Tutorial for MOLE

based on commit `1da8a00`

Carlos Aznarán[†] Adelaida Otazu[‡]

March 18, 2025

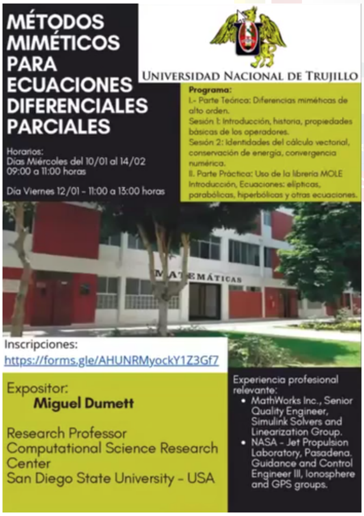
[†]Universidad Nacional de Ingeniería
Av. Tupac Amaru, s/n, Lima, Perú
`caznaranl@uni.pe`
0000-0001-8314-2271
[‡]Universidad Nacional del Altiplano Puno
Avenida El Sol, Puno, Perú
`aotazu@unap.edu.pe`
0000-0003-4793-0400

Preface

This document is for newcomers to MOLE (Mimetic Operator's Library Enhanced) with a solid foundation in numerical analysis for Partial Differential Equations (PDEs). The library's algorithms are continuously evolving, with examples implemented in GNU Octave/MATLAB and C++ using the Armadillo sparse linear algebra library. Currently, these are only two language implementations available. However, mastering this content will enable users to translate the algorithms into other high-performance scientific programming languages, such as Julia, Python, Fortran, C and Rust.

The main goal of this manual is to provide clear explanations and complete 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 libreria MOLE
Introducción, Ecuaciones: elípticas, parabólicas, hiperbólicas y otras ecuaciones.

Inscripciones:
https://forms.gle/AHUNRMyockY1Z3Gf7

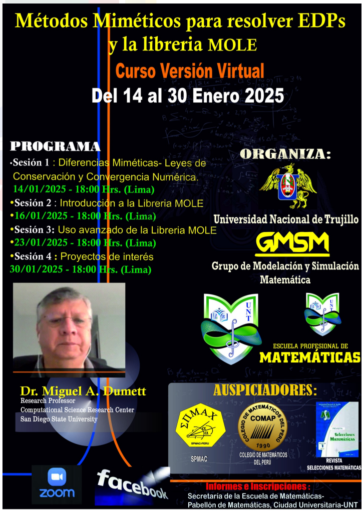
Expositor:
Miguel Dumett

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.

MATEMÁTICAS



Métodos Miméticos para resolver EDPs y la libreria 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

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

March 2025 — Adelaida Otazu

Contents

List of Listings

Part I.

The Foundations of Mimetic Difference Operators 1D

1. Mathematical and Numerical Foundations of the Mimetic Difference Method

1.1. Introduction and Objectives

1.2. Notation and Prerequisites

Δx	tamaño de paso de la malla en la dirección x
Δy	tamaño de paso de la malla en la dirección y
Δz	tamaño de paso de la malla en la dirección z
$\mathbf{D}$	operador divergencia mimético
$\mathbf{G}$	operador gradiente mimético
$\mathbf{L}$	operador laplaciano mimético
$\mathbf{C}$	operador rotacional mimético
$\mathbf{B}$	operador frontera mimético

1.3. Theoretical Considerations

1.3.1. Consistency

1.3.2. Stability

1.3.3. Convergence

1.4. A Short History of Numerical Methods

1.5. Summary and Conclusions

MOLE is an open-source library that implements high-order mimetic operators [7]. Let $\Omega = [a, b]$.

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

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

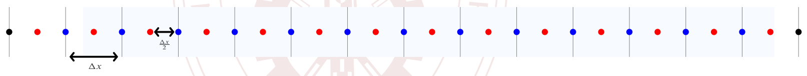


Figure 1.1.: 1D Staggered grid.

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

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

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

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

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

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

Compact operators

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

2. Mimetic Operadors List

2.1. Divergence 1D

2.2. Gradient 1D

2.3. Laplacian 1D

2.4. Interpolation 1D

2.5. Divergence 2D

2.6. Gradient 2D

2.7. Laplacian 2D

2.8. Interpolation 2D

2.9. Divergence 3D

2.10. Gradient 3D

2.11. Laplacian 3D

2.12. Interpolation 3D

```
function D = div(k, m, dx)
% Returns a m+2 by m+1 one-dimensional mimetic divergence operator
%
% Parameters:
%                k : Order of accuracy
%                m : Number of cells
%               dx : Step size
% ----------------------------------------------------------------------------
% SPDX-License-Identifier: GPL-3.0-or-later
% © 2008-2024 San Diego State University Research Foundation (SDSURF).
% See LICENSE file or https://www.gnu.org/licenses/gpl-3.0.html for details.
% ----------------------------------------------------------------------------

    % Assertions:
    assert(k >= 2, 'k ≥ 2');
    assert(mod(k, 2) == 0, 'k % 2 = 0');
    assert(m >= 2*k+1, ['m ≥ ' num2str(2*k+1) ' for k = ' num2str(k)]);

    D = sparse(m+2, m+1);

    switch k
        case 2
            for i = 2:m+1
               D(i, i-1:i) = [-1 1];
            end

        case 4
            A = [-11/12 17/24 3/8 -5/24 1/24];
            D(2, 1:5) = A;
            D(m+1, m-3:end) = -fliplr(A);
            for i = 3:m
               D(i, i-2:i+1) = [1/24 -9/8 9/8 -1/24];
            end

        case 6
            A = [-1627/1920  211/640  59/48  -235/192 91/128 -443/1920 31/960;  ...
                    31/960  -687/640 129/128   19/192 -3/32    21/640  -3/640];
            D(2:3, 1:7) = A;
            D(m:m+1, m-5:end) = -rot90(A,2);
            for i = 4:m-1
                D(i, i-3:i+2) = [-3/640 25/384 -75/64 75/64 -25/384 3/640];
            end

        case 8
            A = [-1423/1792     -491/7168   7753/3072 -18509/5120  3535/1024 -2279/1024  953/1024
            ↪  -1637/7168  2689/107520;  ...
                  2689/107520 -36527/35840  4259/5120   6497/15360 -475/1024  1541/5120 -639/5120
                  ↪  1087/35840  -59/17920;  ...
                   -59/17920    1175/21504 -1165/1024   1135/1024    25/3072  -251/5120   25/1024
                   ↪  -45/7168     5/7168];
            D(2:4, 1:9) = A;
            D(m-1:m+1, m-7:end) = -rot90(A,2);
            for i = 5:m-2
                D(i, i-4:i+3) = [5/7168 -49/5120 245/3072 -1225/1024 1225/1024 -245/3072 49/5120
                ↪  -5/7168];
            end
    end
    D = (1/dx).*D;
end
```

Program 1: Program `div.m`

Program 2: Program `divergence.h`

2.13. Interpolation 1D from center to nodes

2.14. Interpolation 2D from center to nodes

2.15. Interpolation 3D from center to nodes

2.16. Interpolation 1D from center to faces

2.17. Interpolation 2D from center to faces

2.18. Interpolation 3D from center to faces

2.19. Interpolation 1D from nodes to center

2.20. Interpolation 2D from nodes to center

2.21. Interpolation 3D from nodes to center

2.22. Interpolation 1D from faces to center

2.23. Interpolation 2D from faces to center

2.24. Interpolation 3D from faces to center

2.25. Add Boundary Conditions 1D

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.

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

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

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

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

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

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.

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

■

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

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

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

```python
import numpy as np
from scipy.sparse import csr_matrix, identity


def div1D(k, m, dx):
    """ Computes a m+2 by m+1 one-dimensional mimetic divergence operator

    Arguments:
        k (int): Order of accuracy
        m (int): Number of cells
        dx (float): Grid step size

    Returns:
        :obj:`ndarray` containing discrete divergence operator
    """

    assert k >= 2, "Wrong order of accuracy: {}".format(k)
    assert k % 2 == 0, "Order of accuracy must be an even number: {}".format(k)
    assert m >= 2*k + 1, "m must be >= {} for k = {}".format(2*k + 1, k)

    """
    Dimensions of matrix D
    """
    n_rows = m + 2
    n_cols = m + 1

    D = csr_matrix((n_rows, n_cols), dtype=np.float)

    """
    Fill the middle of D
    """
    # Bandwidth = k
    neighbors = np.arange(0.5 - k/2., k/2. + 0.5)

    """
    Create a k by k Vandermonde matrix based on the neighbors
    """
    A = np.transpose(np.vander(neighbors))

    """
    First-order derivative
    """
    b = np.zeros((k, 1), dtype=np.float)
    b[k-2, None] = 1.

    """
    Solve the linear system to get the coefficients
    """
    coeffs = np.transpose(np.linalg.solve(A, b))

    j = 0
    for i in range(int(k/2), int(n_rows - k/2)):
        D[i, j:j+k] = coeffs
        j = j + 1

    """
    Create A
    """
    p = int(k/2 - 1)
    q = int(k + 1)
    A = csr_matrix((p, q), dtype=np.float)
    # For each row of A
    for i in range(p):
        """
        k+1 points are used for the boundaries
        Shifting the stencil to the right
        """
        neighbors = np.arange(-0.5 - i, q - i - 0.5)
        V = np.transpose(np.vander(neighbors))
        b = np.zeros((q, 1), dtype=np.float)
        b[q-2, None] = 1.
        coeffs = np.transpose(np.linalg.solve(V, b))
        A[i, 0:q] = coeffs

    """
    Insert A into D (upper-left corner of D)
    """
    if A.count_nonzero() != 0:
        D[1:p+1, 0:q] = A
    """
    Permutation matrices
    """
    Pp = csr_matrix(np.fliplr(identity(p).toarray()), dtype=np.float)
    Pq = csr_matrix(np.fliplr(identity(q).toarray()), dtype=np.float)

    """
```

```matlab
function G = grad(k, m, dx)
% Returns a m+1 by m+2 one-dimensional mimetic gradient operator
%
% Parameters:
%                 k : Order of accuracy
%                 m : Number of cells
%                dx : Step size
% ----------------------------------------------------------------------------
% SPDX-License-Identifier: GPL-3.0-or-later
% © 2008-2024 San Diego State University Research Foundation (SDSURF).
% See LICENSE file or https://www.gnu.org/licenses/gpl-3.0.html for details.
% ----------------------------------------------------------------------------

    % Assertions:
    assert(k >= 2, 'k ≥ 2');
    assert(mod(k, 2) == 0, 'k % 2 = 0');
    assert(m >= 2*k, ['m ≥ ' num2str(2*k) ' for k = ' num2str(k)]);

    G = sparse(m+1, m+2);

    switch k
        case 2
            A = [-8/3 3 -1/3];
            G(1, 1:3) = A;
            G(end, end-2:end) = -fliplr(A);
            for i = 2:m
               G(i, i:i+1) = [-1 1];
            end

        case 4
            A = [-352/105  35/8  -35/24 21/40 -5/56; ...
                   16/105 -31/24  29/24 -3/40  1/168];
            G(1:2, 1:5) = A;
            G(m:m+1, m-2:end) = -rot90(A,2);
            for i = 3:m-1
               G(i, i-1:i+2) = [1/24 -9/8 9/8 -1/24];
            end

        case 6
            A = [-13016/3465  693/128  -385/128 693/320 -495/448  385/1152 -63/1408; ...
                    496/3465 -811/640   449/384 -29/960  -11/448   13/1152 -37/21120; ...
                     -8/385   179/1920 -153/128 381/320 -101/1344   1/128   -3/7040];
            G(1:3, 1:7) = A;
            G(m-1:m+1, m-4:end) = -rot90(A,2);
            for i = 4:m-2
                G(i, i-2:i+3) = [-3/640 25/384 -75/64 75/64 -25/384 3/640];
            end

        case 8
            A = [-182144/45045     6435/1024    -5005/1024  27027/5120  -32175/7168  25025/9216
            ↪  -12285/11264  3465/13312   -143/5120; ...
                   86048/675675 -131093/107520  49087/46080 10973/76800  -4597/21504  4019/27648
                   ↪  -10331/168960 2983/199680 -2621/1612800;  ...
                   -3776/225225    8707/107520 -17947/15360 29319/25600   -533/21504  -263/9216
                   ↪  903/56320  -283/66560    257/537600; ...
                      32/9009      -543/35840     265/3072  -1233/1024    8625/7168   -775/9216
                      ↪  639/56320  -15/13312       1/21504];
            G(1:4, 1:9) = A;
            G(m-2:m+1, m-6:end) = -rot90(A,2);
            for i = 5:m-3
                G(i, i-3:i+4) = [5/7168 -49/5120 245/3072 -1225/1024 1225/1024 -245/3072 49/5120
                ↪  -5/7168];
            end
    end
    G = (1/dx).*G;
end
```

Program 4: Program `grad.m`

Program 5: Program `gradient.h`

```
function L = lap(k, m, dx)
% Returns a m+2 by m+2 one-dimensional mimetic laplacian operator
%
% Parameters:
%                k : Order of accuracy
%                m : Number of cells
%               dx : Step size
% ----------------------------------------------------------------------------
% SPDX-License-Identifier: GPL-3.0-or-later
% © 2008-2024 San Diego State University Research Foundation (SDSURF).
% See LICENSE file or https://www.gnu.org/licenses/gpl-3.0.html for details.
% ----------------------------------------------------------------------------

    D = div(k, m, dx);
    G = grad(k, m, dx);

    L = D*G;
end
```

Program 6: Program `lap.m`

Program 7: Program `laplacian.h`

```matlab
function I = interpol(m, c)
% Returns a m+1 by m+2 one-dimensional interpolator of 2nd-order
%
% Parameters:
%                m : Number of cells
%                c : Left interpolation coeff.
% ----------------------------------------------------------------------------
% SPDX-License-Identifier: GPL-3.0-or-later
% © 2008-2024 San Diego State University Research Foundation (SDSURF).
% See LICENSE file or https://www.gnu.org/licenses/gpl-3.0.html for details.
% ----------------------------------------------------------------------------

    % Assertions:
    assert(m >= 4, 'm ⩾ 4');
    assert(c >= 0 && c <= 1, '0 ⩽ c ⩽ 1');

    % Dimensions of I:
    n_rows = m+1;
    n_cols = m+2;

    I = sparse(n_rows, n_cols);

    I(1, 1) = 1;
    I(end, end) = 1;

    % Average between two continuous cells
    avg = [c 1-c];

    j = 2;
    for i = 2 : n_rows - 1
        I(i, j:j+2-1) = avg;
        j = j + 1;
    end
end
```

Program 8: Program `interpol.m`

```
function D = div2D(k, m, dx, n, dy)
% Returns a two-dimensional mimetic divergence operator
%
% Parameters:
%                k : Order of accuracy
%                m : Number of cells along x-axis
%               dx : Step size along x-axis
%                n : Number of cells along y-axis
%               dy : Step size along y-axis
% ----------------------------------------------------------------------------
% SPDX-License-Identifier: GPL-3.0-or-later
% © 2008-2024 San Diego State University Research Foundation (SDSURF).
% See LICENSE file or https://www.gnu.org/licenses/gpl-3.0.html for details.
% ----------------------------------------------------------------------------

    Dx = div(k, m, dx);
    Dy = div(k, n, dy);

    Im = sparse(m + 2, m);
    In = sparse(n + 2, n);

    Im(2:(m+2)-1, :) = speye(m, m);
    In(2:(n+2)-1, :) = speye(n, n);

    Sx = kron(In, Dx);
    Sy = kron(Dy, Im);

    D = [Sx Sy];
end
```

Program 9: Program `div2D.m`

```
function G = grad2D(k, m, dx, n, dy)
% Returns a two-dimensional mimetic gradient operator
%
% Parameters:
%                k : Order of accuracy
%                m : Number of cells along x-axis
%               dx : Step size along x-axis
%                n : Number of cells along y-axis
%               dy : Step size along y-axis
% ----------------------------------------------------------------------------
% SPDX-License-Identifier: GPL-3.0-or-later
% © 2008-2024 San Diego State University Research Foundation (SDSURF).
% See LICENSE file or https://www.gnu.org/licenses/gpl-3.0.html for details.
% ----------------------------------------------------------------------------

    Gx = grad(k, m, dx);
    Gy = grad(k, n, dy);

    Im = sparse(m + 2, m);
    In = sparse(n + 2, n);

    Im(2:(m+2)-1, :) = speye(m, m);
    In(2:(n+2)-1, :) = speye(n, n);

    Sx = kron(In', Gx);
    Sy = kron(Gy, Im');

    G = [Sx; Sy];
end
```

Program 10: Program `grad2D.m`

```
function L = lap2D(k, m, dx, n, dy)
% Returns a two-dimensional mimetic laplacian operator
%
% Parameters:
%                 k : Order of accuracy
%                 m : Number of cells along x-axis
%                dx : Step size along x-axis
%                 n : Number of cells along y-axis
%                dy : Step size along y-axis
% ----------------------------------------------------------------------------
% SPDX-License-Identifier: GPL-3.0-or-later
% © 2008-2024 San Diego State University Research Foundation (SDSURF).
% See LICENSE file or https://www.gnu.org/licenses/gpl-3.0.html for details.
% ----------------------------------------------------------------------------

    D = div2D(k, m, dx, n, dy);
    G = grad2D(k, m, dx, n, dy);

    L = D*G;
end
```

Program 11: Program `lap2D.m`

```
function I = interpol2D(m, n, c1, c2)
% Returns a two-dimensional interpolator of 2nd-order
%                 m : Number of cells along x-axis
%                 n : Number of cells along y-axis
%                c1 : Left interpolation coeff.
%                c2 : Bottom interpolation coeff.
% ----------------------------------------------------------------------------
% SPDX-License-Identifier: GPL-3.0-or-later
% © 2008-2024 San Diego State University Research Foundation (SDSURF).
% See LICENSE file or https://www.gnu.org/licenses/gpl-3.0.html for details.
% ----------------------------------------------------------------------------

    Ix = interpol(m, c1);
    Iy = interpol(n, c2);

    Im = sparse(m + 2, m);
    In = sparse(n + 2, n);

    Im(2:(m+2)-1, :) = speye(m, m);
    In(2:(n+2)-1, :) = speye(n, n);

    Sx = kron(In', Ix);
    Sy = kron(Iy, Im');

    I = [Sx; Sy];
end
```

Program 12: Program `interpol2D.m`

```
function D = div3D(k, m, dx, n, dy, o, dz)
% Returns a three-dimensional mimetic divergence operator
%
% Parameters:
%                 k : Order of accuracy
%                 m : Number of cells along x-axis
%                dx : Step size along x-axis
%                 n : Number of cells along y-axis
%                dy : Step size along y-axis
%                 o : Number of cells along z-axis
%                dz : Step size along z-axis
% ----------------------------------------------------------------------------
% SPDX-License-Identifier: GPL-3.0-or-later
% © 2008-2024 San Diego State University Research Foundation (SDSURF).
% See LICENSE file or https://www.gnu.org/licenses/gpl-3.0.html for details.
% ----------------------------------------------------------------------------

    Im = sparse(m + 2, m);
    Im(2:(m + 2) - 1, :) = speye(m, m);

    Dx = div(k, m, dx);

    In = sparse(n + 2, n);
    In(2:(n + 2) - 1, :) = speye(n, n);

    Dy = div(k, n, dy);

    Io = sparse(o + 2, o);
    Io(2:(o + 2) - 1, :) = speye(o, o);

    Dz = div(k, o, dz);

    Sx = kron(kron(Io, In), Dx);
    Sy = kron(kron(Io, Dy), Im);
    Sz = kron(kron(Dz, In), Im);

    D = [Sx Sy Sz];
end
```

Program 13: Program `div3D.m`

```matlab
function G = grad3D(k, m, dx, n, dy, o, dz)
% Returns a three-dimensional mimetic gradient operator
%
% Parameters:
%                 k : Order of accuracy
%                 m : Number of cells along x-axis
%                dx : Step size along x-axis
%                 n : Number of cells along y-axis
%                dy : Step size along y-axis
%                 o : Number of cells along z-axis
%                dz : Step size along z-axis
% ----------------------------------------------------------------------------
% SPDX-License-Identifier: GPL-3.0-or-later
% © 2008-2024 San Diego State University Research Foundation (SDSURF).
% See LICENSE file or https://www.gnu.org/licenses/gpl-3.0.html for details.
% ----------------------------------------------------------------------------

    Im = sparse(m + 2, m);
    Im(2:(m + 2) - 1, :) = speye(m, m);

    Gx = grad(k, m, dx);

    In = sparse(n + 2, n);
    In(2:(n + 2) - 1, :) = speye(n, n);

    Gy = grad(k, n, dy);

    Io = sparse(o + 2, o);
    Io(2:(o + 2) - 1, :) = speye(o, o);

    Gz = grad(k, o, dz);

    Sx = kron(kron(Io', In'), Gx);
    Sy = kron(kron(Io', Gy), Im');
    Sz = kron(kron(Gz, In'), Im');

    G = [Sx; Sy; Sz];
end
```

Program 14: Program `grad3D.m`

```
function L = lap3D(k, m, dx, n, dy, o, dz)
% Returns a three-dimensional mimetic laplacian operator
%
% Parameters:
%                 k : Order of accuracy
%                 m : Number of cells along x-axis
%                dx : Step size along x-axis
%                 n : Number of cells along y-axis
%                dy : Step size along y-axis
%                 o : Number of cells along z-axis
%                dz : Step size along z-axis
% ----------------------------------------------------------------------------
% SPDX-License-Identifier: GPL-3.0-or-later
% © 2008-2024 San Diego State University Research Foundation (SDSURF).
% See LICENSE file or https://www.gnu.org/licenses/gpl-3.0.html for details.
% ----------------------------------------------------------------------------

    D = div3D(k, m, dx, n, dy, o, dz);
    G = grad3D(k, m, dx, n, dy, o, dz);

    L = D*G;
end
```

Program 15: Program `lap3D.m`

```
function I = interpol3D(m, n, o, c1, c2, c3)
% Returns a three-dimensional interpolator of 2nd-order
%                 m : Number of cells along x-axis
%                 n : Number of cells along y-axis
%                 o : Number of cells along z-axis
%                c1 : Left interpolation coeff.
%                c2 : Bottom interpolation coeff.
%                c3 : Front interpolation coeff.
% ----------------------------------------------------------------------------
% SPDX-License-Identifier: GPL-3.0-or-later
% © 2008-2024 San Diego State University Research Foundation (SDSURF).
% See LICENSE file or https://www.gnu.org/licenses/gpl-3.0.html for details.
% ----------------------------------------------------------------------------

    Im = sparse(m + 2, m);
    Im(2:(m + 2) - 1, :) = speye(m, m);

    Ix = interpol(m, c1);

    In = sparse(n + 2, n);
    In(2:(n + 2) - 1, :) = speye(n, n);

    Iy = interpol(n, c2);

    Io = sparse(o + 2, o);
    Io(2:(o + 2) - 1, :) = speye(o, o);

    Iz = interpol(o, c3);

    Sx = kron(kron(Io', In'), Ix);
    Sy = kron(kron(Io', Iy), Im');
    Sz = kron(kron(Iz, In'), Im');

    I = [Sx; Sy; Sz];
end
```

Program 16: Program `interpol3D.m`

```
function [A, b] = addBC1D(A, b, k, m, dx, dc, nc, v)
% This function assumes that the unknown u, which represents the discrete
% solution the continuous second-order 1D PDE operator
%                                    L U = f,
% with continuous boundary condition
%                               a0 U + b0 dU/dn = g,
% are given at the 1D cell centers and vertices. Furthermore, all discrete
% calculations are performed at the 1D cell centers and vertices.
%
% The function receives as input quantities associated to the discrete
% analog of the continuous problem given by the squared linear system
%                                  A u = b
% where A is the discrete analog of L and b is the discrete analog of g,
% both constructed by the user without boundary conditions.
% The function output is the modified square linear system
%                                  A u = b
% where both A and b include boundary condition information.
%
% The boundary condition is always one of the following forms:
%
% For Dirichlet set: a0 not equal zero and b0 = 0.
% For Neumann set  : a0 = 0 and b0 not equal zero.
% For Robin set    : both a0 and b0 not equal zero.
% For Periodic set : both a0 = 0 and b0 = 0.
%
% For periodic bc, it is assumed that not only u but also du/dn are the same
% in both extremes of the domain since a second-order PDE is assumed.
%
% The code assumes the following assertions:
% assert(k ⩾ 2, 'k ⩾ 2');
% assert(mod(k, 2) == 0, 'k % 2 = 0');
% assert(m ⩾ 2*k+1, ['m ⩾ ' num2str(2*k+1) ' for k = ' num2str(k)]);
%
% Parameters:
% output
%          A : Linear operator with boundary conditions added
%          b : Right hand side with boundary conditions added
%
% input
%          A : Linear operator without boundary conditions added
%          b : Right hand side without boundary conditions added
%          k : Order of accuracy
%          m : Number of cells
%         dx : Step size
%         dc : a0 (2×1 vector for left and right vertices, resp.)
%         nc : b0 (2×1 vector for left and right vertices, resp.)
%          v : g (2×1 vector for left and right vertices, resp.)
% ----------------------------------------------------------------------------
% SPDX-License-Identifier: GPL-3.0-or-later
% © 2008-2024 San Diego State University Research Foundation (SDSURF).
% See LICENSE file or https://www.gnu.org/licenses/gpl-3.0.html for details.
% ----------------------------------------------------------------------------

    % verify bc sizes and square linear system
    assert(all(size(dc) == [2 1]), 'dc is a 2×1 vector');
    assert(all(size(nc) == [2 1]), 'nc is a 2×1 vector');
    assert(all(size(v) == [2 1]), 'v is a 2×1 vector');
    assert(size(A,1) == size(A,2), 'A is a square matrix');
    assert(size(A,2) == numel(b), 'b size = A columns');

    % remove first and last rows of A
    vec = sparse(2,1);
    vec(1) = 1;
    vec(2) = size(A,1);

    [rows,cols,s] = find(A(vec,:));
    A = A - sparse(vec(rows), cols, s, size(A,1), size(A,2));

    % remove first and last coefficients of right-hand-side vector b
    b(vec) = 0;

    [Abcl,Abcr] = addBC1Dlhs(k, m, dx, dc, nc);
    A = A + Abcl + Abcr;
    b = addBC1Drhs(b, dc, nc, v, vec);
end
```

Program 17: Program addBC1D.m

3. Numerical Methods for ODEs

3.1. Numerical Solution of Initial Value Problems

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

Consider the Initial Value Problem

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

3.1.1. 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 $\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$.

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

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

Integramos y obtenemos la solución general.

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

Ahora, apliquemos las condiciones de frontera Robin.

$$\begin{cases} 0 = u(0) - \left(\frac{\mathrm{d}u}{\mathrm{d}x}\right)_{x=0} = e^0 + C_1(0) + C_2 - (e^0 + C_1) = C_2 - C_1. \\ 2e = u(1) + \left(\frac{\mathrm{d}u}{\mathrm{d}x}\right)_{x=1} = e^1 + C_1(1) + C_2 + (e^1 + C_1) = 2e + 2C_1 + C_2. \end{cases} \tag{3.1}$$

El sistema (3.1) tiene como solución $C_1 = C_2 = 0$. $\therefore$ la solución de (5.1) es $u(x) = e^x$.

- `https://docs.octave.org/v9.4.0/Ranges.html`
- `https://docs.octave.org/v9.4.0/Solvers.html#XREFfzero`

```
#!/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
```

Program 18: Program `backward_euler.m`

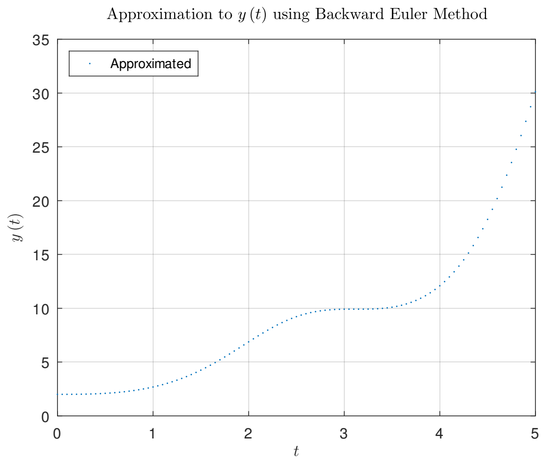


Figure 3.1.: Numerical solution by the Backward Euler Method.

- `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`

3.1.2. Explicit Runge-Kutta 2

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

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
```

Program 19: Program `RK2.m`

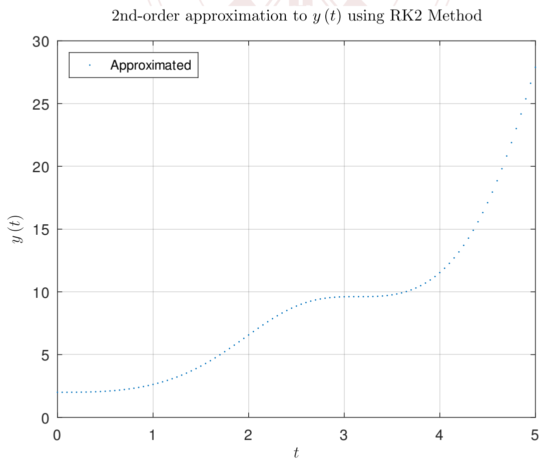


Figure 3.2.: Numerical solution by the Runge-Kutta 2 Method.

3.1.3. 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 $\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{4}{6} & \frac{4}{6} & \frac{1}{6} \end{array}$.

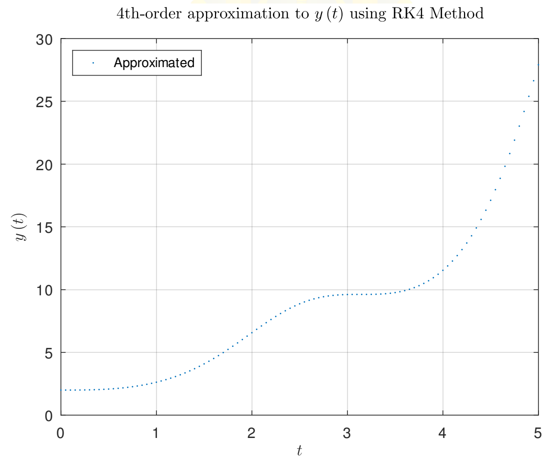


Figure 3.3.: Numerical solution by the Runge-Kutta 4 Method.

3.1.4. Verlett

3.2. Numerical Solutions of Boundary Value Problems

```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; }
```

```
#!/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
```

Program 21: Program `RK4.m`

4. Numerical Methods for PDEs

4.1. von Neumann Stability Criterion

4.2. Stability and Convergence Analysis

Part II.

Numerical Exercises

5. Getting started with MOLE

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. Compiling and running the first code

5.1.1. Create the staggered grid

5.2. Transport 1D

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

```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);
  }

```

Program 22: Program `hyperbolic1Dupwind.cpp`

```
#!/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')
```

Program 23: Program `elliptic1D.m`

5.3. Poisson 1D

$$\begin{cases} \dfrac{\mathrm{d}^2 u}{\mathrm{d}x^2} = e^x, \text{ para } x \in [0,1]\,. \\ \quad 0 = u\,(0) - \left(\dfrac{\mathrm{d}u}{\mathrm{d}x}\right)_{x=0}. \\ \quad 2e = u\,(1) + \left(\dfrac{\mathrm{d}u}{\mathrm{d}x}\right)_{x=1}. \end{cases} \tag{5.1}$$

- 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 (5.1).

- 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 $L \in \mathbb{R}^{(m+2)\times(m+2)}$.

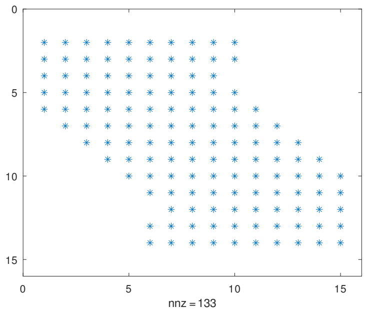
- En la línea 22, con la función `figure` desactivamos que se muestre la figura en la pantalla, preferimos solamente guardar la gráfica.

- 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

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



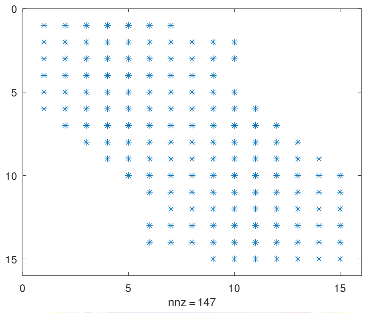


Figure 5.1.: Izquierda: Representación dispersa de L hasta la línea 21. La primera y la última fila son vectores de ceros. Derecha: Representación dispersa de L hasta la línea 29. La matriz $L \in \mathbb{R}^{15\times 15}$.

`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 $U \leftarrow \text{solve}\,(A, F)$;
```

- 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`.

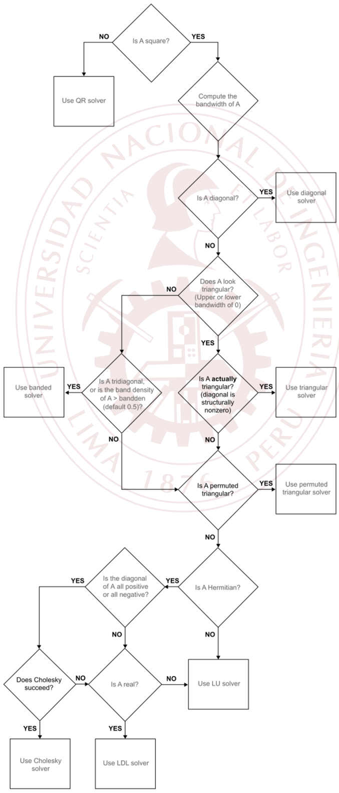

Is A square?
NO
YES
Use QR solver
Compute the bandwidth of A
Is A diagonal?
YES
Use diagonal solver
NO
Does A look triangular? (Upper or lower bandwidth of 0)
NO
YES
Use banded solver
YES
Is A tridiagonal, or is the band density of A > bandden (default 0.5)?
Is A actually triangular? (diagonal is structurally 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 5.2.: 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 23

En primer lugar, mostramos la gráfica a escala 1:1 de la solución exacta y de la solución mimética obtenida en el Program 23.

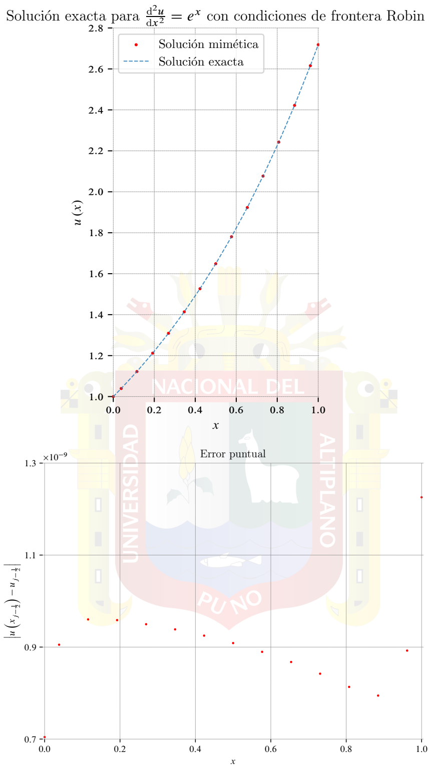


Figure 5.3.: Izquierda: Solución de (5.1) usando `k=6` y `m=2k+1=13`. Derecha: Error en la malla escalonada $\left\{0, \ldots, x_{j-\frac{1}{2}} \ldots, 1\right\}$

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

5.4. The 1D Diffusion Equation

Given the one-dimensional heat equation

$$\begin{cases} \frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}, & (x,t) \in (a,b) \times (0,\infty). \\ u(x,0) = f(x), & x \in [a,b]. \\ u(a,t) = \alpha(t), & t \in (0,\infty), \\ u(b,t) = \beta(t), & t \in (0,\infty), \end{cases} \tag{5.2}$$

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:

$$\begin{cases} \frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}, & (x,t) \in (0,1) \times (0,1). \\ u(x,0) = 0, & x \in [0,1]. \\ u(0,t) = 100., & t \in (0,1), \\ u(1,t) = 100., & t \in (0,1), \end{cases} \tag{5.3}$$

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:

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2} \tag{5.4}$$

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

$$\frac{u_i^{n+1} - u_i^n}{dt} = \kappa L u_i^n \tag{5.5}$$

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

$$u_i^{n+1} = u_i^n + \kappa dt L u_i^n \tag{5.6}$$

Factoring u_i^n we have:

$$u_i^{n+1} = (I + \kappa dt L) u_i^n \tag{5.7}$$

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

$$u_i^{n+1} = L * u_i^n \tag{5.8}$$

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:

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2} \tag{5.9}$$

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

$$\frac{u_i^{n+1} - u_i^n}{dt} = \kappa L u_i^{n+1} \tag{5.10}$$

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

$$u_i^{n+1}(I - \kappa dt L) = u_i^n \tag{5.11}$$

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 $U = dL \setminus U$.

5.5. Convection-diffusion

Given the three-dimensional convection diffusion equation

$$\begin{cases} \frac{\partial u}{\partial t} + \nabla \cdot (\mathbf{v}u) = \nabla \cdot (D\nabla u) + R, & \mathbf{x} \in \Omega, t > 0 \\ u(\mathbf{x}, 0) = \bar{\alpha}, & \mathbf{x} \in \Omega. \\ u(\mathbf{x}, t) = \bar{\alpha}_0, & t > 0, \mathbf{x} \in \partial\Omega \end{cases} \tag{5.12}$$

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 $dz = 1$.

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

```
#!/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
```

Program 24: Program `parabolic1D.m`

```
#!/usr/bin/env -S octave -qf
% convection_diffusion.m | Solves convection-diffusion equation using MOLE

clc
close all
format short

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

% Mimetic operator's parameters
k = 2;
m = 101;
n = 51;
o = 101;

% Domain's dimensions
a = 0;
b = 101;
c = 0;
d = 51;
e = 0;
f = 101;

% Spatial step sizes
dx = (b - a) / m;
dy = (d - c) / n;
dz = (f - e) / o;

% Mimetic operators
D = div3D(k, m, dx, n, dy, o, dz);
G = grad3D(k, m, dx, n, dy, o, dz);
I = interpol3D(m, n, o, 1, 1, 1);
```

Program 25: Program `convection_diffusion.m`

```

% 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);

```

Program 26: Program `convection_diffusion.m`

Part III.

Tests Cases in MOLE

6. Divergence

6.1. Introduction and Objectives

6.2. Background and Goals

MOLE uses GoogleTest framework.

6.3. Test 1

Program 27: Program `test1.cpp`

Program 28: Program `test1.m`

6.4. Test 2

Program 29: Program `test2.cpp`

Program 30: Program `test2.m`

6.5. Test 3

Program 31: Program `test3.cpp`

Program 32: Program `test3.m`

Program 33: Program `test4.cpp`

Program 34: Program `test4.m`

6.6. Test 4

6.7. Test 5

Program 35: Program `test5.cpp`

Program 36: Program `test5.m`

A. Installing MOLE software on GNU/Linux

To work through this tutorial requires to have a working installation of MOLE. It relies on `armadillo` [12], 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 sucessfully tested by us.

A.1. MOLE on Arch Linux

This distribution is supported by a proactive group of developers, package maintainers and support staff that try to provides the latest stable software releases. The steps are outlined in the Program 37.

```bash
#!/bin/bash

# [1 / 3] Update the system and install developer tools
sudo pacman --needed --noconfirm -Syu base-devel git
# [1.5 / 3] 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

# [2 / 3] Install armadillo from https://aur.archlinux.org/armadillo.git
git clone https://aur.archlinux.org/armadillo.git
pushd armadillo
makepkg -s --noconfirm
popd

# [3 / 3] 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
```

Program 37: Steps for a system-wide installation both C++ and Octave MOLE library vía `installerarchlinux.sh`.

Even if you are using Windows, the Docker Desktop WSL 2 backend is ideal for using MOLE via Program 38 or installing Arch Linux on WSL 2 and following the Program 37.

A.2. MOLE on 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

```
#!/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 38: Pull container based on Arch Linux with set up MOLE library vía `docker.sh`.

the Program 39.

```
#!/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 libgtest-dev
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
```

Program 39: Steps for a system-wide installation both C++ and Octave MOLE library vía `installerubuntu.sh`.

B. MOLE Documentation

B.1. Mimetic Operator's Library Enhanced Reference

We split the MOLE documentation in three categories:

General MOLE documentation It contains general information and examples.

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

Octave MOLE documentation It contains API GNU/Octave Reference.

B.2. Programming languages documentation

C++ docs
```
#include <iostream>
#include <cmath>
#include <vector>.
```

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

MATLAB docs .

B.3. Linear Algebra software documentation

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. Sparse Linear Algebra software examples

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

C.5. Fortran Sparse

C.6. Sparse Linear Algebra in Rust

C.7. PETSc sparse matrices in C

```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 40: Program `1.cc`

D. Method of characteristics

[5, 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.

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

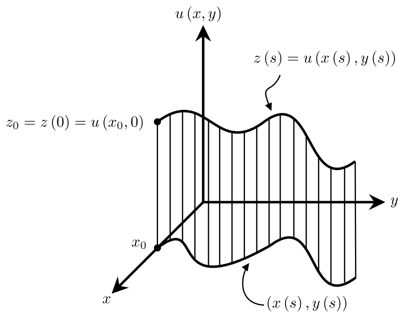


Figure D.1.: The solution u is described by the surface defined by $z = u(x,y)$. From any point x_0 on the x-axis, there is a curve $(x(s), y(s))$ in the xy-plane, upon which wa can calculate the solution $z = u(x(s), y(s))$. Knowing only the structure of the PDE, x_0 and z_0 we can solve ODEs to find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

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

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. Method of separation of variables

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

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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`.

[12] Conrad Sanderson and Ryan Curtin. *Armadillo: An Efficient Framework for Numerical Linear Algebra*. 2025. DOI: `10.48550/arXiv.2502.03000`. arXiv: `2502.03000 [cs.MS]`.

[13] 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`.

[14] 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`.