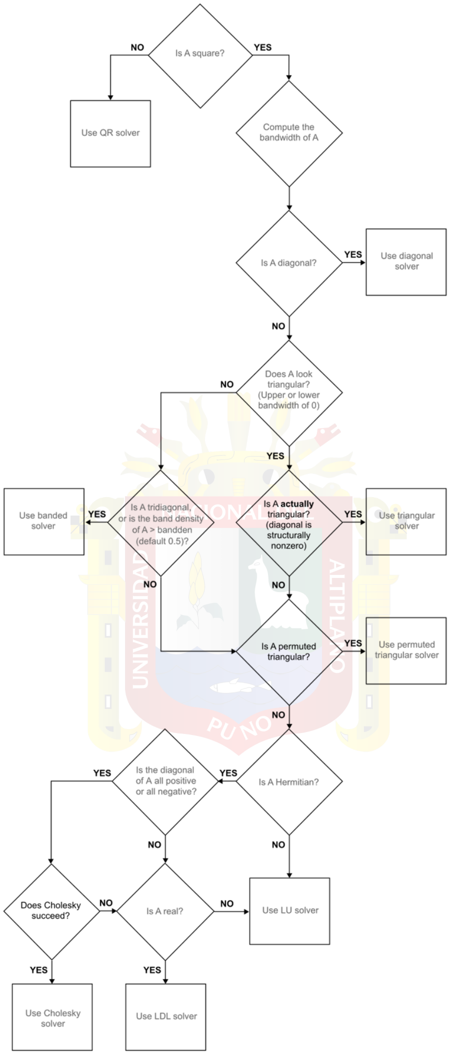

NO
Is A square?
YES
Use QR solver
Compute the bandwidth of A
Is A diagonal?
YES
Use diagonal solver
NO
NO
Does A look triangular? (Upper or lower bandwidth of 0)
YES
Use banded solver
YES
Is A tridiagonal, or is the band density of A > bandden (default 0.5)?
Is A **actually** triangular? (diagonal is structurally nonzero)
YES
Use triangular solver
NO
NO
Is A permuted triangular?
YES
Use permuted triangular solver
NO
YES
Is the diagonal of A all positive or all negative?
YES
Is A Hermitian?
NO
NO
Does Cholesky succeed?
NO
Is A real?
NO
Use LU solver
YES
YES
Use Cholesky solver
Use LDL solver

Figure 6.2.: Diagrama de flujo del solucionador `mldivide` para matrices disperas que emplea MATLAB. Recuperado de `https://www.mathworks.com/help/matlab/ref/double.mldivide.html`.

Resultados del Program 6.1

En primer lugar, mostramos la gráfica a escala 1:1 de la solución exacta y de la solución mimética obtenida en el Program 6.1.

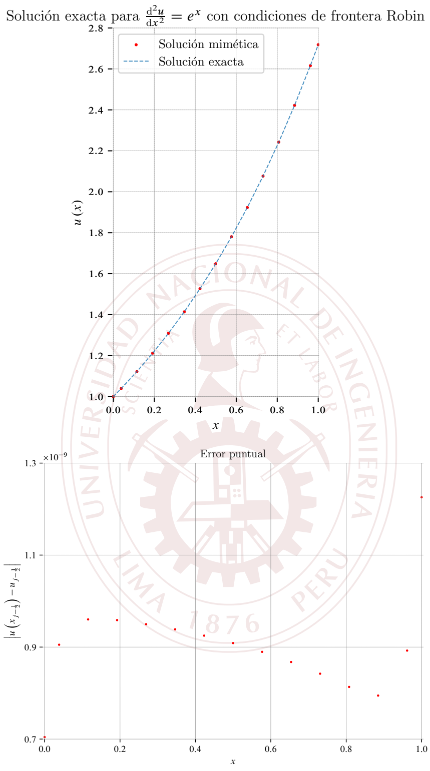


Figure 6.3.: Izquierda: Solución de (6.1) usando `k=6` y `m=2k+1=13`. Derecha: Error en la malla escalonada $\left\{0, \dots, x_{j-\frac{1}{2}} \dots, 1\right\}$

En segundo lugar, mostramos una gráfica del error en cada punto de la malla computacional dada por

$$\text{Error de } u \text{ en } x_{j-\frac{1}{2}} = \left|u\left(x_{j-\frac{1}{2}}\right) - u_{j-\frac{1}{2}}\right|.$$

Por último, mostramos la tabla de los errores y el orden de convergencia numérico.

Δx	Error ℓ_1	Orden
1.562×10^{-2}	4.410×10^{-2}	-
1.535×10^{-2}	4.464×10^{-2}	-0.678

Table 6.1.: Tabla de errores de aproximación de U en $x_{j-\frac{1}{2}}$ y el orden convergencia numérico obtenido.

6.3. Diffusion Equation in 1D

Given the one-dimensional heat equation

$$\begin{cases} \frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}, & (x,t) \in (a,b) \times (0,\infty) . \\ u(x,0) = f(x), & x \in [a,b] . \\ u(a,t) = \alpha(t), & t \in (0,\infty) , \\ u(b,t) = \beta(t), & t \in (0,\infty) , \end{cases} \tag{6.2}$$

where κ is the thermal diffusivity.

The 1D heat equation code by mimetic methods
In the one-dimensional heat equation in the parabolic1D.m code, the PDE being solved is given by:

$$\begin{cases} \frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}, & (x,t) \in (0,1) \times (0,1) . \\ u(x,0) = 0, & x \in [0,1] . \\ u(0,t) = 100., & t \in (0,1) , \\ u(1,t) = 100., & t \in (0,1) , \end{cases} \tag{6.3}$$

Line 9: In this part the code defines the value of the diffusion coefficient $\kappa = 1$.

Line 10: Define the left side $a = 0$ of the domain of the variable x.

Line 11: Define the right hand side $b = 1$ of the domain of the variable x.

Line 13: The Operator's order of accuracy $k = 2$.

Line 14: m is the number of cells, where m can take values $m \geq 2 * k + 1$; in this case $m = 2 * (2) + 1 = 5$.

Line 15: In this line the step size in x is defined, defined as $dx = (b-a)/m$, en this case, replacing it, we have $dx = \frac{1-0}{5} = \frac{1}{5}$

Line 17: End time $t = 1$.

Line 18: Von Neumann stability criterion for $k = 2$, we have $dt = \frac{(dx)^2}{3\kappa}$, in this exercise the step in time is given by: $dt = \frac{1}{75}$

Line 20: $L = lap(k, m, dx)$ 1D mimetic Laplacian operator is of order $m + 2$ so $m + 2$ for this exercise is of order 7 by 7, where $k = 2$, $m = 5$ and $\frac{1}{5}$, then $L = lap(2, 5, \frac{1}{5})$.

Line 23: The initial condition value $u(x, 0) = 0$ is a matrix of order $m+2$ by 1, in this case 7 by 1.

Line 25: The boundary condition is on the left side u(a,t)= u(0,t)=100

Line 26: The boundary condition on the right side u(b,t)= u(1,t)=100

Line 29: The mesh used in the mimetic method grid = $[west\ west+dx/2 : dx : east - dx/2\ east]$, then grid = $[a\ a + \frac{dx}{2} : dx : b - \frac{dx}{2}\ b]$, in this exercise the mesh is given by:

$$\text{grid} = [0\ \tfrac{1}{10} : \tfrac{1}{5} : \tfrac{9}{10}\ 1] = \{0, \tfrac{1}{10}, \tfrac{3}{10}, \tfrac{5}{10}, \tfrac{7}{10}, \tfrac{9}{10}, 1\}$$

Line 31: If explicit=1 then the PDE will be solved in time by the explicit method, and if explicit=0 then the PDE will be solved by the implicit method.

Line 33 to 50: In this section we have the explicita solution of the PDE and the graph.

Line 36: The value of L is obtained by:

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2} \tag{6.4}$$

Discretizing $\frac{\partial u}{\partial t}$ by forward finite differences, and $\frac{\partial^2 u}{\partial x^2}$ by applying mimetic methods

$$\frac{u_i^{n+1} - u_i^n}{dt} = \kappa L u_i^n \tag{6.5}$$

then clearing u_i^{n+1} we obtain:

$$u_i^{n+1} = u_i^n + \kappa dt L u_i^n \tag{6.6}$$

Factoring u_i^n we have:

$$u_i^{n+1} = (I + \kappa dt L) u_i^n \tag{6.7}$$

where I is the identity matrix of general order $m + 2$ by $m + 2$, for this exercise of order 7 by 7, being

$$L = (\kappa dt L + I)$$

being in the code: $L = alpha * dt * L + speye(size(L))$; finally to obtain the solution

$$u_i^{n+1} = L * u_i^n \tag{6.8}$$

in the code on line 47 we have $U = L * U$.

Line 39: The iteration starts for $t = 0$ until $t = t/dt + 1$, in this exercise $t/dt + 1 = 75 + 1 = 76$.

Line 40 and 60: The graph of the mesh on the x axis by grid and the solution of the PDE U.

Line 41 and 62: In this line the graph is on the x axis from 0 to 1 and on the y axis from 0 to 105.

Line 42 and 63: In this line prints the different times $(i * dt)$for explicit method, with two decimal $(.2f)$.

Line 43 and 64: Use the title command to display the title designated on line 42.

Line 44 and 65: Label the x-axis in this exercise as x.
Line 45 and 66: Label the y-axis in this exercise as T.

Line 46 and 67: Shows the graph for the different times with a pause of 0.01.

Line 50 to 71: In this section we have the implicita solution of the PDE and the graph.

Line 53: The value of L is obtained by:

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2} \tag{6.9}$$

Discretizing $\frac{\partial u}{\partial t}$ by forward finite differences, and $\frac{\partial^2 u}{\partial x^2}$ by applying mimetic methods

$$\frac{u_i^{n+1} - u_i^n}{dt} = \kappa L u_i^{n+1} \tag{6.10}$$

then clearing u_i^{n+1} we obtain:

$$u_i^{n+1}(I - \kappa dt L) = u_i^n \tag{6.11}$$

where I is the identity matrix of general order $m+2$ by $m+2$, for this exercise of order 7 by 7, being

$$L = (-\kappa dt L + I)$$

being in the code: $L = -alpha * dt * L + speye(size(L))$.

Line 54: In this line $dL =$ decomputation(L) the matrix L is decomposed into lower triangular matrix L and upper triangular matrix U using LU decomposition method.

Line 68: In this line solve the $u^{n+1} = (dL)^{-1} * u^n$, thus finding the solution $U = dL \setminus U$.

6.3.1. Time Evolution of Diffusion

6.4. Stationary Convection-Diffusion Equation in 1D

```
#!/usr/bin/env -S octave -qf
% Solves the 1D Heat equation with Dirichlet boundary conditions

clc
close all

addpath('/usr/share/mole/matlab/')

alpha = 1; % Thermal diffusivity
west = 0; % Domain's limits
east = 1;

k = 2; % Operator's order of accuracy
m = 2*k+1; % Minimum number of cells to attain the desired accuracy
dx = (east-west)/m;

t = 1; % Simulation time
dt = dx^2/(3*alpha); % von Neumann stability criterion for explicit scheme, if k > 2 then /(4*alpha)

L = lap(k, m, dx); % 1D Mimetic laplacian operator

% IC
U = zeros(m+2, 1);
% BC
U(1) = 100;
U(end) = 100;

% 1D Staggered grid
grid = [west west+dx/2: dx :east-dx/2 east];

explicit = 1; % 0 = Implicit scheme

if explicit
    tic
    % Explicit
    L = alpha*dt*L + speye(size(L));

    % Time integration loop
    for i = 0 : t/dt+1
        plot(grid, U, 'o-')
        axis([0 1 0 105])
        str = sprintf('Explicit \t t = %.2f', i*dt);
        title(str)
        xlabel('x')
        ylabel('T')
        pause(0.01)
        U = L*U; % Apply the operator
    end
    toc
else
    tic
    % Implicit
    L = -alpha*dt*L + speye(size(L));
    dL=decomposition(L)
    %This line precomputes the LU decomposition of L and stores it as a
    %decomposition object. Because it's being stored as a decomposition
    %object. Matlab knows not to bother with LU factorizing L every time we
    %run \, which means that solving the system is sped up.
    %
    for i = 0 : t/dt+1
        plot(grid, U, 'o-')
        axis([0 1 0 105])
        str = sprintf('Implicit \t t = %.2f', i*dt);
        title(str)
        xlabel('x')
        ylabel('T')
        pause(0.01)
        U = dL\U;
    end
    toc
end
```

Octave script 6.2.: Program `parabolic1D.m`

7. 2D and 3D Case Studies

7.1. Extending Operators to Higher Dimensions

7.2. Heat Transfer in 2D

7.3. Fluid Flow in 3D

7.4. Convection-diffusion

Given the three-dimensional convection diffusion equation

$$\begin{cases} \frac{\partial u}{\partial t} + \nabla \cdot (\mathbf{v}u) = \nabla \cdot (D\nabla u) + R, & \mathbf{x} \in \Omega, t > 0 \\ u(\mathbf{x}, 0) = \bar{\alpha}, & \mathbf{x} \in \Omega. \\ u(\mathbf{x}, t) = \bar{\alpha}_0, & t > 0, \mathbf{x} \in \partial\Omega \end{cases} \tag{7.1}$$

where Ω is the three-dimensional domain with boundary $\partial\Omega$.
Line 11: The Operator's order of accuracy $k = 2$.

Line 12: m is the number of cells on the x-axis, where m can take values $m = 101$.

Line 13: m is the number of cells on the y-axis, where m can take values $m = 51$.

Line 14: m is the number of cells on the z-axis, where m can take values $m = 101$.

Line 17 and 18: The domain on the x-axis, $a = 0 \leq x \leq 101 = b$

Line 19 and 20: The domain on the y-axis, $c = 0 \leq y \leq 51 = d$

Line 21 and 22: The domain on the z-axis, $e = 0 \leq x \leq 101 = f$

Line 25: The spatial step sizes on the x-axis, $dx = (b - a)/m$, this case $dx = 1$.

Line 26: The spatial step sizes on the y-axis, $dy = (d - c)/n$, this case $dy = 1$.

Line 27: The spatial step sizes on the z-axis, $dz = (f - e)/o$, this case $dz = 1$.

```
1   #!/usr/bin/env -S octave -qf
2   % convection_diffusion.m | Solves convection-diffusion equation using MOLE
3
4   clc
5   close all
6   format short
7
8   addpath('/usr/share/mole/matlab/')
9
10  % Mimetic operator's parameters
11  k = 2;
12  m = 101;
13  n = 51;
14  o = 101;
15
16  % Domain's dimensions
17  a = 0;
18  b = 101;
19  c = 0;
20  d = 51;
21  e = 0;
22  f = 101;
23
24  % Spatial step sizes
25  dx = (b - a) / m;
26  dy = (d - c) / n;
27  dz = (f - e) / o;
28
29  % Mimetic operators
30  D = div3D(k, m, dx, n, dy, o, dz);
31  G = grad3D(k, m, dx, n, dy, o, dz);
32  I = interpol3D(m, n, o, 1, 1, 1);
```

Octave script 7.1.: Program `convection_diffusion.m`

Line 30: three-dimensional mimetic divergence operator $div3D(k, m, dx, n, dy, o, dz) = div3D(2, 101, 1, 51, 1, 101, 1)$

Line 31: three-dimensional mimetic gradient operator $grad3D(k, m, dx, n, dy, o, dz) = grad3D(2, 101, 1, 51, 1, 101, 1)$

Line 32: The three-dimensional mimetic interpolation operator $interpol3D(m, n, o, c1, c2, c3)$. Where $c1$: Left interpolation coeff, $c2$: Bottom interpolation coeff, $c3$: Front interpolation coeff.

Line 35: size(G,1) calculates the size of matrix G, 1 indicates that it calculates the number of rows in matrix G. Then zeros(a,1) is a zeros matrix of a rows and 1 column

7.5. Advanced Applications: Coupled PDEs

```

% Pore velocity vector
V = zeros(size(G, 1), 1);
% Density vector
C = zeros(m + 2, n + 2, o + 2);

% Impose initial conditions ---------------------------------------------------
bottom = 10; % Well
top = 15; % Well
seal = 40; % Shale

% Velocity field --------------------------------------------------------------
y = ones(m, n + 1, o);
y(:, seal, :) = 0;
y(:, seal + 5, :) = 0;
y = y(:);
V(((m + 1) * n * o + 1):((m + 1) * n * o + numel(y))) = y; % Shale

% Density ---------------------------------------------------------------------
C(ceil((m + 2) / 2), bottom:top, ceil((o + 2) / 2)) = 1; % ceil((o+2)/2)
C = C(:);
idx = find(C);

```

Octave script 7.2.: Program `convection_diffusion.m`

Part III.

Validation and Benchmarking

8. Operator Validation

8.1. Divergence Operator Tests

8.1.1. Accuracy in 1D

8.1.2. Conservation in 2D

8.1.3. Boundary Compliance in 3D

8.2. Gradient and Laplacian Tests

8.3. Interpolation Robustness Checks

9. Application-Driven Tests

9.1. Stability Analysis for PDE Solvers

MOLE uses GoogleTest framework.

```cpp
/**
 * Nullity test of Divergence operator
 */

#include <gtest/gtest.h>
#include <mole/divergence.h>
#include <mole/operators.h>

void run_nullity_test(int k, Real tol)
{
  int m = 2 * k + 1;
  Real dx = 1;

  Divergence D(k, m, dx);
  arma::vec field(m + 1, arma::fill::ones);
  arma::vec sol = D * field;

  EXPECT_NEAR(arma::norm(sol), 0, tol);
}

TEST(DivergenceTests, Nullity)
{
  Real tol = 1e-10;
  for (int k : {2, 4, 6}) {
    run_nullity_test(k, tol);
  }
}

int main(int argc, char **argv)
{
  ::testing::InitGoogleTest(&argc, argv);
  return RUN_ALL_TESTS();
}
```

C++ script 9.1.: Program test1.cpp

```
#!/usr/bin/env -S octave-cli -qf
% Nullity test of Divergence operator

addpath('/usr/share/mole/matlab/')

tol = 1e-10;

for k = [2, 4, 6, 8]; % Different orders of accuracy
    m = 2 * k + 1;
    dx = 1 / m;
    D = div(k, m, dx);
    field = ones(m + 1, 1);
    sol = D * field;

    if (norm(sol) > tol)
        fprintf("Test FAILED!\n");
    end

end

fprintf("Test PASSED!\n");
```

Octave script 9.1.: Program `test1.m`

```
/**
 * Nullity test of Gradient operator
 */

#include <gtest/gtest.h>
#include <mole/gradient.h>
#include <mole/operators.h>

void run_nullity_test(int k, Real tol)
{
  int m = 2 * k + 1;
  Real dx = 1;

  Gradient G(k, m, dx);
  arma::vec field(m + 2, arma::fill::ones);
  arma::vec sol = G * field;

  ASSERT_LT(arma::norm(sol), tol)
      << "Gradient Nullity Test failed for k = " << k;
}

TEST(GradientTests, Nullity)
{
  Real tol = 1e-10;
  for (int k : {2, 4, 6, 8}) {
    run_nullity_test(k, tol);
  }
}
```

C++ script 9.2.: Program `test2.cpp`

```
#!/usr/bin/env -S octave-cli -qf
% Nullity test of Gradient operator

addpath('/usr/share/mole/matlab/')

tol = 1e-10;

for k = [2, 4, 6, 8]; % Different orders of accuracy
    m = 2 * k + 1;
    dx = 1 / m;
    G = grad(k, m, dx);
    field = ones(m + 2, 1);
    sol = G * field;

    if (norm(sol) > tol)
        fprintf("Test FAILED!\n");
    end

end

fprintf("Test PASSED!\n");
```

Octave script 9.2.: Program `test2.m`

```
/**
 * Nullity test of Laplacian operator
 */
#include <gtest/gtest.h>
#include <mole/laplacian.h>
#include <mole/operators.h>

void run_nullity_test(int k, Real tol)
{
  int m = 2 * k + 1;
  Real dx = 1;

  Laplacian L(k, m, dx);
  arma::vec field(m + 2, arma::fill::ones);
  arma::vec sol = L * field;

  ASSERT_LT(arma::norm(sol), tol)
      << "Laplacian Nullity Test failed for k = " << k;
}

TEST(LaplacianTests, Nullity)
{
  Real tol = 1e-10;
  for (int k : {2, 4, 6}) {
    run_nullity_test(k, tol);
  }
}
```

C++ script 9.3.: Program `test3.cpp`

```
#!/usr/bin/env -S octave-cli -qf
% Nullity test of Laplacian operator

addpath('/usr/share/mole/matlab/')

tol = 1e-10;

for k = [2, 4, 6, 8]; % Different orders of accuracy
    m = 2 * k + 1;
    dx = 1 / m;
    L = lap(k, m, dx);
    field = ones(m + 2, 1);
    sol = L * field;

    if (norm(sol) > tol)
        fprintf("Test FAILED!\n");
    end

end

fprintf("Test PASSED!\n");
```

Octave script 9.3.: Program `test3.m`

```
/**
 * Energy test
 */

#include <algorithm>
#include <gtest/gtest.h>
#include <mole/laplacian.h>
#include <mole/operators.h>

TEST(EnergyTests, EigenvalueTest)
{
  int k = 4;
  Real a = -5;
  Real b = 5;
  int m = 500;
  arma::vec grid = arma::linspace(a, b, m);
  Real dx = grid(1) - grid(0);
  Real tol = 1e-10;

  Laplacian L(k, m - 2, dx);

  std::transform(grid.begin(), grid.end(), grid.begin(),
                 [](Real x) { return x * x; });

  arma::sp_mat V(m, m);
  V.diag(0) = grid;

  arma::sp_mat H = -0.5 * (arma::sp_mat)L + V;
  arma::cx_vec eigval;
  arma::eig_gen(eigval, (mat)H);

  eigval = arma::sort(eigval);
  arma::vec expected{1, 3, 5, 7, 9};

  for (int i = 0; i < expected.size(); ++i) {
    ASSERT_LT(std::norm(std::real(eigval(i) / eigval(0)) - expected(i)), tol)
        << "Energy Test failed for eigenvalue index " << i;
  }
}
```

C++ script 9.4.: Program `test4.cpp`

```
#!/usr/bin/env -S octave-cli -qf
% Energy test

addpath('/usr/share/mole/matlab/')

% Parameters
k = 4;
a = -5;
b = 5;
m = 500;
grid = linspace(a, b, m);
dx = grid(2) - grid(1);
tol = 1e-6;

% Laplacian
L = lap(k, m - 2, dx);

% Transform grid using an anonymous function
grid = arrayfun(@(x) x^2, grid);

% Potential matrix V
V = spdiags(grid', 0, m, m); % Transpose grid to match dimension

% Hamiltonian H
H = -0.5 * L + V;

% Eigenvalue computation
eigval = eig(full(H)); % Convert sparse matrix to full for eig function
eigval = sort(eigval);

% Expected eigenvalues
expected = [1, 3, 5, 7, 9];

% Test for failure
failed = false;

for i = 1:length(expected)

    if (abs(real(eigval(i) / eigval(1)) - expected(i)) > tol)
        fprintf(2, 'Test FAILED!\n');
        failed = true;
    end

end

if ~failed
    fprintf(1, 'Test PASSED!\n');
end
```

Octave script 9.4.: Program `test4.m`

```cpp
/**
 * Poisson accuracy test
 */
#define ARMA_DONT_USE_WRAPPER
// #define ARMA_USE_SUPERLU
#include <gtest/gtest.h>
#include <mole/laplacian.h>
#include <mole/operators.h>

void run_test(int k, arma::vec grid_sizes)
{
  Real west = 0; // Domain's limits
  Real east = 1;

  arma::vec errors(grid_sizes.size());

  for (int i = 0; i < grid_sizes.size(); ++i) {
    int m = grid_sizes(i);       // Number of cells
    Real dx = (east - west) / m; // Step length

    Laplacian L(k, m, dx);
    RobinBC BC(k, m, dx, 1, 1);
    L = L + BC;

    arma::vec grid(m + 2);
    grid(0) = west;
    grid(1) = west + dx / 2.0;
    for (int j = 2; j <= m; j++) {
      grid(j) = grid(j - 1) + dx;
    }
    grid(m + 1) = east;

    arma::vec U = arma::exp(grid);
    U(0) = 0;                    // West BC
    U(m + 1) = 2 * std::exp(1); // East BC

#ifdef EIGEN
    arma::vec computed_solution = Utils::spsolve_eigen(L, U);
#elif LAPACK
    arma::vec computed_solution = arma::spsolve(L, U, "lapack"); // Will use LAPACK
#else
    arma::vec computed_solution = arma::spsolve(L, U); // Will use SuperLU
#endif

    arma::vec analytical_solution = arma::exp(grid);
    errors(i) = arma::max(arma::abs(computed_solution - analytical_solution));
  }

  arma::vec order(errors.size() - 1);
  for (int i = 0; i < errors.size() - 1; ++i) {
    order(i) = std::log2(errors(i) / errors(i + 1));
    ASSERT_GE(order(i), k - 0.5) << "Poisson Test failed for k = " << k;
  }
}

TEST(PoissonTests, Accuracy)
{
  arma::vec grid_sizes = {20, 40};
  for (int k : {2, 4, 6}) {
    run_test(k, grid_sizes);
  }
```

C++ script 9.5.: Program `test5.cpp`

```
#!/usr/bin/env -S octave-cli -qf
% Poisson accuracy test

addpath('/usr/share/mole/matlab/')

west = 0; % Domain's limits
east = 1;
grid_sizes = [20, 40]; % Grid sizes to test

for k = [2, 4, 6]; % Different orders of accuracy

    errors = zeros(size(grid_sizes));

    for i = 1:numel(grid_sizes)
        m = grid_sizes(i); % Number of cells
        dx = (east - west) / m; % Step length
        L = lap(k, m, dx); % 1D Mimetic Laplacian operator

        % Impose Robin BC on Laplacian operator
        a = 1;
        b = 1;
        L = L + robinBC(k, m, dx, a, b);

        % 1D Staggered grid
        grid = [west west + dx / 2:dx:east - dx / 2 east];

        % RHS
        U = exp(grid)';
        U(1) = 0; % West BC
        U(end) = 2 * exp(1); % East BC

        % Solve a linear system of equations
        computed_solution = L \ U;

        % Compute error using L2 norm
        analytical_solution = exp(grid);
        errors(i) = max(abs(computed_solution' - analytical_solution));
    end

    % Compute order of accuracy
    order = zeros(numel(errors) - 1, 1);

    for i = 1:numel(errors) - 1
        order(i) = log2(errors(i) / errors(i + 1));

        if order(i) - k < -0.5
            fprintf("Test FAILED for k = %d!\n", k);
            return
        end

    end

end

fprintf("Test PASSED!\n");
```

Octave script 9.5.: Program `test5.m`

A. Installation Notes

To work through this tutorial requires to have a working installation of MOLE. It relies on `armadillo` [18], a C++ library that provides data structures for sparse matrices. We explain the step-by-step process for Arch Linux and Ubuntu Linux, as both systems have been successfully tested by us.

A.1. Arch Linux

This distribution is supported by a proactive group of developers, package maintainers and support staff that try to provide the latest stable software releases. The steps are outlined in the Program A.1.

```bash
#!/bin/bash

# [1/5] Update the system and install developer tools
sudo pacman --needed --noconfirm -Syu base-devel git
# [2/5] Optionally install Intel MKL as a replacement for LAPACK linear algebra library
sudo pacman -S intel-oneapi-mkl # or depending on your needs: intel-oneapi-basekit, intel-oneapi-hpckit,
↪  lapack, blas-openblas

# [3/5] Install armadillo from https://aur.archlinux.org/armadillo.git
git clone https://aur.archlinux.org/armadillo.git
pushd armadillo
makepkg -s --noconfirm
popd

# [4/5] Install MOLE C++/Octave from https://aur.archlinux.org/libmole.git
git clone https://aur.archlinux.org/libmole.git
pushd libmole
makepkg -s --noconfirm
popd

# [5/5] Install utilities
sudo pacman -S ffmpeg gtest python-matplotlib python-scipy texlive
```

Program code A.1.: Steps for a system-wide installation both C++ and Octave MOLE library via `installerarchlinux.sh`.

A.2. Ubuntu Linux

This Debian-derived distribution is managed by Canonical Ltd. Each 2 years they launch a Long Term Support(LTS) release. The steps are outlined in the Program A.3.

```bash
#!/bin/bash

docker run -it --rm ghcr.io/carlosal1015/mole_examples/libmole-git # or libmole instead of libmole-git
docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
ghcr.io/carlosal1015/mole_examples/libmole-git         latest     a86fb0fef044   3 hours ago     2.16GB
ghcr.io/carlosal1015/mole_examples/libmole             latest     d20a1ec0091b   3 hours ago     2.14GB
```

Program code A.2.: Pull container based on Arch Linux with set up MOLE library via `docker.sh`.

```bash
#!/bin/bash

sudo apt-get update
sudo apt-get --no-install-recommends --yes install \
    build-essential cmake git octave \
    libarmadillo-dev libsuperlu-dev libeigen3-dev \
    ffmpeg libgtest-dev python3-matplotlib python3-scipy texlive-full
sudo apt-get install --reinstall ca-certificates

git clone --filter=blob:none --depth=1 https://github.com/csrc-sdsu/mole.git
sed -i '/^if(POLICY/,+51 s/^/#/' mole/CMakeLists.txt
sed -i '/^set(LINK_LIBS/,+3 s/^/#/' mole/CMakeLists.txt
sed -i '96i set(LINK_LIBS ${ARMADILLO_LIBRARIES} ${OpenBLAS_LIBRARIES} ${LAPACK_LIBRARY}
↪   ${SUPERLU_INSTALL_DIR}/lib/x86_64-linux-gnu/libsuperlu.so)' mole/CMakeLists.txt
sed -i '/^include(/,+9 s/^/#/' mole/tests/cpp/CMakeLists.txt

cmake \
    -S mole \
    -B build \
    -DBUILD_SHARED_LIBS=TRUE \
    -DCMAKE_BUILD_TYPE=None \
    -DCMAKE_CXX_STANDARD=14 \
    -DCMAKE_CXX_COMPILER=g++ \
    -DCMAKE_INSTALL_PREFIX=/usr \
    -Wno-dev
cmake --build build --target mole_C++
sudo cmake --build build --target install
sudo install -d /usr/include/mole
sudo install mole/src/cpp/*.h /usr/include/mole
```

Program code A.3.: Steps for a system-wide installation both C++ and Octave MOLE library vía `installerubuntu.sh`.

Even if you are using Windows, the Docker Desktop WSL 2 backend is ideal for using MOLE via Program A.2 or installing Arch Linux on WSL 2 and following the Program A.1.

B. MOLE Documentation

We split the MOLE documentation in three categories:

B.1. C++ API Reference

C++ MOLE documentation
  It contains API C++ Reference.

General MOLE documentation It contains general information and examples.

B.2. Octave API Reference

Octave MOLE documentation
  It contains API GNU/Octave Reference.

B.3. Python API Reference

`pyMOLE` documentation
  It contains API Python Reference.

C++ docs `#include <iostream>`

`#include <cmath>`

`#include <vector>`.

Octave docs `addpath("/usr/share/")`.

MATLAB docs .

C. Sparse Linear Algebra Libraries

Intel MKL docs .

Openblas docs .

Netlib Lapack docs .

SuperLU docs .

Eigen docs .

Armadillo docs .

Gtest docs .

SciPy sparse docs .

Matplotlib docs .

HDF5 for Python docs .

C.1. Armadillo

We follow this gentle *Introduction to Armadillo*.

C.1.1. Vectors

C.1.2. Matrices

Dense Matrices

Sparse Matrices

Solvers

C.2. Eigen

C.3. SciPy Sparse

See pymole.

C.4. Sparse Arrays Julia

See

```cpp
#define ARMA_DONT_USE_WRAPPER
#include <armadillo>
#include <iostream>

int main()
{
  // Show Armadillo version
  std::cout << "Armadillo version: " << ARMA_VERSION_MAJOR << "."
            << ARMA_VERSION_MINOR << "." << ARMA_VERSION_PATCH << " "
            << ARMA_VERSION_NAME " ";

  int n = 5;
  int m = 4;

  std::cout
      // Vector of n entries
      << arma::vec(n)
      << "\n"
      // Vector of m entries of 2s
      << arma::vec(m).fill(2)
      << "\n"
      // Declare and fill a vector with random values from a uniform
      // distribution.
      << arma::vec(n).randu()
      << "\n"
      // Declare and fill a vector with the values 0.1, 0.2 and 0.3.
      << arma::vec("0.0 0.1 0.2");

  std::cout
      // Matrix of n times m entries
      << arma::mat(n, m)
      << "\n"
      // Matrix of n times m entries of 2s
      << arma::mat(n, m).fill(2.)
      << "\n"
      // Declare and fill a matrix with random values from a uniform
      // distribution
      << arma::mat(n, m).randn()
      << "\n"
      // Declare and fill a vector with the values 0.0, 0.1 and 0.2.
      << arma::mat("0.0 0.1 0.2 ; 1.0 1.1 1.2 ; 2.0 2.1 2.2");

  arma::mat A = arma::mat("0.0 0.1 0.2 ; 1.0 1.1 1.2 ; 2.0 2.1 2.2");
  arma::vec x = arma::vec("20. 10. 30.");
  arma::uvec x_sort_indices =
      arma::sort_index(x); // Get index ordering that sorts x.
  x = x(x_sort_indices);
  A = A.cols(x_sort_indices);

  // A.save(filename);
  // B = arma::mat();  // Initialize an arma::mat variable
  // B.load(filename); // Load content of arma::mat A stored earlier into
  // arma::mat B.

  return 0;
}
```

Program code C.1.: Program `1.cc`

C.5. Fortran Sparse

C.6. Sparse Linear Algebra in Rust

C.7. PETSc sparse matrices in C

D. Visualizing Results with Matplotlib

D.1. Static Plots

D.2. Animating Time-Dependent Solutions

D.3. Exporting Data from MOLE for visualization

D.4. Visualizing 2D/3D Results

```python
def U(
    x: ArrayN[np.float64], a: float = 1.0, b: float = 1.0, c: float = 1.0
) -> ArrayN[np.float64]:
    """Params
    a: height of the curve's peak
    b: position of the center of the peak
    c: width of the bell

    See https://en.wikipedia.org/wiki/Gaussian_function
    """

    return a * np.exp(-np.divide(np.power(x - b, 2), 2 * np.power(c, 2)))
```

```python
a: float = 5
b: float = 2
c: float = 0.2
speed: float = 0.5
fig: Figure
ax: Axes
fig, ax = plt.subplots(layout="constrained")
(ln,) = ax.plot(
    [],
    [],
    color="DarkBlue",
    label="Exacta",
    linestyle="solid",
    linewidth=0.5,
)
ax.set_xlim(left=x[0], right=x[-1])
ax.set_ylim(bottom=0, top=U(x=x, a=a).max())
ax.grid(c="gray", linewidth=0.1, linestyle="dashed")
ax.set_xlabel(xlabel=r"$x$ (m)")
ax.set_ylabel(ylabel=r"$y$ (m)")
ax.legend(loc="upper left")
ax.set_aspect("equal", adjustable="box")
ax.spines["bottom"].set_color("none")
ax.spines["top"].set_alpha(alpha=0.8)
ax.spines["top"].set_edgecolor(color="gray")
ax.spines["top"].set_linewidth(w=0.5)
ax.spines["left"].set_alpha(alpha=0.8)
ax.spines["left"].set_edgecolor(color="gray")
ax.spines["left"].set_linewidth(w=0.5)
ax.spines["right"].set_alpha(alpha=0.8)
ax.spines["right"].set_edgecolor(color="gray")
ax.spines["right"].set_linewidth(w=0.5)
```

```python
def init():
    ln.set_data(x, U(x=x, a=a, b=b, c=c))

    return (ln,)
```

```python
def update(frame):
    ln.set_data(x, U(x=x - speed * frame * Δt, a=a, b=b, c=c))
    ax.set_title(
        label=rf"Travel wave $U\left(x - ct\right)=U\left(x, t={frame * Δt:.2f}s\right)$ with $c={speed}$
        ↪  "
        + r"$\unit{\metre\per\second}$",
        loc="center",
        fontsize=12,
        wrap=True,
    )

    return (ln,)
```

```python
anim = FuncAnimation(
    fig=fig,
    func=update,
    frames=tqdm(iterable=range(t.size), file=sys.stdout, colour="green"),
    interval=1,
    init_func=init,
    blit=True,
).save(
    filename="animation.mkv",
    writer=FFMpegWriter(
```

E. Supplementary Methods

E.1. Vector Calculus Theorems

Vector Fields are maps of the form

$$F\colon \mathbb{R}^n \to \mathbb{R}^n$$

Scalar fields are maps of the form

$$\phi\colon \mathbb{R}^n \to \mathbb{R}$$

The gradient of a scalar field is a vector field, defined as

$$\nabla\phi = \frac{\partial\phi}{\partial x^i}\mathbf{e}_i$$

Given Cartesian coordinates x^i with $i = 1, \ldots, n$ on $\mathbb{R}^n$, the gradient of ϕ is defined as

$$\nabla\phi = \frac{\partial\phi}{\partial x^i}\mathbf{e}_i.$$

A vector field $\mathbf{F}$ is called conservative if it can be written as

$$\mathbf{F} = \nabla\phi.$$

Given two vectors, a derivative acting on vector fields known as the divergence

$$\nabla\cdot\mathbf{F} = \left(\mathbf{e}_i\frac{\partial}{\partial x^i}\right)\cdot(\mathbf{e}_j F_j) = \frac{\partial F_i}{\partial x^i}$$

where $\mathbf{e}_i\cdot\mathbf{e}_j = \delta_{ij}$. Note that the gradient of a scalar filed gave a vector field. Now the divergence of a vector field gives a scalar field. If $\mathbf{F}\colon \mathbb{R}^3 \to \mathbb{R}^3$, we can take cross product.

$$\nabla\times\mathbf{F} = () \times () = \epsilon_{ijk}\frac{\partial F_j}{\partial x^i}\mathbf{e}_k$$

$$\nabla\times\mathbf{F} = \left(\frac{\partial F_3}{\partial x^2} - \frac{\partial F_2}{\partial x^3}, \frac{\partial F_1}{\partial x^3} - \frac{\partial F_3}{\partial x^1}, \frac{\partial F_2}{\partial x^1} - \frac{\partial F_1}{\partial x^2}\right) = \begin{vmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ \frac{\partial}{\partial x^1} & \frac{\partial}{\partial x^2} & \frac{\partial}{\partial x^3} \\ F_1 & F_2 & F_3 \end{vmatrix}$$

$$\nabla(\alpha\phi + \psi) = \alpha\nabla\phi + \nabla\psi.$$
$$\nabla\cdot(\alpha\mathbf{F} + \mathbf{G}) = \alpha\nabla\cdot\mathbf{F} + \nabla\cdot\mathbf{G}.$$
$$\nabla\times(\alpha\mathbf{F} + \mathbf{G}) = \alpha\nabla\times\mathbf{F} + \nabla\times\mathbf{G}.$$

for any scalar fields ϕ and ψ, vector fields $\mathbf{F}$ and $\mathbf{G}$, and any constant α.

$$\begin{aligned}\nabla(\phi\psi) &= \phi\nabla\psi + \psi\nabla\phi.\\ \nabla\cdot(\phi\mathbf{F}) &= (\nabla\phi)\cdot\mathbf{F} + \phi(\nabla\cdot\mathbf{F})\\ \nabla\times(\phi\mathbf{F}) &= (\nabla\phi)\times\mathbf{F} + \phi(\nabla\times\mathbf{F})\end{aligned}$$

$$\nabla\cdot(\phi\mathbf{F}) = \frac{\partial(\phi F_i)}{\partial x^i} = \frac{\partial\phi}{\partial x^i}F_i + \phi\frac{\partial F_i}{\partial x^i} = (\nabla\phi)\cdot\mathbf{F} + \phi(\nabla\cdot\mathbf{F}).$$

$$\nabla\cdot(F\times G) = (\nabla\times\mathbf{F})\cdot\mathbf{G} - \mathbf{F}\cdot(\nabla\times\mathbf{G}).$$

$$\mathbf{F}\cdot\nabla = F_i\frac{\partial}{\partial x^i}$$

$$\mathbf{F}\times\nabla = \mathbf{e}_k\epsilon_{ijk}F_i\frac{\partial}{\partial x^j}$$

$$\begin{aligned}\nabla(\mathbf{F}\cdot\mathbf{G}) &= \mathbf{F}\times(\nabla\times\mathbf{G}) + \mathbf{G}\times(\nabla\times\mathbf{F}) + (\mathbf{F}\cdot\nabla)\mathbf{G} + (\mathbf{G}\cdot\nabla)\mathbf{F}.\\ \nabla\times(\mathbf{F}\times\mathbf{G}) &= (\nabla\cdot\mathbf{G})\mathbf{F} - (\nabla\cdot\mathbf{F})\mathbf{G} + (\mathbf{G}\cdot\nabla)\mathbf{F} - (\mathbf{F}\cdot\nabla)\mathbf{G}.\end{aligned}$$

$$\nabla\times\mathbf{F} = 0 \iff \mathbf{F} = \nabla\phi \iff \oint_C \mathbf{F}\cdot\mathrm{d}\mathbf{x} = 0.$$

$$\nabla\cdot\mathbf{F} = 0 \iff \mathbf{F} = \nabla\times\mathbf{A}.$$

The Laplacian is a second order differential operator defined by

$$\nabla^2 = \nabla\cdot\nabla = \frac{\mathrm{d}^3}{\mathrm{d}(x^i)^2\mathrm{d}x^i}$$

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}.$$

$$\nabla\times(\nabla\times\mathbf{F}) = \nabla(\nabla\cdot\mathbf{F}) - \nabla^2\mathbf{F}.$$

$$\nabla^2\mathbf{F} = \nabla(\nabla\cdot\mathbf{F}) - \nabla\times(\nabla\times\mathbf{F})$$

$$\begin{aligned}\nabla\cdot\mathbf{E} &= \frac{\rho}{\epsilon_0}\\ \nabla\times\mathbf{E} &= -\frac{\partial\mathbf{B}}{\partial t}\\ \nabla\cdot\mathbf{B} &= 0\\ \nabla\times\mathbf{B} &= \mu_0\left(\mathbf{J} + \epsilon_0\frac{\partial\mathbf{E}}{\partial t}\right)\end{aligned}$$

$$\mathrm{d}f = \frac{\partial f}{\partial u}\,\mathrm{d}u + \frac{\partial f}{\partial v}\,\mathrm{d}v + \frac{\partial f}{\partial w}\,\mathrm{d}w = \nabla f \cdot \left(h_u \mathbf{e}_u\,\mathrm{d}u + h_v \mathbf{e}_v\,\mathrm{d}v + h_w \mathbf{e}_w\,\mathrm{d}w\right)$$

$$\nabla = \frac{1}{h_u}\mathbf{e}_u \frac{\partial}{\partial u} + \frac{1}{h_v}\mathbf{e}_v \frac{\partial}{\partial v} + \frac{1}{h_w}\mathbf{e}_w \frac{\partial}{\partial w}$$

$$F\left(u, v, w\right) = F_u \mathbf{e}_u + F_v \mathbf{e}_v + F_w \mathbf{e}_w$$

$$\nabla \cdot \mathbf{F} = \frac{1}{h_u h_v h_w}\left(\frac{\partial}{\partial u}\left(h_v h_w F_u\right) + \frac{\partial}{\partial v}\left(h_u h_w F_v\right) + \frac{\partial}{\partial w}\left(h_u h_v F_w\right)\right)$$

$$\nabla \times \mathbf{F} = \frac{1}{h_u h_v h_w}\begin{vmatrix} h_u \mathbf{e}_u & h_v \mathbf{e}_v & h_w \mathbf{e}_w \\ \frac{\partial}{\partial u} & \frac{\partial}{\partial v} & \frac{\partial}{\partial w} \\ h_u F_u & h_v F_v & h_w F_w \end{vmatrix}$$

$$\nabla^2 f = \nabla \cdot \nabla f = \frac{1}{h_u h_v h_w}\left[\frac{\partial}{\partial u}\left(\frac{h_v h_w}{h_u}\frac{\partial f}{\partial u}\right) + \frac{\partial}{\partial v}\left(\frac{h_u h_w}{h_v}\frac{\partial f}{\partial v}\right) + \frac{\partial}{\partial w}\left(\frac{h_u h_v}{h_w}\frac{\partial f}{\partial w}\right)\right]$$

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}\frac{\partial^2 f}{\partial z^2}.$$

$$\nabla^2 f = \frac{1}{\rho}\frac{\partial}{\partial \rho}\left(\rho \frac{\partial f}{\partial \rho}\right) + \frac{1}{\rho^2}\frac{\partial^2 f}{\partial \phi^2} + \frac{\partial^2 f}{\partial z^2}.$$

$$\nabla^2 f = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 \frac{\partial f}{\partial r}\right) + \frac{1}{r^2 \sin\theta}\frac{\partial}{\partial \theta}\left(\sin\theta \frac{\partial f}{\partial \theta}\right) + \frac{1}{r^2 \sin^2\theta}\frac{\partial^2 f}{\partial \phi^2}.$$

E.1.1. The Divergence Theorem

For a smooth vector field $\mathbf{F}\left(\mathbf{x}\right)$ over $\mathbb{R}^3$,

$$\int_V \nabla \cdot \mathbf{F}\,\mathrm{d}V = \int_S \mathbf{F} \cdot \mathrm{d}\mathbf{S}$$

where V is a bounded region whose boundary $\partial V = S$ is a piecewise smooth closed surface. The integral on the right-hand side is taken with the normal $\mathbf{n}$ pointing outward.

$$\int_V \nabla \cdot \mathbf{F}\,\mathrm{d}V \approx V \nabla \cdot \mathbf{F}\left(\mathbf{x}\right).$$

$$\nabla \cdot \mathbf{F} = \lim_{V \to 0} \frac{1}{V}\int_S \mathbf{F} \cdot \mathrm{d}\mathbf{S}$$

$$\int_S \mathbf{F} \cdot \mathrm{d}\mathbf{S} = \int_V \nabla \cdot \mathbf{F}\,\mathrm{d}V.$$

We introduce the density $\rho(\mathbf{x},t)$ of the conserved object. The total electric charge in some region V is then given by the integral

$$Q = \int_V \rho \, \mathrm{d}V.$$

The conservation of charge is captured by the following statement: there exists a vector field $\mathbf{J}(\mathbf{x},t)$ such that

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \mathbf{J} = 0$$

This is known as the continuity equation and $\mathbf{J}$ is called the current density.

$$\frac{\partial Q}{\partial t} = \int_V \frac{\partial \rho}{\partial t} \, \mathrm{d}V = -\int_V \nabla \cdot \mathbf{J} \, \mathrm{d}V = -\int_S \mathbf{J} \cdot \mathrm{d}\mathbf{S}.$$

The continuity equation plays an important role in Fluid Dynamics where the mass is conserved. In that case, $\rho(\mathbf{x},t)$ is the density of the fluid and the current is $\mathbf{J} = \rho\mathbf{u}$ where $\mathbf{u}(\mathbf{x},t)$ is the velocity field. The continuity equation then read as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0.$$

In many circumstances, liquids can be modelled as incompressible, meaning that $\rho(\mathbf{x},t)$ is a constant in both space and time. In these circumstances, we have $\dot{\rho} = \nabla \rho = 0$ and the continuity equation tell us that the velocity field is necessarily solenoidal:

$$\nabla \cdot \mathbf{u} = 0$$

There is a close connection between conserved quantities and the idea of diffusion. We will illustrate this with the idea of energy conservation. The story takes slightly different from depending on the context, but here we will think of the energy contained in a hot gas. First, since energy is conserved there is necessarily a corresponding continuity equation.

$$\frac{\partial \mathcal{E}}{\partial t} + \nabla \cdot \mathbf{J} = 0$$

where $\mathcal{E}(\mathbf{x},t)$ is the energy density of the gas, and $\mathbf{J}$ is the heat current which tell us how energy is transported from one region of space to another. First, the energy density in a gas is proportional to the temperature of the gas

$$\mathcal{E}(\mathbf{x},t) = c_V T(\mathbf{x},t)$$

where c_V is the specific heat capacity.

$$\mathbf{J} = -\kappa \nabla T$$

where κ is called the thermal conductivity. This relation is known as Fick's law.

Let $P(x, y)$ and $Q(x, y)$ be smooth functions on $\mathbb{R}^2$. Then

$$\int_A \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} \right) \mathrm{d}A = \oint_C (P \, \mathrm{d}x + Q \, \mathrm{d}y)$$

where A is a bounded region in the plane and $C = \partial A$ is a piecewise smooth, non-intersecting closed curve which is traversed anti-clockwise.

E.1.2. The Stokes Theorem

Stoke's theorem is an extension of Green's theorem, but where the surface is no longer restricted to lie in a plane.

Let S be a smooth surface in $\mathbb{R}^3$ with boundary $C = \partial S$ a piecewise smooth curve. For any smooth vector field $\mathbf{F}(\mathbf{x})$, we have

$$\int_S \nabla \times \mathbf{F} \cdot \mathrm{d}\mathbf{S} = \int_C \mathbf{F} \cdot \mathrm{d}\mathbf{x}$$

$$\int_S \nabla \times \mathbf{F} \cdot \mathrm{d}\mathbf{S} \approx A\mathbf{n} \cdot (\nabla \times \mathbf{F}) .$$

$$\mathbf{n} \cdot (\nabla \times \mathbf{F}) = \lim_{A \to 0} \int_C \mathbf{F} \cdot \mathrm{d}\mathbf{x}.$$

In other words, at any given point, the value of $\nabla \times \mathbf{F}$ in the direction $\mathbf{n}$ tell us about the circulation of $\mathbf{F}$ in the plane normal to $\mathbf{n}$.

$$\nabla \times \mathbf{F} = 0 \implies \oint_C \mathbf{F} \cdot \mathrm{d}\mathbf{x} = 0$$

for all closed curve C.

E.1.3. Tensor Fields

A tensor field over $\mathbb{R}^3$ is the assignment of a tensor $T_{i,\ldots,k}(\mathbf{x})$ to every point $\mathbf{x} \in \mathbb{R}^3$. This is a generalization of a vector field

$$\mathbf{F} \colon \mathbb{R}^3 \to \mathbb{R}^3$$

to a map of the kind

$$T \colon \mathbb{R}^3 \to \mathbb{R}^m$$

with m the number of components of the tensor.

E.2. Method of characteristics

[7, 2]

Let's consider the problem of

$$\begin{cases} \partial_t u + c\partial_x u = 0, & x \in (0,1)\,, t > 0. \\ u(0,t) = u(1,t)\,, & t > 0. \\ u(x,0) = g(x)\,, & x \in [0,1]\,. \end{cases}$$

Consider the problem for the explicit form of linear first-oder PDEs in two independent variables

$$\begin{cases} a(x,y)\,\partial_x u + b(x,y)\,\partial_y u = c_1(x,y)\,u + c_2(x,y)\,, \\ u(x,y)\,\text{given for}\,(x,y) \in \Gamma. \end{cases}$$

to be solved in some domain $\Omega \subset \mathbb{R}^2$ with data given on some curve $\Gamma \subset \overline{\Omega}$. Often the $\Gamma \subset \partial\Omega \subset \mathbb{R}^2$ it will just be one of the coordinate axes.

We find the characteristics, i.e., the curves which follow these directions, by solving

$$\frac{\mathrm{d}x}{\mathrm{d}s} = a(x(s)\,,y(s))\,, \qquad \frac{\mathrm{d}y}{\mathrm{d}s} = b(x(s)\,,y(s))\,.$$

Now suppose u is a solution to the PDE. Let $z(s)$ denote the values of the solution u along a characteristic; i.e.,

$$z(s) := u(x(s)\,,y(s))\,.$$

Then by the chain rule, we have

$$\frac{\mathrm{d}z}{\mathrm{d}s} = \partial_x u(x(s)\,,y(s))\,\frac{\mathrm{d}x}{\mathrm{d}s}(x(s)\,,y(s)) + \partial_y u(x(s)\,,y(s))\,\frac{\mathrm{d}y}{\mathrm{d}s}(x(s)\,,y(s))\,.$$
$$\frac{\mathrm{d}z}{\mathrm{d}s} = \partial_x u(x(s)\,,y(s))\,a(x(s)\,,y(s)) + \partial_y u(x(s)\,,y(s))\,b(x(s)\,,y(s))\,.$$
$$\frac{\mathrm{d}z}{\mathrm{d}s} = c_1(x(s)\,,y(s))\,z(s) + c_2(x(s)\,,y(s))\,.$$

Definición 1: Characteristics equations

There are three *dependent variables* x, y and z and one *independent* variable s.

$$\begin{cases} \frac{\mathrm{d}x}{\mathrm{d}s}(s) & = a(x(s)\,,y(s))\,. \\ \frac{\mathrm{d}y}{\mathrm{d}s}(s) & = b(x(s)\,,y(s))\,. \\ \frac{\mathrm{d}z}{\mathrm{d}s}(s) & = c_1(x(s)\,,y(s))\,z(s) + c_2(x(s)\,,y(s))\,. \end{cases}$$

$$a(x,y)\,\frac{\partial u}{\partial x} + b(x,y)\,\frac{\partial u}{\partial y} = c_1(x,y)\,u + c_2(x,y)\,, u \text{ given for } (x,y) \in \Gamma$$

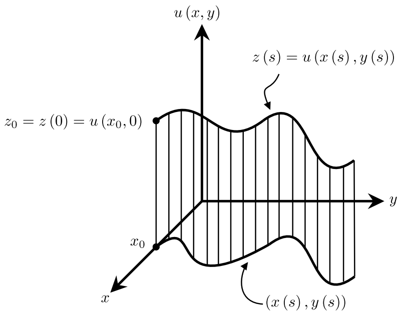


Figure E.1.: The solution u is described by the surface defined by $z = u(x,y)$. From any point x_0 on the x-axis, there is a curve $(x(s), y(s))$ in the xy-plane, upon which wa can calculate the solution $z = u(x(s), y(s))$. Knowing only the structure of the PDE, x_0 and z_0 we can solve ODEs to find the part of the solution surface which lies above the curve.

where we have a linear PDE in the independent variables x and y with a given functions a, b, c_1 and c_2 of (x,y). $\Gamma \subset \partial\Omega$. We find the characteristics, i.e., the curves which follow these directions, by solving

$$\frac{\mathrm{d}x}{\mathrm{d}s} = a(x(s), y(s)).$$
$$\frac{\mathrm{d}y}{\mathrm{d}s} = b(x(s), y(s)).$$
$$z(s) = u(x(s), y(s)).$$

$$\frac{\mathrm{d}z}{\mathrm{d}s} = c_1(x(s), y(s))\, z(s) + c_2(x(s), y(s)).$$

E.3. Method of separation of variables

F. Code Repositories and Resources

References

[1] Daniele Andreucci et al. “Some Numerical Results on Chemotactic Phenomena in Stem Cell Therapy for Cardiac Regeneration”. In: *Mathematics* 12.13 (2024). ISSN: 2227-7390. DOI: 10.3390/math12131937. URL: https://www.mdpi.com/2227-7390/12/13/1937.

[2] Daniel Arrigo. *An Introduction to Partial Differential Equations*. Cham: Springer International Publishing, 2023. ISBN: 978-3-031-22087-6.

[3] Carlos Aznarán. pyMOLE*: Python Mimetic Operators Library Enhanced*. URL: https://github.com/carlosal1015/pymole (visited on 03/21/2025).

[4] Jared Brzenski. “Building an Ocean Model Using Mimetic Operators”. Doctoral Dissertation. California: UC Irvine, 2024. URL: https://escholarship.org/uc/item/5bt2c69f.

[5] Jared Brzenski and Jose E. Castillo. “Solving Navier-Stokes with mimetic operators”. In: *Computers & Fluids* 254 (2023), p. 105817. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2023.105817.

[6] Jose E. Castillo and Guillermo F. Miranda. *Mimetic Discretization Methods*. 1st ed. Springer International Publishing, 2013. DOI: 10.1201/b14575.

[7] Rustum Choksi. *Partial Differential Equations: A First Course*. American Mathematical Society, Apr. 2022. ISBN: 978-1-4704-6491-2.

[8] Johnny Corbino and Jose E. Castillo. “High-order mimetic finite-difference operators satisfying the extended Gauss divergence theorem”. In: *Journal of Computational and Applied Mathematics* 364 (2020), p. 112326. ISSN: 0377-0427. DOI: 10.1016/j.cam.2019.06.042.

[9] Johnny Corbino, Miguel A. Dumett, and Jose E. Castillo. “MOLE: Mimetic Operators Library Enhanced”. In: *Journal of Open Source Software* 9.99 (2024), p. 6288. DOI: 10.21105/joss.06288. URL: https://doi.org/10.21105/joss.06288.

[10] George Fitton. *A Comparative Study of Computational Methods in Cosmic Gas Dynamics Continued*. 2010. URL: https://www.reading.ac.uk/maths-and-stats/-/media/project/uor-main/schools-departments/maths/documents/george-fitton.pdf.

[11] Yessica Judith Gonzales Aredo. “Convergencia del método mimético para la ecuación de difusión no estática”. MA thesis. Trujillo: Universidad Nacional de Trujillo, 2023. URL: https://dspace.unitru.edu.pe/items/1742d2ae-7727-4c4b-9bf9-0cadbbb75d3c.

[12] Konstantin Lipnikov, Gianmarco Manzini, and Mikhail Shashkov. "Mimetic finite difference method". In: *Journal of Computational Physics* 257 (2014). Physics-compatible numerical methods, pp. 1163–1227. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2013.07.031.

[13] Abdul Lugo and Giovanni Calderón. *Un Análisis Comparativo de los Métodos Miméticos, Diferencias Finitas y Elementos Finitos para problemas Estacionarios.* 2015. arXiv: 1504.04913 [math.NA]. URL: https://arxiv.org/abs/1504.04913.

[14] Sergio Andrés Rivera Antolinez. "Un análisis comparativo del método de diferencias finitas y el método mimético para la ecuación de convección difusión unidimensional". Bachelor's Thesis. Universidad Industrial de Santander, 2024. URL: https://noesis.uis.edu.co/handle/20.500.14071/44577.

[15] Bertha K. Rodriguez-Chavez and Yessica E. Zarate-Pedrera. "Spectral Differentiation and Mimetic Methods for Solving the Scalar Burger's Equation". In: *Selecciones Matemáticas* 11.02 (Dec. 2024), pp. 259–270. DOI: 10.17268/sel.mat.2024.02.05. URL: https://revistas.unitru.edu.pe/index.php/SSMM/article/view/6157.

[16] Nicolas P. Rougier. *Scientific Visualization: Python + Matplotlib.* Ed. by Nicolas P. Rougier. Nov. 2021. URL: https://inria.hal.science/hal-03427242.

[17] Franco Rubio López and Mardo Gonzales Herrera. "Algorítmo para la Ecuación de Difusión en Estado Estacionario 2D usando el Método Mimético en Diferencias Finitas." In: *Selecciones Matemáticas* 1.01 (Apr. 2015). DOI: 10.17268/sel.mat.2014.01.04. URL: https://revistas.unitru.edu.pe/index.php/SSMM/article/view/826.

[18] Conrad Sanderson and Ryan Curtin. "Practical Sparse Matrices in C++ with Hybrid Storage and Template-Based Expression Optimisation". In: *Mathematical and Computational Applications* 24.3 (2019). ISSN: 2297-8747. DOI: 10.3390/mca24030070.

[19] G. Sosa Jones, J. Arteaga, and O. Jiménez. "A study of mimetic and finite difference methods for the static diffusion equation". In: *Computers & Mathematics with Applications* 76.3 (2018), pp. 633–648. ISSN: 0898-1221. DOI: 10.1016/j.camwa.2018.05.004.

[20] David Tong. *Vector Calculus.* https://www.damtp.cam.ac.uk/user/tong/vc.html. [Online; accessed 21-March-2025]. 2021.

[21] Lourenço Beirão da Veiga, Konstantin Lipnikov, and Gianmarco Manzini. *The Mimetic Finite Difference Method for Elliptic Problems.* Cham: Springer International Publishing, 2014, pp. 41–65. ISBN: 978-3-319-02663-3. DOI: 10.1007/978-3-319-02663-3_2.

[22] Angel Boada Velazco. "High order mimetic finite differences on nontrivial problems". Doctoral Dissertation. San Diego: San Diego State University, 2021. URL: https://digitalcollections.sdsu.edu/do/e67f17cb-b906-4847-a574-a8781e581024.

Indexx