MatIB's User Guide

Brittany D. Froese and Jeffrey Wiens

July 2013

1 Introduction

The immersed boundary (IB) method is a mathematical framework for studying fluid-structure interaction initially developed by Charles Peskin to study blood flow through a heart valve [2]. Since its conception, the IB method has found a wide variety of applications in biofluid mechanics and has evolved into a generalized framework [3] for studying fluid-structure interaction problems.

MatIB is a simple Matlab implementation of the IB method that allows students and researchers to solve simple fluid-structure interaction problems with minimal overhead. The algorithm employed in MatIB is described in Peskin's review paper [3] and is an adaptation of the Lai-Peskin algorithm [1]. For clarity, we have limited the scope of our implementation and this code should therefore not be thought as a generalized IB toolkit. Instead, MatIB's codebase acts as a foundation for further experimentation and extension.

MatIB is released under the MIT Open Source License[1] and is free to use for any purpose. However, any resulting publications should cite this MatIB User Guide.

2 Governing Equations

MatIB is designed to simulate the interaction between a two-dimensional Newtonian, incompressible fluid and a one-dimensional, closed, elastic membrane. The fluid is defined on a periodic box $\Omega = [0, H_x] \times [0, H_y]$ using the Eulerian coordinates $\boldsymbol{x} = (x, y)$. Immersed in the fluid contains a neutrally-buoyant membrane $\Gamma \subset \Omega$ defined using on moving Lagrangian coordinates which is parameterized by $s \in [0, 1]$.

The IB method is mathematically defined by a set of differential equations involving a mixture of Eulerian and Lagrangian variables. The fluid is modelled using the incompressible Navier-Stokes equations

$$\rho \left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} \right) + \nabla p = \mu \nabla^2 \boldsymbol{u} + \boldsymbol{f}, \tag{1}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{2}$$

[1]http://www.opensource.org/licenses/mit-license.php

where

- $\boldsymbol{u}(\boldsymbol{x},t) = (u(\boldsymbol{x},t), v(\boldsymbol{x},t))$ and $p(\boldsymbol{x},t)$ are the fluid velocity and pressure at location $\boldsymbol{x}$ and time t,
- ρ and μ are the fluid density and dynamic viscosity (both constants), and
- $\boldsymbol{f}(\boldsymbol{x},t)$ is the external body force.

The immersed boundary is coupled to the fluid through the external body force $\boldsymbol{f}$ allowing the membrane to exert a force onto the fluid. Specifically, the external body force is defined by

$$\boldsymbol{f}(\boldsymbol{x},t) = \int_{\Gamma} \boldsymbol{F}(s,t)\,\delta(\boldsymbol{x} - \boldsymbol{X}(s,t))\,ds, \tag{3}$$

where $\boldsymbol{X}(s,t) = (X(s,t), Y(s,t))$ is a parametric curve representing the IB configuration and $\boldsymbol{F}(s,t)$ is the elastic force density. The delta function $\delta(\boldsymbol{x}) = \delta(x)\delta(y)$ is a Cartesian product of one-dimensional Dirac delta functions, and acts to "spread" the IB force from Γ onto adjacent fluid particles. In general, the force density $\boldsymbol{F}$ is a functional of the current IB configuration

$$\boldsymbol{F}(s,t) = \boldsymbol{\mathcal{F}}\left[\boldsymbol{X}(s,t)\right]. \tag{4}$$

In MatIB, we define the force density as

$$\boldsymbol{\mathcal{F}}[\boldsymbol{X}(s,t)] = \sigma \frac{\partial}{\partial s}\left(\frac{\partial \boldsymbol{X}}{\partial s}\left(1 - \frac{L}{|\frac{\partial \boldsymbol{X}}{\partial s}|}\right)\right) \tag{5}$$

which corresponds to an elastic fiber having a "spring constant" σ and an equilibrium state where the elastic strain $|\partial \boldsymbol{X}/\partial s| \equiv L$.

The final equation needed to close the system is an evolution equation for the immersed boundary, which comes from the simple requirement that Γ must move at the local fluid velocity:

$$\frac{\partial \boldsymbol{X}(s,t)}{\partial t} = \boldsymbol{u}(\boldsymbol{X}(s,t),t) = \int_{\Omega} \boldsymbol{u}(\boldsymbol{x},t)\,\delta(\boldsymbol{x} - \boldsymbol{X}(s,t))\,d\boldsymbol{x}. \tag{6}$$

This last equation is nothing other than the no-slip condition which can be written as a delta function convolution. Periodic boundary conditions are imposed on both the fluid and the immersed structure and appropriate initial values are prescribed for the fluid velocity $\boldsymbol{u}(\boldsymbol{x},0)$ and IB position $\boldsymbol{X}(s,0)$. Further details on the mathematical formulation of the immersed boundary problem and its extensions can be found in Peskin's review paper [3].

3 Tutorial

In the following section, we give a brief overview of MatIB's functionality. The codebase is split between three folders: `./solver/`, `./examples/`, and `./unit tests/`. The main IB solver is contained in the `./solver/Peskin-TwoStep/` folder which is an implementation of the Lai and Peskin fractional-step scheme [3]. Here, the solver is modularized into three major components:

- *Update Membrane Position:* The fluid velocity is is interpolated onto the immersed boundary and is evolved forward in time.
- *Calculate Force:* The force density exerted by the immersed boundary is calculated and is spread onto nearby fluid grid points.
- *Fluid Solve:* The fluid variables are evolved in time using the external force computed in the calculate force step.

Each of these components can be easily extended to handle a wide variety of problems. Lastly, the `./examples/` and `./unit tests/` folders contains example problems and units tests to validate the solver.

3.1 Example: Oscillations of an elliptical membrane

In the folder `./examples/`, there contains several example problems demonstrating the use of MatIB. In this section, we will consider the example problem found in

`./examples/Elliptical Membrane/EllipticalMembrane.m`,

which simulates the oscillations of a pressurized membrane with an initially elliptical shape. Here, the initial configuration of the membrane is the ellipse parameterized by

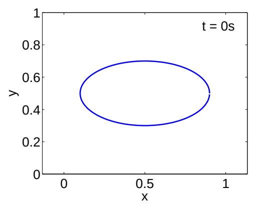
$$\boldsymbol{X}(s,0) = \left(\frac{1}{2} + r_{\max}\cos(2\pi s),\ \frac{1}{2} + r_{\min}\sin(2\pi s)\right),$$

which resides in a stationary fluid ($\boldsymbol{u}(\boldsymbol{x},0) = 0$). Since the fluid in the interior of the membrane is confined, the membrane will oscillate and eventually settle into a circular state as shown in Figure 1.

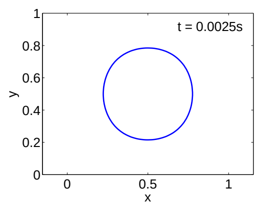
Before any IB simulation using MatIB, the solver first needs to be loaded. In the `EllipticalMembrane.m` script[2], this is accomplished using the commands:

```
% Add PATH reference in order to run solver
addpath('../../solver/Peskin-TwoStep');
addpath('../../solver/utils');
```

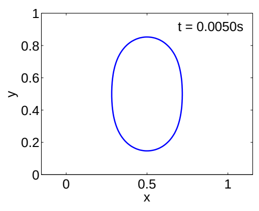
[2]The code assumes that `"cd ../../"` wil bring you to MatIB's root directory which is true when in the `"examples/Elliptical Membrane"` directory.



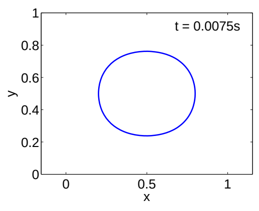
(a)



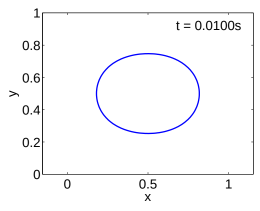
(b)



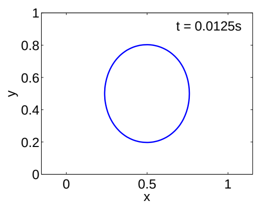
(c)



(d)



(e)



(f)

Figure 1: Profiles of an elliptical membrane at different times.

which adds the solver folders to the top of Matlab's search path. This allows the IB solver `solver/Peskin-TwoStep/IBSolver.m` and corresponding `util` functions to be called from within the Matlab script. Similarly, at the end of the Matlab script, we remove these folders from the search path using the commands:

```
% Remove PATH reference to avoid clutter
rmpath('../../solver/Peskin-TwoStep');
rmpath('../../solver/utils');
```

By using this convention, one can switch between IB solvers by simply changing the path name of the solver.

Once the script correctly links to the IB solver, the `IBSolver` function can be called with the following input parameters:

- `mu` : Dynamic viscosity μ in Navier-Stokes equation (1).
- `rho`: Fluid density ρ in Navier-Stokes equation (1).
- `sigma` : Scalar spring constant σ of the membrane defined in equation (5).
- `L`: Scalar resting strain L of the membrane defined in equation (5).
- `IC_U`, `IC_V`: A function handle describing the initial velocity of the fluid $\boldsymbol{u}(\boldsymbol{x}, 0)$ with the function profile:

```
U = IC_U(X,Y);
```

Here, `X, Y` are matrices defining points on the Eulerian grid (created by Matlab's `meshgrid` function) and `U` is a matrix defining the x component of the velocity field.

- `IC_ChiX`, `IC_ChiY`: Function handles returning the initial location of the membrane $\boldsymbol{X}(s, 0)$ with the function profile:

  ```
  ChiX = IC_ChiX(S);
  ```

  The vector `S` corresponds to the discretized Lagrangian grid $s \in [0, 1]$ with `ChiX` defining the x component of the membrane position.

- `Nx`, `Ny` : Number of Eulerian grid points along the x- and y-axes.
- `Lx`, `Ly` : Length of domain along the x- and y-axes.
- `Nb` : Number of grid points on the Lagrangian grid.
- `NTime`: Number of time steps in the simulation.
- `Tfinal`: The final time to compute the simulation to.
- `ActionFun`: A function handle which is called after each time step with the function profile

  ```
  ActionFun(dx, dy, dt, indT, X, Y, U, V, chiX, chiY, Fx, Fy);
  ```

  where

  - `dx`, `dy`: The spatial step-size of the Eulerian grid.
  - `dt`: The time-step used.
  - `indT`: Index of current time step.
  - `X`, `Y`: Matrices defining spatial Eulerian grid points.
  - `U`, `V`: Matrices containing the current velocity field of the fluid on the Eulerian grid.
  - `chiX`, `chiY`: Vectors containing the membrane's current position on the Lagrangian grid $s \in [0, 1]$.

When the function **IBSolver** finishes running, it will return `[X, Y, S, U, V, chiX, chiY]` which are matrices of the Eulerian and Lagrangian grid, the fluid velocity, and the membrane position at the last time step.

In the `EllipticalMembrane.m` script, the IB solver is called as follows:

```
% The number of grid points.
N = 32;
Nb = 3*N;

% Parameter values.
mu = 1;          % Viscosity.
sigma = 1e4;     % Spring constant.
rho = 1;         % Density.
rmin = 0.2;      % Length of semi-minor axis.
rmax = 0.4;      % Length of semi-major axis.
L = 0;           % Resting strain.

% Time step and final time.
Tfinal = .04;
dt = 5e-5;
NTime = floor(Tfinal./dt)+1;
dt = Tfinal ./ NTime;

% The initial velocity is zero.
IC_U = @(X,Y) zeros(size(X));
IC_V = @(X,Y) zeros(size(Y));

% The membrane is an ellipse.
IC_ChiX = @(S) 0.5 + rmax * cos(2*pi*S);
IC_ChiY = @(S) 0.5 + rmin * sin(2*pi*S);

% The action function.
Action = @( dx, dy, dt, indexT, X, Y, U, V, chiX, chiY, Fx, Fy)...
    PlotMembrane(X, Y, U, V, chiX, chiY, 1);

% Do the IB solve.
[X, Y, S, U, V, chiX, chiY] = ...
        IBSolver(mu, rho, sigma, L, IC_U, IC_V, IC_ChiX, IC_ChiY,...
        N, N, 1, 1, Nb, NTime, Tfinal, Action);
```

Note that there is a handy routine in the `solver/utils` folder called `PlotMembrane` which will plot the membrane's position and fluid velocity field.

4 Acknowledgements

MatIB grew out of a course project and has since been revised and used by numerous students at Simon Fraser University. We would specifically like to thank Professor John Stockie for his insight and feedback.

References

[1] Ming-Chih Lai and Charles S. Peskin. An immersed boundary method with formal second-order accuracy and reduced numerical viscosity. *Journal of Computational Physics*, 160(2):705–719, 2000.

[2] Charles S. Peskin. Flow patterns around heart valves: A numerical method. *Journal of Computational Physics*, 10(2):252 – 271, 1972.

[3] Charles S. Peskin. The immersed boundary method. *Acta Numerica*, 11:479–517, 2002.