MOLE

MOLE
User's Manual

(MATLAB version)

8/8/2017

Dr. Jose Castillo : jcastillo@mail.sdsu.edu **Johnny Corbino : johnnycorbino@gmail.com**

Getting the library…

You can download MOLE from its main repository on GitHub:

https://github.com/jcorbino/mole

or from the MathWorks' File Exchange website:

https://www.mathworks.com/matlabcentral/fileexchange

You will find out two versions of MOLE under the same repository—MATLAB and C++

Licensing

MOLE is distributed under a “dual-license” model. GPLv3 for academic and nonprofit purposes, and copyright for commercial intentions:

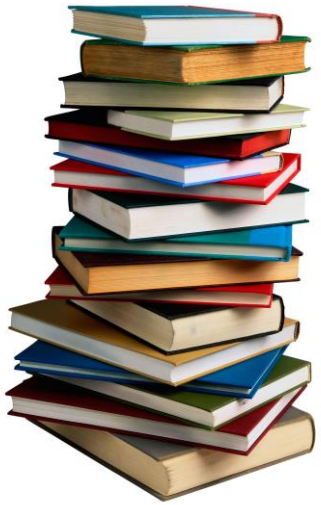
https://github.com/jcorbino/mole/LICENSE.txt

If you are planning to use or modify MOLE for commercial purposes, feel free to contact:

tmartindale@sdsu.edu or kwelch@foundation.sdsu.edu

On the mathematics

The mimetic discrete operators implemented in MOLE are based on the work of [*Corbino and Castillo 2017, to be published*]. They are a substantial improvement over the operators introduced in [*Castillo and Grone 2003*]. These new operators have no free parameters, and in the worst case they deliver the same accuracy as the ones from 2003.



Installing the library…

This document covers the installation and usage of MOLE for MATLAB. Once you download the library, you need to add the path to it on your MATLAB script:

```
addpath('path_to_mole')
```

After this, you will be able to call any of the 29 functions provided by MOLE to construct mimetic operators.

Getting the operators…

Functions syntax is consistent throughout the library. In addition, you can see a brief explanation of each function by typing the following command:

```
help 'function_name'
```

For instance,

```
>> help div
  Returns a m+2 by m+1 one-dimensional mimetic divergence operator

  Parameters:
                 k : Order of accuracy
                 m : Number of cells
                dx : Step size
```

Getting the operators…

Another example,

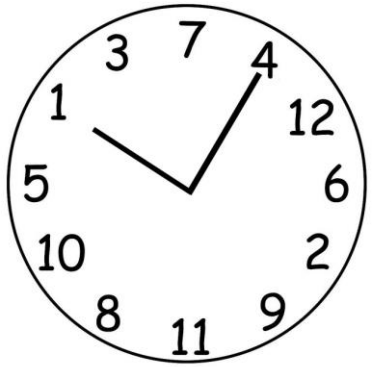
```
>> help grad2D
  Returns a two-dimensional mimetic gradient operator

  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
```

All functions in MOLE return a sparse matrix representation of the requested operator!

Getting the operators…

Try this: Obtain a 2nd-order 1D mimetic Laplacian and plot its sparsity pattern!



Getting the operators…

Try this: Obtain a 2nd-order 1D mimetic Laplacian and plot its sparsity pattern!

There are two ways,

1) Calling: `div(k, m, dx)*grad(k, m, dx);`

or

2) Calling: `lap(k, m, dx);`

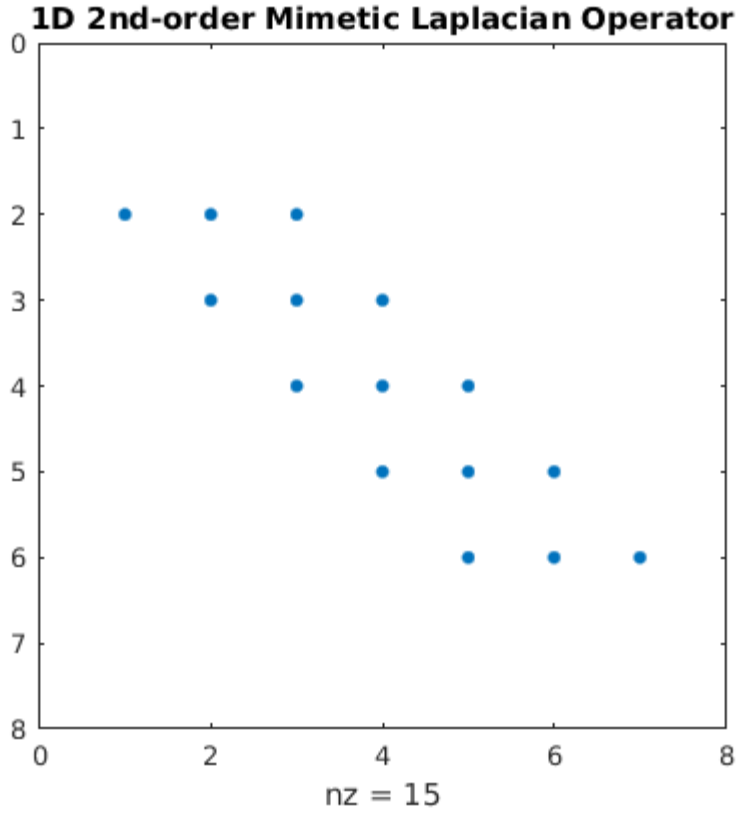
Order of accuracy

Number of cells

Cell's width

Getting the operators…

`spy(lap(2, 5, 1))` yields→

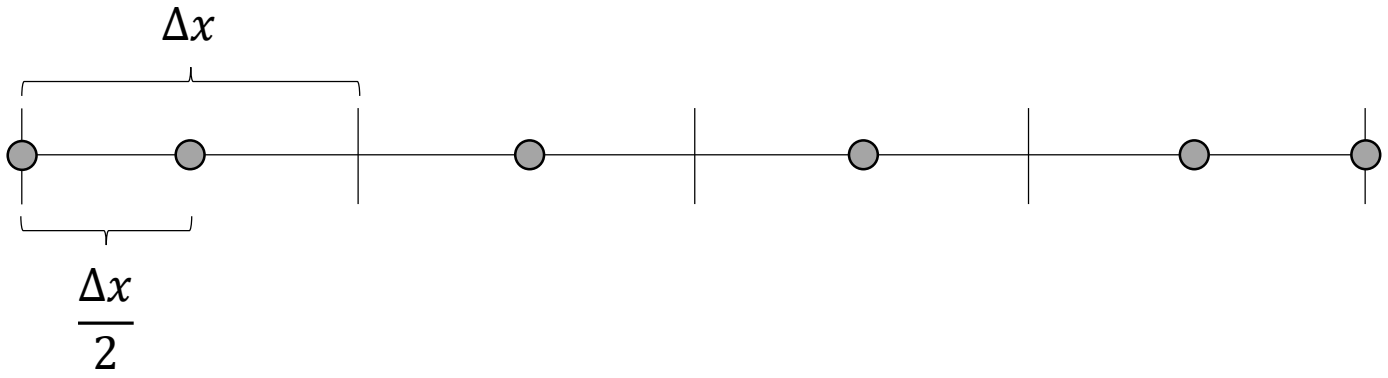


Try this with other functions to get familiar with the "form" of the operators

a 7x7 sparse matrix with 15 nonzero elements

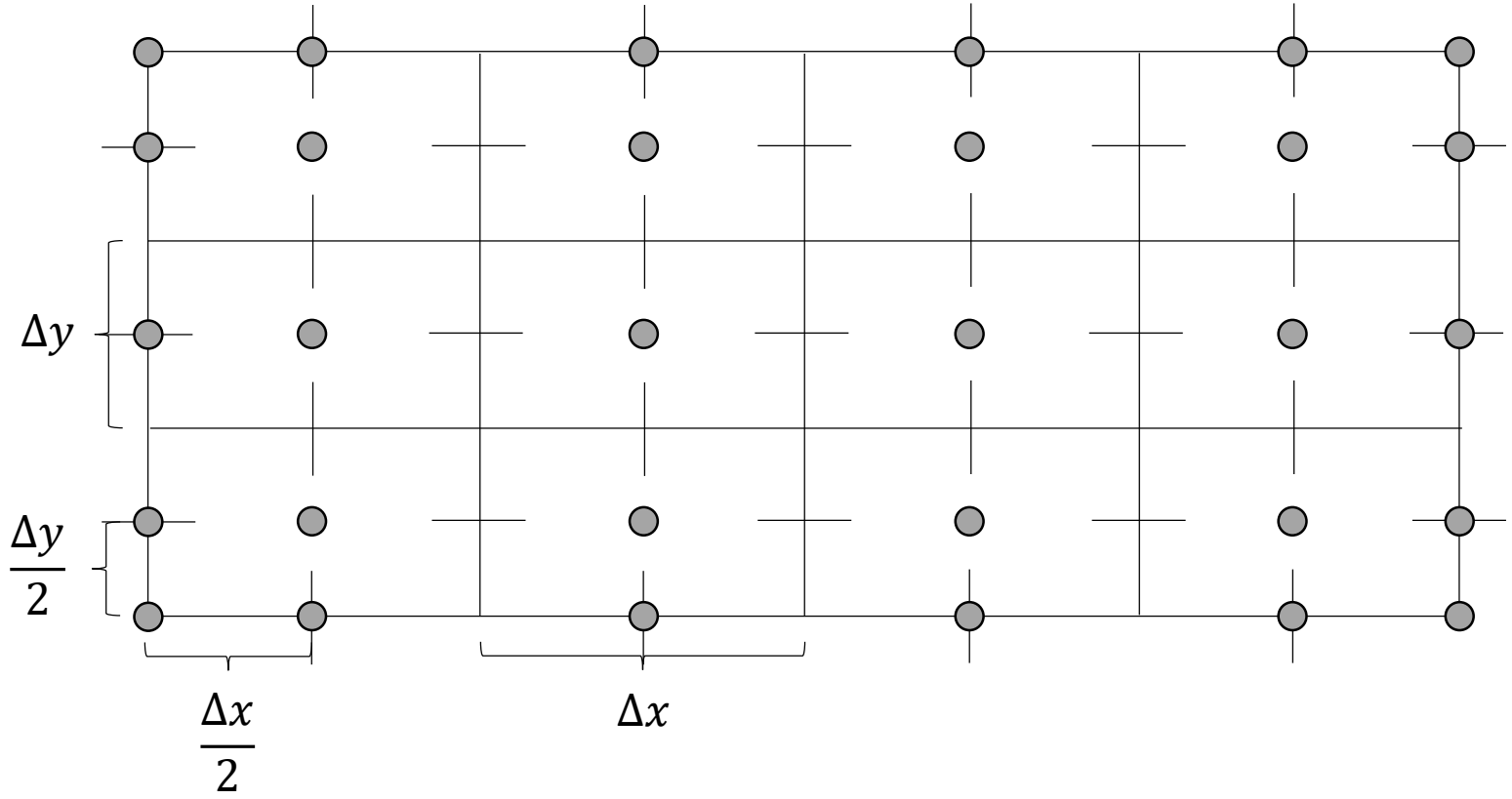
The grid (1D)

It is important to remember that Mimetic Operators act on Staggered Grids.



Scalar quantities | Vector quantities

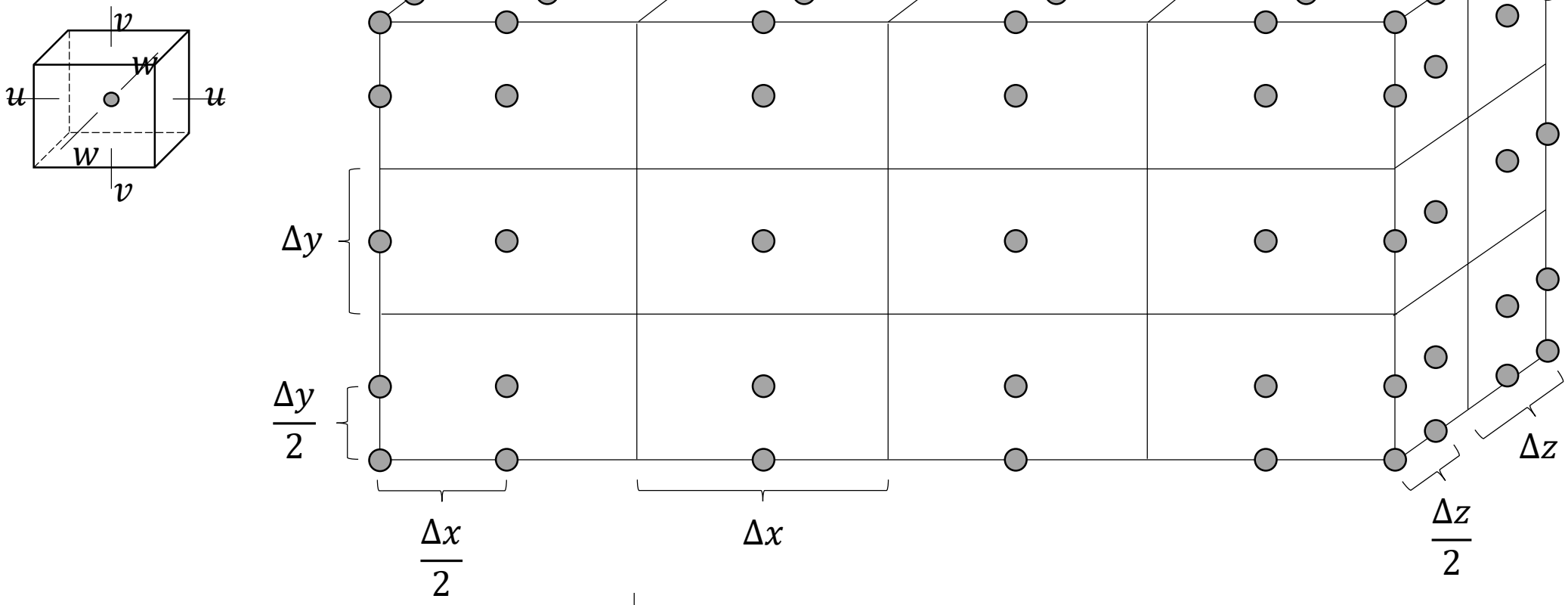
The grid (2D)



Scalar quantities

Vector quantities

The grid (3D)



Scalar quantities

Vector quantities

Generating a 1D staggered grid

There are several ways to accomplish this. What is important is to remember that we need to store the coordinates for two different quantities.

Suppose:

```
west = 0; east = 1; m = 10; dx = (east-west)/m;
xgridSca = [west west+dx/2 : dx : east-dx/2 east];
```

This grid holds the coordinates of all scalar quantities ○ (cell centers + boundaries).

Vector quantities | coordinates are just:

```
xgridVec = west : dx : east;
```

Generating a 1D staggered grid

The previous commands just create a couple of one-dimensional arrays with the coordinates of each field (scalar and vectorial). Now we need a couple of arrays to hold the actual values of these fields.

NOTE: Depending on the problem, you may need or not to explicitly create such arrays. It is often a good practice to preallocate the memory for better performance:

```
scalarField = zeros(numel(xgridSca), 1);
```

and,

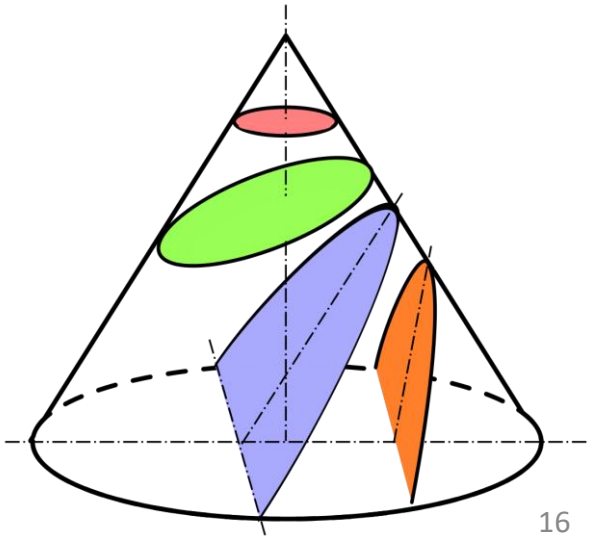
```
vectorField = zeros(numel(xgridVec), 1);
```

Using the operators…

Inside the "`examples`" folder you will find several MATLAB scripts that use MOLE to solve well known partial differential equations.

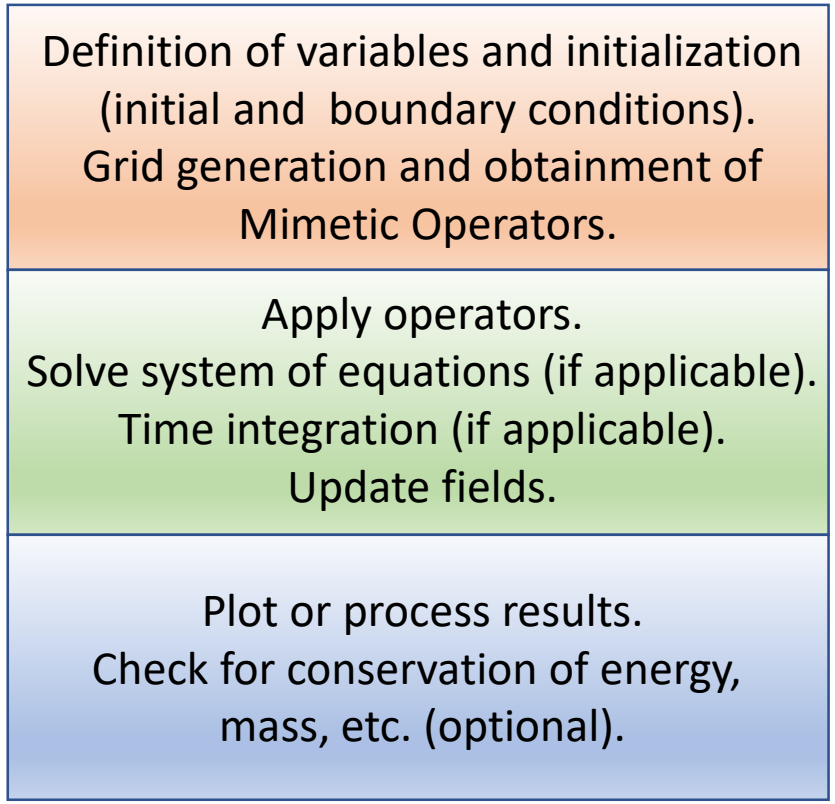
Our selection includes steady-state and time-dependent problems.

- Burger's eq.
- Richards' eq. (highly nonlinear, mixed form)
- Wave eq. (with symplectic schemes)
- Heat eq. (explicitly and implicitly)
- Etc.



Using the operators…

Each script in the "`examples`" folder is adequately commented. You may notice that all programs have the same taxonomy:



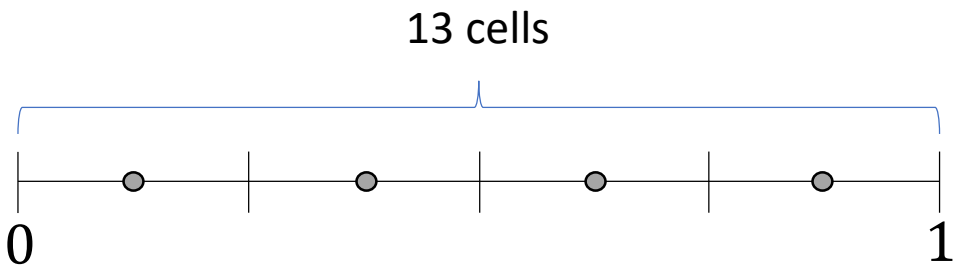
Code snippet

Lets have a look at the script named “elliptic1D.m”

```
% Solves the 1D Poisson's equation with Robin boundary conditions
clc; close all
addpath('../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 size
```



Code snippet

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

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

$$\nabla^2 u = f$$

f

$au(0) - bu'(0) = 0$

$au(1) + bu'(1) = 2e$

Robin boundary condition

Code snippet

```
% Get 1D 6th-order mimetic laplacian operator from MOLE
L = lap(k, m, dx);

% Impose Robin BC on laplacian operator
a = 1;  % The ‘a’ coefficient in Robin BC
b = 1;  % The ‘b’ coefficient in Robin BC
L = L + robinBC(k, m, dx, a, b);  % robinBC also provided by MOLE
```

$$\nabla^2 u = f$$

Code snippet

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

$\nabla^2 u = f$

```
plot(xgrid, U, 'o-')
title('Poisson''s equation with Robin BC')
xlabel('x')
ylabel('u(x)')
% End of elliptic1D.m
```

As you can see, the previous script followed the structure outlined on page 16.

It is really easy to solve 1D, 2D and 3D problems using MOLE!!!

Remember: MOLE is also available in C++

Need extra assistance?

If you need assistance or want to report a problem with any of the functions, scripts or classes that constitute MOLE, please contact us at:

- jcorbino@mail.sdsu.edu
- angelboada2@gmail.com
- billbyrd320@gmail.com

We will gladly assist you ☺