Quick Guide to MOLE's C++ Version

Johnny Corbino

May 31, 2024

Contents

1 Introduction

MOLE (Mimetic Operators Library Enhanced) is an open-source C++ library designed to implement high-order mimetic operators for solving partial differential equations (PDEs). This guide provides a quick introduction to using MOLE in C++.

2 Installation

2.1 Dependencies

Before installing MOLE, ensure the following dependencies are installed:

- Armadillo: `http://arma.sourceforge.net`
- SuperLU: `https://portal.nersc.gov/project/sparse/superlu`
- OpenBLAS: `https://www.openblas.net`

2.2 Downloading and Building MOLE

To download and build MOLE:

```
wget https://sourceforge.net/projects/arma/files/armadillo-12.6.6.tar.xz
tar xvf armadillo-12.6.6.tar.xz
cd armadillo-12.6.6
./configure
make
```

Set up Armadillo with SuperLU and OpenMP support:

```
#define ARMA_USE_SUPERLU
#define ARMA_USE_OPENMP
```

Build MOLE:

```
cd mole/
make
```

3 Getting Started

3.1 Including MOLE in Your Project

To include MOLE in your project, add the following include directive:

```
#include "mole.h"
```

Link against the MOLE library during compilation:

```
g++ your_code.cpp -o your_program -I/path/to/mole -L/path/to/mole -lmole
```

3.2 Basic Usage Example

Here is a basic example demonstrating the setup and solution of a simple PDE using MOLE:

```
#include <iostream>
#include "mole.h"

int main() {
    /* Your MOLE code here:
       Step 1: Set up problem's parameters
       Step 2: Construct grids (space discretization)
       Step 3: Obtain the mimetic operators
       Step 4: Enforce boundary conditions and initial conditions
       Step 5: Construct time discretization scheme (if any) involving ope
       Step 6: Apply operators to their respective fields (column vectors 
               scalar or vector fields)
       Step 7: Show results */
    return 0;
}
```

4 Core Concepts

4.1 Staggered Grids

Mimetic operators are defined over staggered grids. Unlike regular (or collocated) grids, where all variables are stored at the same grid points, staggered grids distribute variables across different, interlaced grid points. For instance, in a one-dimensional domain, if the primary variable u is stored at grid points x_i, its derivative or related quantities might be stored at midpoints $x_{i+1/2}$.

4.2 Mimetic Operators

Mimetic operators are derived by constructing discrete analogs of the continuum differential operators ∇, $\nabla\cdot$, $\nabla\times$, and ∇^2. Since most continuum models are described in terms of these operators, schemes based on mimetic operators have recently gained a lot of traction in the context of numerical PDEs. MOLE implements the following operators:

- Gradient: `grad(k, m, dx)`
- Divergence: `div(k, m, dx)`
- Laplacian: `lap(k, m, dx)`
- Boundary operator: `robinBC(k, m, dx)`
- Bilaplacian: $\nabla^2 * \nabla^2$
- Curl: `curl2D(k, m, dx, n, dy)`

where ‘k’ is the desired order of accuracy, ‘m’ is the number of cell-centers, and ‘dx’ is the step-size. Each of the aforementioned operators is also available for 2D and 3D problems, as well as for nonuniform and curvilinear staggered grids.

5 Examples

5.1 Example 1: Solving Transport Problem in 1D

This example uses MOLE to solve the 1D advection-reaction-dispersion equation. The equation is given by:

$$\frac{\partial C}{\partial t} + v\frac{\partial C}{\partial x} = D\frac{\partial^2 C}{\partial x^2}$$

where C is the concentration, v is the pore-water flow velocity, and D is the dispersion coefficient.

```
#include "mole.h"
#include <iostream>

int main() {

  int k = 2;                // Operators' order of accuracy
  Real a = 0;               // Left boundary
  Real b = 130;             // Right boundary
  int m = 26;               // Number of cells
  Real dx = (b - a) / m;    // Cell's width [m]
  Real t = 4;               // Simulation time [years]
  int iter = 208;           // Number of iterations
  Real dt = t / iter;       // Time step
  Real dis = 5;             // Dispersivity [m]
  Real vel = 15;            // Pore-water flow velocity [m/year]
  Real R = 2.5;             // Retardation (Cl^- = 1, Na^+ = 2.5)
  Real C0 = 1;              // Displacing solution concentration [mmol/kgw]

  // Get 1D mimetic operators
  Gradient G(k, m, dx);
  Divergence D(k, m, dx);
  Interpol I(m, 0.5);

  // Allocate fields
  vec C(m + 2); // Scalar field (concentrations)
  vec V(m + 1); // Vector field (velocities)

  // Impose initial conditions
  C(0) = C0;
  V.fill(vel);

  // Hydrodynamic dispersion coefficient [m^2/year]
  dis *= vel; // 75
  // dt = dt/R (retardation)
  dt /= R;

  // Time integration loop
  for (int i = 0; i <= iter; i++) {

    // First-order forward-time scheme
    C += dt * (D * (dis * (G * C)) - D * (V % (I * C)));

    // Right boundary condition (reflection)
    C(m + 1) = C(m);
  }

  // Spit out the new concentrations!
  cout << C;

  return 0;
}
```

5.2 Example 2: Poisson Equation

This example solves the 1D Poisson equation:

$$-\frac{d^2u}{dx^2} = f(x)$$

with Robin boundary conditions:

$$au + b\frac{du}{dx} = g$$

The equation is discretized using mimetic operators.

```
#include "mole.h"
#include <iostream>

int main() {

  int k = 6;                 // Operators' order of accuracy
  Real a = 0;                // Left boundary
  Real b = 1;                // Right boundary
  int m = 2 * k + 1;         // Number of cells
  Real dx = (b - a) / m;     // Step size

  // Get mimetic operators
  Laplacian L(k, m, dx);
  Real d = 1; // Dirichlet coefficient
  Real n = 1; // Neumann coefficient
  RobinBC BC(k, m, dx, d, n);
  L = L + BC;

  // 1D Staggered grid
  vec grid(m + 2);
  grid(0) = a;
  grid(1) = a + dx / 2.0;
  int i;
  for (i = 2; i <= m; i++) {
    grid(i) = grid(i - 1) + dx;
  }
  grid(i) = b;

  // Build RHS for system of equations
  vec rhs(m + 2);
  rhs = exp(grid); // rhs(0) = 1
  rhs(0) = 0;
  rhs(m + 1) = 2 * exp(1); // rhs(1) = 2e

  // Solve the system of linear equations
#ifdef EIGEN
  // Use Eigen only if SuperLU (faster) is not available
  vec sol = Utils::spsolve_eigen(L, rhs);
#else
  vec sol = spsolve(L, rhs); // Will use SuperLU
#endif

  // Print out the solution
  cout << sol;

  return 0;
}
```

5.3 Example 3: Schrodinger Equation

This example solves the 1D time-independent Schrodinger equation:

$$H\psi = E\psi$$

where H is the Hamiltonian operator, ψ is the wave function, and E is the energy. The Hamiltonian includes the kinetic energy term represented by the Laplacian and a potential energy term.

```
#include "mole.h"
#include <algorithm>
#include <iostream>

int main() {

  int k = 4;    // Operators' order of accuracy
  Real a = -5; // Left boundary
  Real b = 5;  // Right boundary
  int m = 500; // Number of cells
  vec grid = linspace(a, b, m);
  Real dx = grid(1) - grid(0); // Step size

  // Get mimetic Laplacian operator
  Laplacian L(k, m - 2, dx);

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

  sp_mat V(m, m); // Potential energy operator
 V.diag(0) = grid;

  // Hamiltonian
  sp_mat H = -0.5 * (sp_mat)L + V;

  cx_vec eigval;
  eig_gen(eigval, (mat)H); // Compute eigenvalues

  eigval = sort(eigval);

  cout << "Energy levels = [ ";
  for (int i = 0; i < 4; ++i)
    cout << real(eigval(i) / eigval(0)) << ' ';
  cout << "]\n";

  return 0;
}
```

5.4 Community Support

We warmly welcome contributions to MOLE, whether they involve adding new functionalities, providing examples, addressing existing issues, reporting bugs, or requesting new features. Please refer to our Contribution Guidelines for more details.

6 References

- Corbino and Castillo, 2020: `https://doi.org/10.1016/j.cam.2019.06.042`
- Castillo and Grone, 2003: `https://doi.org/10.1137/S0895479801398025`