fluids

Article

Suite-CFD: An Array of Fluid Solvers Written in MATLAB and Python

Nicholas A. Battista

Department of Mathematics and Statistics, The College of New Jersey, 2000 Pennington Road, Ewing Township, NJ 08628, USA; battistn@tcnj.edu; Tel.: +1-609-771-2445

Received: 1 October 2019; Accepted: 19 February 2020; Published: 25 February 2020

Abstract: Computational Fluid Dynamics (CFD) models are being rapidly integrated into applications across all sciences and engineering. CFD harnesses the power of computers to solve the equations of fluid dynamics, which otherwise cannot be solved analytically except for very particular cases. Numerical solutions can be interpreted through traditional quantitative techniques as well as visually through qualitative snapshots of the flow data. As pictures are worth a thousand words, in many cases such visualizations are invaluable for understanding the fluid system. Unfortunately, vast mathematical knowledge is required to develop one's own CFD software and commercial software options are expensive and thereby may be inaccessible to many potential practitioners. To that extent, CFD materials specifically designed for undergraduate education are limited. Here we provide an open-source repository, which contains numerous popular fluid solvers in 2*D* (projection, spectral, and Lattice Boltzmann), with full implementations in both MATLAB and Python3. All output data is saved in the *.vtk* format, which can be visualized (and analyzed) with open-source visualization tools, such as VisIt or ParaView. Beyond the code, we also provide teaching resources, such as tutorials, flow snapshots, measurements, videos, and slides to streamline use of the software.

Keywords: fluid dynamics education; applied mathematics education; computational fluid dynamics; fluids visualization; open-source CFD; projection method; spectral fluid solver; Lattice Boltzmann Method; cavity flow; circular flow; interacting vortices; flow past cylinder; flow past porous cylinder; MATLAB; Python

1. Introduction

Computational Fluid Dynamics (CFD) models are being applied to problems across all sciences and engineering. From designing more aerodynamic sportswear and vehicles, to understanding animal locomotion, to personalized medicine, to disease transmission, and to predicting hurricanes and atmospheric phenomena, it is difficult to find situations where greater knowledge of the underlying fluid dynamics isn't desired or valuable. Due to its immense importance, many scientists have dedicated their careers to the field, whether they study particular fluid phenomena or develop tools, either experimental or numerical, for other scientists and engineers to use. Possibly because of how vital understanding fluid dynamics is to human flourishing, proving existence, uniqueness, and the possible breakdown of solutions to the system of non-linear partial differential equations that govern fluid dynamics is one of the *Millennium Problems*, to which the Clay Institute offers a 1 million dollar prize for solving [1]. On the other hand, CFD allows us to solve these equations, the Navier-Stokes equations, which are the equations that detail the conversation of momentum and mass for a fluid. For an incompressible, viscous fluid, they can be written as follows,

$$\rho\left[\frac{\partial \mathbf{u}}{\partial t}(\mathbf{x},t) + \left(\mathbf{u}(\mathbf{x},t)\cdot\nabla\right)\mathbf{u}(\mathbf{x},t)\right] = -\nabla p(\mathbf{x},t) + \mu\Delta\mathbf{u}(\mathbf{x},t) \quad (1)$$

$$\nabla \cdot \mathbf{u}(\mathbf{x}, t) = 0 \tag{2}$$

where $\mathbf{u}(\mathbf{x}, t)$ and $p(\mathbf{x}, t)$ are the fluid's velocity and pressure, respectively. The physical properties of the fluid are given by ρ and μ, its density and dynamic viscosity, respectively. The independent variables are the time, t, and spatial position, $\mathbf{x}$. Here all variables that pertain to the fluid, e.g., $\mathbf{u}$ and p, are written in an Eulerian framework on a fixed Cartesian mesh, $\mathbf{x}$. Think of a fixed Cartesian mesh as a rectangular grid to which over the course of the simulation physical quantities, like velocity, pressure, etc., are measured at specific lattice points. In an Eulerian framework, specific blobs of fluid are not tracked over time, as though they are billiard balls, but rather, we focus on measuring quantities of interest at specific locations in space, located on a fixed grid. Note that Equations (1) and (2) provide the conservation of momentum and mass, respectively.

Many techniques have been developed to solve the above system of equations, such as projection methods [2–6], spectral methods [7–9], and Lattice Boltzmann methods [10–12]; however, they are typically left out of undergraduate curricula due to the mathematical background required for implementation, while they are commonly found in graduate curricula [13]. Moreover, it is not only mathematical knowledge that is a barrier; CFD requires expertise in at least four knowledge domains [14]: flow physics, numerical methods, computer programming, and validation, either experimental or theoretical. Thankfully, this has not stopped academics from developing and researching methods to integrate CFD into undergraduate courses [13–21]. Some have offered best practices guides and advice for future CFD undergraduate courses [13,21]. Stern et al. [13] has suggested for beginners to focus on the overall CFD process and flow visualizations, in order to solidify their fundamental understanding of fluid physics. Appropriate flow visualizations also help students validate and verify their simulations through careful inspection [21].

As computer literacy is rapidly becoming more of a central focus in undergraduate curricula, CFD could be a natural outlet where students may not only implement and test algorithms, but also practice proper data analysis and data visualization, while building computational experience. Much of the fluid dynamics software that has been used in previous courses is either foreign to students or only commercially available [14,17,19,20]. Commercial software has been recommended by the American Physical Society and American Association of Physics Teachers as a tool to help students prepare for 21st century careers by exposing them to a greater range of experiences and opportunities [22,23]. Unfortunately due to expensive software licenses, some schools, particularly smaller institutions, may not be able to afford such software, especially if they would only be used intermittently to complement the existing course material.

Recently, popular programming languages selected for many undergraduate curricula in science, mathematics, and engineering have been either MATLAB or Python [24–28]. MATLAB [29] and Python [30] are both interpreted languages, making it easier (and more attractive) for students to begin programming [31]. Moreover if students have already been exposed to these programming languages, giving them a platform to refine their skills in these languages while at the same strengthening their intuition in fluid dynamics may prove beneficial.

For this reason, many scientists, engineers, and mathematicians who are passionate about education have begun developing CFD software intended for educational purposes written in these languages [32–36]. Such open source software packages can be integrated into courses in a well-streamlined fashion, with tutorials, activities, exercises, and notes provided, and may even blur the line between classroom activities and contemporary research [32,35–37]. In particular L.A. Barba and G. Forsyth [33] have developed a guide for students to numerically solve the Navier-Stokes equations in 12 scaffolding steps and L.A. Barba and O. Mesnard [34] have developed a similar scaffolding lesson structure for students to study classical aerodynamics using potential flow models. Both are written in Python using Jupyter Notebooks [38]. Pawar and San [39] developed modules for teaching advanced undergraduate and graduate students how to develop their own fluid solvers in the Julia programming language [40].

Rather than develop a collection of modules that teach how to implement particular CFD numerical schemes, we offer the scientific community a variety of popular fluid solvers that solve traditional problems in fluid dynamics, with two independent but equal implementations written in MATLAB and Python. The emphasis is not placed on students having to develop or implement numerical schemes, but to allow them the opportunity to get an accelerated start in CFD, and run, tweak, and analyze simulations of traditionally studied problems in fluids courses, as well as explore data visualization and how to present data that is appropriate for fluid physics interpretation. All of this can be done while testing their own hypotheses in familiar programming environments. In the remainder of this manuscript, we will give a high-level overview of the CFD schemes currently implemented in the software (Section 2), guided tutorials on how to run, visualize, and analyze simulations (Section 3), and provide numerous examples to which students may elect to study or modify as well (Section 4).

Furthermore, we provide an in-depth mathematical description of each numerical scheme in Appendix B. The open-source software repository can be accessed at https://github.com/nickabattista/Holy_Grail. Any simulation data produced from the software can be visualized and analyzed in open-source software, such as VisIt [41], maintained by Lawrence Livermore National Laboratory, or ParaView [42], developed by Kitware, Inc., Los Alamos National Laboratory, and Sandia National Labs. We also provide teaching resources, such as slides, figures, and movies in the Supplementary Materials.

2. Brief Overview of the Three Fluid Solvers

In this section we will give a brief overview of the three fluid solvers currently implemented in the software—a projection method, a spectral (FFT) method, and the Lattice-Boltzmann method. Our goal is to provide students a high-level overview of the methods and how they compare to one another. Further details regarding their mathematical foundations and implementations are given in Appendix B.

2.1. Projection Method

The projection method was first introduced by Chorin in 1967 [2] and independently by Temam in 1968 [4], to solve the viscous, incompressible Navier-Stokes equations. Projection methods are finite difference based numerical schemes [43], in which the fluid equations are solved in an Eulerian framework, i.e., the computational grid is static and fluid dynamics variables, like velocity, pressure, etc., are measured at particular locations on the grid over time during a simulation. Projection methods have been extensively used throughout the fluid dynamics community for decades, in numerous applications across many fields, while being continually improved for efficiency and accuracy [5,44–46].

In a nutshell, this method decouples the velocity and pressure fields, using operator splitting and a Helmholtz-Hodge decomposition. This makes it possible to explicitly solve Equations (1) and (2) in a few steps [2] while also increasing computational efficiency. Please see Appendix B.1 for further mathematical details.

Furthermore, projection methods allow one to define explicit boundary conditions (BCs) for velocity on edges of the computational domain, whether they are *Dirichlet*, *Neumann*, or *Robin* boundary conditions [5,47]. In our software, we allow users to impose tangential velocity boundary conditions along the edges of the rectangular domain, while the normal direction boundary conditions for velocity are explicitly set to zero. The choice to set normal BCs to zero was made in order to simplify the code by ensuring that volume conservation would be automatically satisfied, i.e., situations will not arise in which a specified amount of fluid is being pumped into the domain while a different amount is leaving the domain per unit time.

Having defined the fluid velocity as $\mathbf{u} = (u, v)$, one can set the boundary conditions for u on the top and bottom of the domain ($v = 0$ along these edges) or v on the left and right sides ($u = 0$ along these edges). See Figure 1 for an illustration. Later in Section 4, we will showcase two different traditional problems in fluid dynamics with a projection method:

1. Cavity Flow
2. Circular Flow in a Square Domain

The examples are different due to the changes in boundary conditions that the user can impose, see Figure 1. In Figure 1 we illustrate the boundary conditions for each example: cavity flow (a) and circular flow (b). This figure also illustrates that the domain could be constructed to be either rectangular or square and that the boundary conditions do not have to be uniform among all sides, nor across the domain.

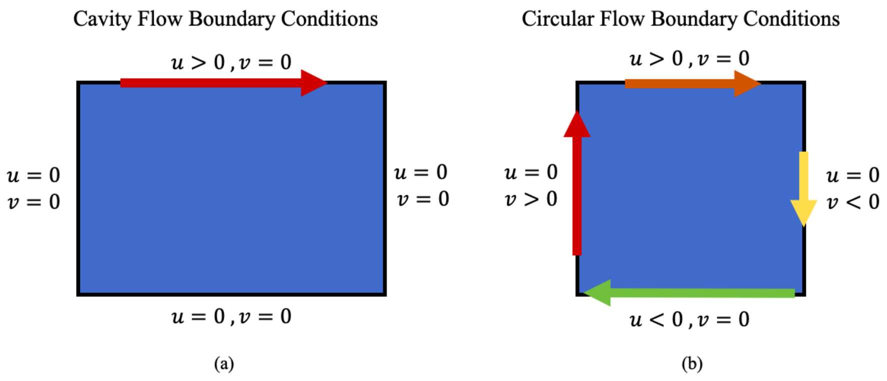


Figure 1. Boundary conditions are depicted pertaining to (**a**) cavity flow and (**b**) circular flow for the projection methods examples.

We will also compare different fluid scales in both examples, using the Reynolds Number, *Re*. The Reynolds Number is given by

$$Re = \frac{\rho L V}{\mu}, \tag{3}$$

where ρ and μ are the fluid's density and dynamic viscosity, respectively, while L and V are a characteristic length- and velocity-scale of the system. In essence, Re is a ratio of inertial forces to viscous forces in a system. If $Re >> 1$, inertia dominates, if $Re << 1$, viscous forces dominate, and if $Re = 1$, the inertia and viscous forces are balanced. An example of high Re flow would be the flow around a marble quickly falling through air. An example of low Re flow would be the fluid flow around a marble slowly falling through honey.

2.2. Spectral Method (FFT)

Spectral methods are a different class of differential equation solvers. They are well-known for achieving highly accurate solutions, i.e., spectral accuracy [48–50]. Spectral accuracy occurs when error decreases exponentially with a small change in grid resolution, e.g., there is a linear relationship between the logarithm of error and grid resolution, i.e.,

$$\log(error) \sim N,$$

where N is the number of grid points. These methods approximate solutions using orthogonal expansions, such as Fourier Series or other orthogonal basis of functions, like Chebyshev Polynomials or Legendre Polynomials. In contrast to finite difference based schemes, like the projection method, where solutions are approximated at specific locations in space, in spectral methods a particular orthogonal series expansion is chosen and the earnest is on finding the coefficients of the expansion.

Here we chose to use the basis of Discrete Fourier Transform (DFT). The DFT can be used to express functions that are not periodic, unlike Fourier Series expansions, which are used to represent periodic functions. Moreover, we used the Fast Fourier Transform (FFT), which yields the same numerical values as a DFT, but is considerable more computationally efficient, i.e., fast.

Thus, we implemented a FFT based spectral scheme to solve the viscous, incompressible Navier-Stokes equations in $2D$. We also chose to solve these equations in their vorticity formulation (see Appendix B.2 for mathematical details). The vorticity formulation will lead to a few numerical benefits.

In a nutshell, this will recast our problem into solving for the vorticity, $\boldsymbol{\omega} = \omega\hat{k}$, and the streamfunction, ψ. Velocity can be defined in terms of the curl of the streamfunction, i.e., a vector potential, see below

$$\mathbf{u} = \nabla \times \psi\hat{k}. \tag{4}$$

Hence this allows us to find the velocity field, $\mathbf{u} = (u, v)$, once we find ψ, e.g.,

$$u = \frac{\partial\psi}{\partial y} \quad \text{and} \quad v = -\frac{\partial\psi}{\partial x}.$$

Moreover, the incompressibility condition (Equation (2)) will be automatically satisfied since $\mathbf{u}$ is defined to be the curl of a scalar, and the divergence of the curl of a vector is always identically zero.

Furthermore, working in the vorticity formulation requires us to only solve a Poisson equation for the streamfunction in terms of the vorticity, $\boldsymbol{\omega}$, i.e.,

$$\Delta\psi = -\omega. \tag{5}$$

Notice that Equation (5) is a linear equation. Moreover, upon taking the FFT of the Equation (5), the computation is transformed into frequency (Fourier) space, which offers two advantages—increased speed and accuracy.

In practice, we will solve for the streamfunction at the next time-step based on the previously computed vorticity and then update the vorticity using information from the newly updated streamfunction. We chose to use a Crank-Nicholson time-stepping scheme [51] to update the vorticity to the next time-step. The Crank-Nicholson scheme is well-known for being unconditionally-stable for diffusion problems and as well as for being 2nd order accurate in time and space, as an implicit method [52]. These accuracy and stability properties make the Crank-Nicholson an ideal candidate for numerically solving such equations.

Although using a FFT (spectral method) preserves high accuracy of the spatial information at a particular time-step, numerical error still unfortunately creeps in due to the time-stepping nature of solving an evolution equation for the fluid's momentum (see Equation (A15) in Appendix B.2). That is, evolving the system forward in time is where the brunt of the error takes place, rather than the spatial discretization.

In contrary to the projection method, this FFT-based approach allows us to explore periodic boundary conditions. For example, if fluid flow is moving vertically-upwards through the top of the computational domain, it will re-enter moving vertically-upwards through the bottom of the domain. This has advantages and disadvantages. Some advantages are that explicit boundary conditions do not need to be satisfied nor enforced, which aids in computational speed-up. However, to that extent, one disadvantage is that this makes it more difficult to enforce particular boundary conditions when desired. FFT-based fluid solvers have been used in the fluid-structure interaction community, in particular within immersed boundary methods [32,37,53], when one wants to study the interactions of an object and the fluid to which it's immersed. As the focus is on the object and fluid interactions, it is typically modeled away from the domain boundaries in the middle of the computational domain to avoid boundary artifacts. For a mathematically detailed description of this FFT-based spectral method see Appendix B.2.

For this particular FFT-based fluid solver for the vorticity-formulation of the viscous, incompressible Navier-Stokes equations, we will highlight the following examples:

1. Side-by-Side Vorticity-Region Interactions
2. Evolution of Vorticity from an Initial Velocity Field

Contrary to the examples shown for the projection method in Section 2.1, these examples are not differentiated by different boundary conditions, but rather different initial vorticity configurations in the computational domain. Figure 2 gives the initial vorticity configurations for the two cases considered: side-by-side vorticity region interactions (left) and an initial vorticity field that defined by a velocity vector field (right). In Figure 2a, counterclockwise (*CCW*) and clockwise (*CW*) correspond to regions of uniform vorticity, where vorticity initialized as a positive or negative constant for *CCW* and *CW*, respectively. In all other regions of the computational domain, the vorticity is either initialized to zero (Figure 2a). Although not shown, one could also initialize simulations on a rectangular grid. In Figure 2b, the initial vorticity is defined by computing the vorticity from a simulation's velocity vector field. The velocity vector field was taken from a time-point in a *Pulsing_Heart* simulation [36] in the open-source *IB2d* software [32,37]. In this example, we illustrate that if a snapshot of the velocity field is known, it is possible to evolve the fluid dynamics forward using this spectral (FFT) method, even though it is based on the vorticity formulation of the Navier-Stokes equations.

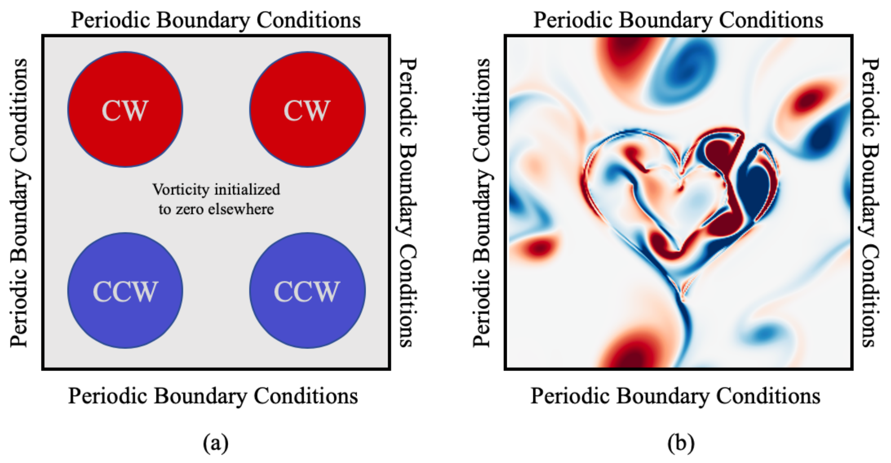

Periodic Boundary Conditions
Periodic Boundary Conditions
Periodic Boundary Conditions
Periodic Boundary Conditions
CW
CW
Vorticity initialized
to zero elsewhere
CCW
CCW
(a)
Periodic Boundary Conditions
Periodic Boundary Conditions
Periodic Boundary Conditions
Periodic Boundary Conditions
(b)

Figure 2. Illustrations of the boundary conditions and vorticity initialization for the cases of (**a**) interacting vorticity regions and (**b**) an initial vorticity field defined from a velocity vector field.

2.3. Lattice Boltzmann Method

The Lattice Boltzmann method (LBM) does not explicitly (or implicitly) solve the incompressible, Navier-Stokes equations, rather it uses discrete Boltzmann equations to model the flow of a fluid [11]. In a nutshell, the LBM tracks fictitious particles of fluid flow, thinking of the problem more as a transport equation, e.g.,

$$\frac{\partial f}{\partial t} + \mathbf{u} \cdot \nabla f = \Omega, \tag{6}$$

where $f(\mathbf{x}, t)$ is the particle distribution function, i.e., a probability density, $\mathbf{u}$ is the fluid particle's velocity, and Ω is what is called the collision operator. However, rather than these particles moving in a Lagrangian framework, the Lattice Boltzmann method simplifies this assumption and restricts the particles to the nodes on a lattice.

In traditional CFD methods, one typically discretizes the fluid equations in an Eulerian framework, as in a projection or spectral method. When you are free of such restriction, as in the LBM, it allows one to more readily deal with complex boundaries, model porous structures and multi-phase flow, as well as incorporate microscopic interactions, while also allowing for massive parallelization of the algorithm, leading to increased computational efficiency [12,54–56].

The LBM involves *collision* and *streaming* steps, where fictional fluid particles consecutively propagate (and collide) over a discretized lattice, and the fluid density is evolved forward. There are multiple ways to handle boundary conditions in the LBM. One such way is to use *bounce-back boundary conditions*, which are executed by masking points in the domain via boolean expressions, thus defining regions where the fluid can and cannot flow [55]. In short, the propagating (streaming) directions are simply reversed when they hit a boundary node, see Figure A3 in Appendix B.3. The bounce-back conditions can be used to enforce the necessary and physical *no-slip* boundary conditions for fluid flow as well as defining solid objects in the interior of the grid (for an example see Figure 33). For a detailed mathematical description of the LBM see Appendix B.3.

To highlight the LBM fluid solver, we showcase the following examples:

1. Flow past one or more cylinders
2. Flow past porous cylinder

3. How to Run the Simulations, Visualize, and Analyze

In this section we will give instructions on how to run, visualize, and analyze the simulations produced in this software. We will explain how to change parameters below as well. In particular, we will provide both overviews of what is possible to visualize and analyze as well as a guided walk-through of the steps required, designated by *Guide* in their title. We broke this section into the following subsections:

1. Running a Simulation
2. Visualizing the Data in VisIt
3. *Guide*: Running the Spectral (FFT) Method's 'bubble3' Example
4. *Guide*: Visualizing the Spectral (FFT) Method's 'bubble3' Data
5. *Guide*: Analyzing the Spectral (FFT) Method's 'bubble3' Data

3.1. Running a Simulation

Here we will briefly describe how to run each fluid solver's corresponding simulations. To illustrate the software structure and how to run the simulations, we will use the MATLAB (https://www.mathworks.com/products/matlab.html) [29] version. Note that the Python implementation is consistent in its structure as well as naming conventions.

To run the simulation, one would need to do the following:

1. Either clone the Holy_Grail repository or download the Holy_Grail zip file at https://github.com/nickabattista/Holy_Grail/ to your local machine. Note you can download or clone this repository to any directory on your local machine.

2. Open MATLAB and go to the appropriate sub-directory for the particular fluid solver example you wish to run, e.g.,

   - `Projection_Method`: for the projection method solver and its examples
   - `FFT_NS_Solver`: for the spectral (FFT) method solver and its examples
   - `Lets_Do_LBM`: for the Lattice Boltzmann method solver and its examples

   and go into the MATLAB subfolder. (If you wanted to run an example in Python, you would change directories until you are in the complementary Python subfolder.)

3. The main script name for each fluid solver is named accordingly, e.g.,

   - `Projection_Method.m` - main script for the projection method
   - `FFT_NS_Solver.m` - main script for the spectral (FFT) solver
   - `lets_do_LBM.m` - main script for the Lattice Boltzmann solver

   Each of these scripts contains all the functions and modules necessary to run each example. The file `print_vtk_files.m` produces the *.vtk* data files during each simulation. The user should not have to change this file, unless they do not wish to save some of the data for storage reasons. If this is the case, this can be accomplished by commenting out the lines corresponding to whichever data is not to be saved.

4. Depending on the subfolder that you are in, to run an example, you would type its main script name into the MATLAB command window and click `enter`.

5. The simulations are not instantaneous; they may take a few minutes. Upon starting the simulation, a folder named `vtk_data` is produced, which will be used for storing the simulation data files in *.vtk* format. As the simulation runs, it will dump more data into this folder. Note that depending on the simulation you choose and the grid resolution, the simulations may take from a few minutes to $\sim$30 min. Moreover, if you wish to run multiple simulations, note that the default folder name is set to `vtk_data`, and so running additional simulations will overwrite any previously saved data in the previous `vtk_data` folder. Please manually change the name of the folder after each simulation to avoid this.

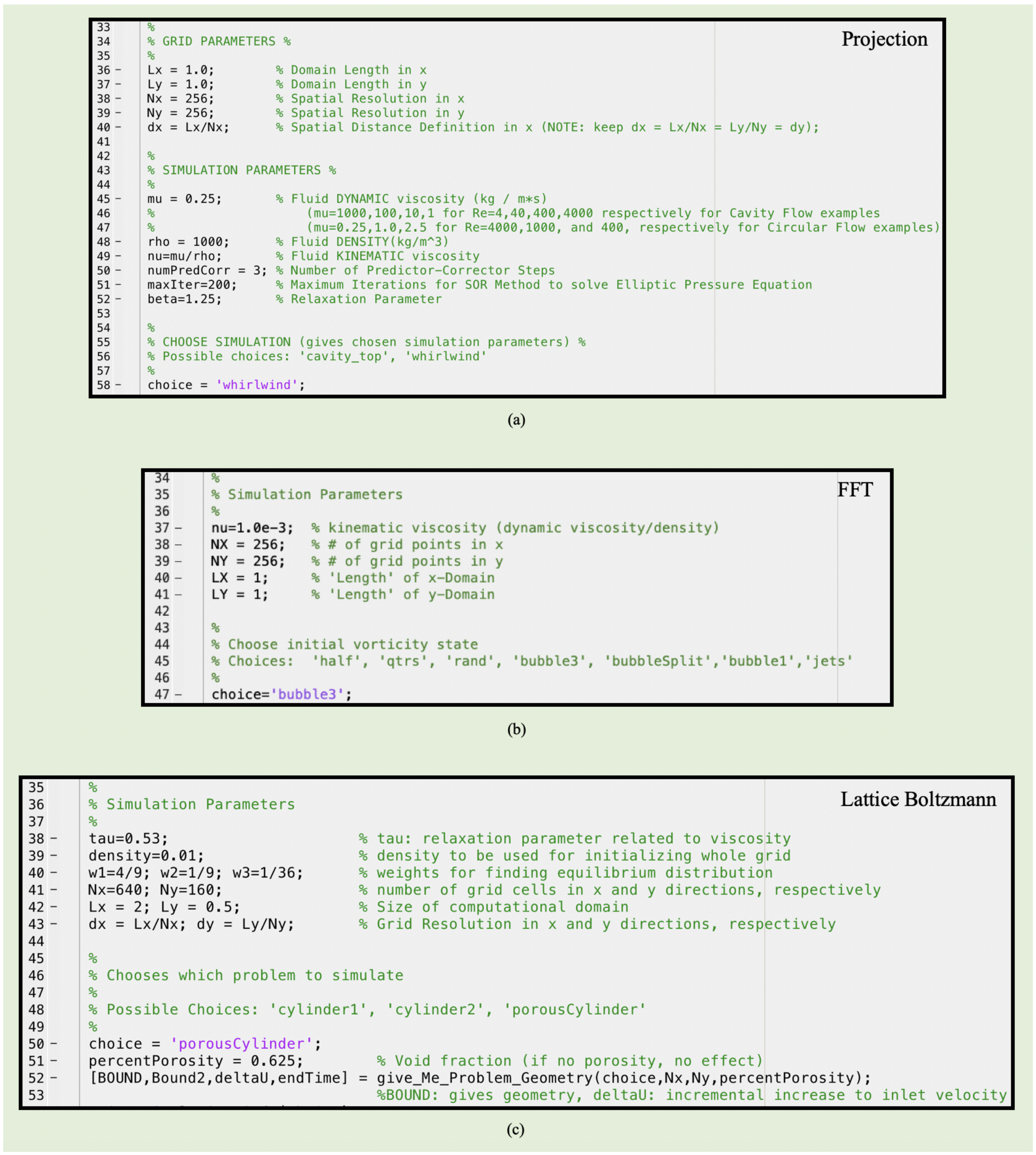
Digging deeper into each fluid solver script:

1. If one desires to change parameters for a particular simulation, e.g., grid resolution, viscosity (either μ (Projection) or ν (FFT) or τ (LBM), etc.), options are available to the user without digging deep into any of the sub-functions or individual modules beyond the main function itself, see Figure 3.

2. Figure 3 highlights the main choices the user can make for each of the three fluid solvers (a) Projection, (b) Spectral (FFT), and (c) Lattice Boltzmann. The user can identify which of the built-in examples they wish to run by changing the appropriate string flag labeled `choice`.

3. Note that each example will run as described in Section 4 if all the parameters are selected to be those in the parameter Table that provided within each example's section. *Caution*: numerical stability conditions may be violated by choice of other parameters. To test other parameter values, we suggest making incremental variations, e.g., try doubling or halving viscosity, rather than testing values orders of magnitude apart. Numerical instability will appear if solvers cannot converge or begin dumping `NaNs` (*not-a-number*), which represent unrepresentable numbers to computers, due to their inherent floating-point arithmetic.

4. As the entire fluid-solver is contained within each script, the user can go deeper into the code to modify examples or create their own. This can be done in the following sub-functions:
    - Projection Method: *please_Give_Me_BCs* (choice)
    - Spectral (FFT) Solver: *please_Give_Initial_Vorticity_State* (choice, NX, NY)
    - Lattice Boltzmann Solver: *give_Me_Problem_Geometry* (choice, Nx, Ny, percentPorosity)



```
%
% GRID PARAMETERS %
%
Lx = 1.0;          % Domain Length in x
Ly = 1.0;          % Domain Length in y
Nx = 256;          % Spatial Resolution in x
Ny = 256;          % Spatial Resolution in y
dx = Lx/Nx;        % Spatial Distance Definition in x (NOTE: keep dx = Lx/Nx = Ly/Ny = dy);

%
% SIMULATION PARAMETERS %
%
mu = 0.25;         % Fluid DYNAMIC viscosity (kg / m*s)
%                      (mu=1000,100,10,1 for Re=4,40,400,4000 respectively for Cavity Flow examples
%                      (mu=0.25,1.0,2.5 for Re=4000,1000, and 400, respectively for Circular Flow examples)
rho = 1000;        % Fluid DENSITY(kg/m^3)
nu=mu/rho;         % Fluid KINEMATIC viscosity
numPredCorr = 3; % Number of Predictor-Corrector Steps
maxIter=200;       % Maximum Iterations for SOR Method to solve Elliptic Pressure Equation
beta=1.25;         % Relaxation Parameter

%
% CHOOSE SIMULATION (gives chosen simulation parameters) %
% Possible choices: 'cavity_top', 'whirlwind'
%
choice = 'whirlwind';
```

```
%
% Simulation Parameters
%
nu=1.0e-3;  % kinematic viscosity (dynamic viscosity/density)
NX = 256;   % # of grid points in x
NY = 256;   % # of grid points in y
LX = 1;     % 'Length' of x-Domain
LY = 1;     % 'Length' of y-Domain

%
% Choose initial vorticity state
% Choices:  'half', 'qtrs', 'rand', 'bubble3', 'bubbleSplit','bubble1','jets'
%
choice='bubble3';
```

```
%
% Simulation Parameters
%
tau=0.53;                        % tau: relaxation parameter related to viscosity
density=0.01;                    % density to be used for initializing whole grid
w1=4/9; w2=1/9; w3=1/36;         % weights for finding equilibrium distribution
Nx=640; Ny=160;                  % number of grid cells in x and y directions, respectively
Lx = 2; Ly = 0.5;                % Size of computational domain
dx = Lx/Nx; dy = Ly/Ny;          % Grid Resolution in x and y directions, respectively

%
% Chooses which problem to simulate
%
% Possible Choices: 'cylinder1', 'cylinder2', 'porousCylinder'
%
choice = 'porousCylinder';
percentPorosity = 0.625;           % Void fraction (if no porosity, no effect)
[BOUND,Bound2,deltaU,endTime] = give_Me_Problem_Geometry(choice,Nx,Ny,percentPorosity);
                                   %BOUND: gives geometry, deltaU: incremental increase to inlet velocity
```

Figure 3. Options for practitioners to change the grid resolution and size, fluid properties (density, viscosity), and other solver attributes within the software for the (**a**) Projection method, (**b**) Fast Fourier Transform (FFT)-based spectral solver, and (**c**) Lattice Boltzmann solver. The possible choices for built-in examples are also listed for each.

3.2. Visualizing the Data in VisIt

In this section we will briefly describe how produce visualizations, such as those Section 4. We will briefly detail the steps to visualize the data using the open-source software VisIt (https://visit.llnl.gov) [41]. Note that each simulation produces data in the *.vtk* format and so one could alternatively use the open-source software ParaView [42] for visualization purposes as well. Later, in Section 3.3 we will guide a user through running an example (the spectral (FFT) method's `bubble3` example) followed by a step-by-step guide to visualizing its corresponding data in Section 3.4. We used VisIt version 2.13.3.

To visualize the data, one would need to do the following:

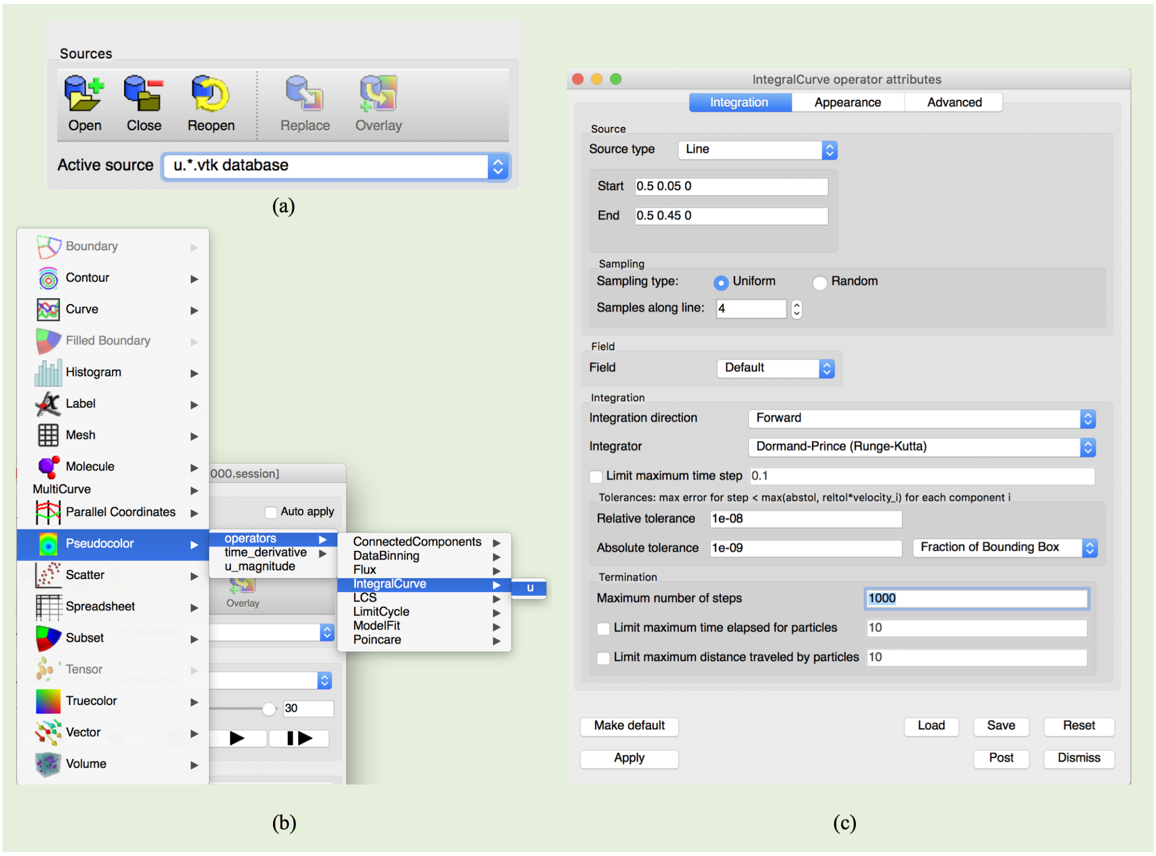
1. Open VisIt [41]
2. Open the desired Eulerian data (magnitude of velocity, vorticity, velocity vector field, etc.) from the `vtk_data` folder.
3. A table of the possible data stored and their corresponding name when saved is found below in Table 1.

Table 1. Data stored to *.vtk* format and its corresponding storage name.

Parameter	Type	'*.vtk*' Name
Magnitude of Velocity	Scalar	*uMag*
Velocity Vectors	Vector	*u*
Horizontal Velocity	Scalar	*uX*
Vertical Velocity	Scalar	*uY*
Pressure (*projection only*)	Scalar	*P*
Vorticity	Scalar	*Omega*
Geometry (*LBM only*)	Mesh	*Bounds*

4. *To Visualize Geometry (LBM only)*:
   (a) Click `Open`
   (b) Go to the `vtk_data` data folder that the simulation produced
   (c) Click on the grouping of `Bounds`, click `OK`
   (d) In VisIt, click on `Add` then `Mesh`→`mesh`.
   (e) Then click `Draw`
   (f) You can elect to change the color of boundary or size by double clicking on the Mesh in the VisIt database listing window.

5. *To Visualize Velocity Vectors*:
   (a) Click `Open`
   (b) Go to the `vtk_data` data folder that the simulation produced
   (c) Click on the grouping of `u`, click `OK`
   (d) In VisIt, click on `Add` then `Vector`→`u`
   (e) Then click `Draw`. An error message might pop up during at time-step zero (first data point) if velocity is identically zero
   (f) You can elect to change the color, size, scaling, and number of velocity vectors by double-clicking on *u* in the VisIt database listing window
   (g) To add streamlines of the velocity field, first make sure that your *Active Source* is set to *u* (see Figure 4a), then click `Add`→`Pseudocolor`→`operators`→`IntegralCurve`→`u` (see Figure 4b). Note you have the option to put numerous seed points for streamlines to stem in the domain. Here we have chosen to use a line of points, sampled it at 4 evenly spaced locations within that line, and specified particular integration tolerances, and a maximum number of steps (see Figure 4c). Click `Draw`. You can also adjust the streamline aesthetics to how you desire, e.g., color, thickness, etc.



Active source: u.*.vtk database

Source
Source type: Line
Start: 0.5 0.05 0
End: 0.5 0.45 0

Sampling
Sampling type: [✓ Uniform] Random
Samples along line: 4

Field
Field: Default

Integration
Integration direction: Forward
Integrator: Dormand-Prince (Runge-Kutta)
Limit maximum time step: 0.1
Tolerances: max error for step < max(abstol, reltol*velocity_i) for each component i
Relative tolerance: 1e-08
Absolute tolerance: 1e-09 Fraction of Bounding Box

Termination
Maximum number of steps: 1000
Limit maximum time elapsed for particles: 10
Limit maximum distance traveled by particles: 10

Figure 4. VisIt graphical user interfaces (GUI) interface showing steps to illustrate velocity streamlines, e.g., (**a**) active source must be set on velocity field, (**b**) sequence to plot streamlines, and (**c**) streamline seeding location(s), accuracy tolerances, and solver-type.

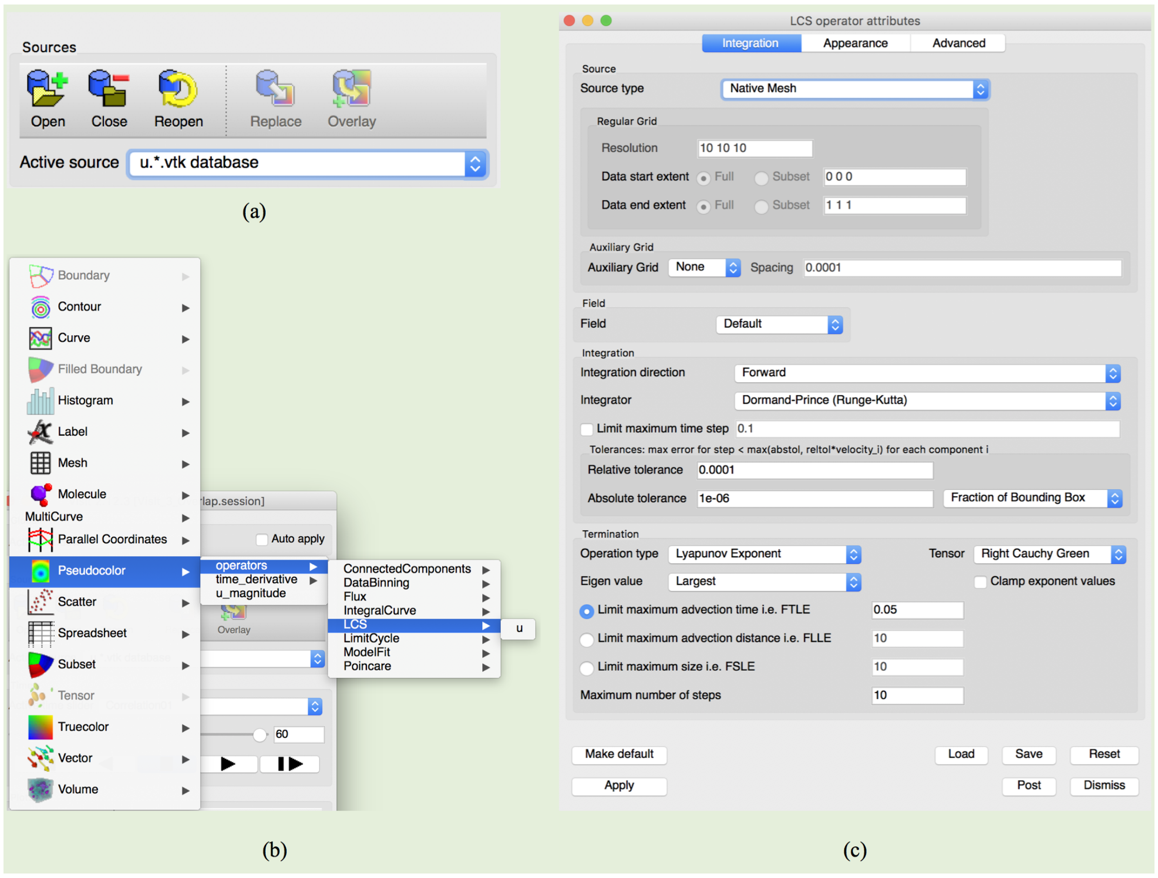
6. *To Visualize the Eulerian scalar data (e.g., Vorticity, Magnitude of Velocity, etc.):*

   (a) Click `Open`
   (b) Go to the `vtk_data` data folder that the simulation produced
   (c) Click on the grouping of the desired Eulerian scalar data, for example, `Omega` (for Vorticity), click `OK`
   (d) In VisIt, click on `Add` then `Pseudocolor`→`Omega`.
   (e) Then click `Draw`
   (f) You can elect to change the colormap and/or colormap scaling by double clicking on Omega in the VisIt database listing window. The following table (Table 2) details the specific colormap used for each scalar variable in the manuscript,

Table 2. Colormaps used for each scalar variable for visualization.

Parameter	Colormap
Magnitude of Velocity	RdYlBu or BrBG (Section 4.1 only)
Horizontal Velocity	RdYlBu
Vertical Velocity	RdYlBu
Pressure	orangehot (Section 4.1 only)
Vorticity	hot_desaturated
FTLE	PuRd

(g) To visualize the finite-time Lyapunov exponent (FTLE), your *Active Source* in Visit (see Figure 5a) must be on the velocity vectors, u. Then click `Add`→`Pseudocolor`→`operators`→`LCS`→`u` (see Figure 5b). We used the attributes in Figure 5c for computing the FTLE. In particular, we limited the maximum advection time to 0.05 and maximum number of steps to 10. Once those are changed, click `Draw`.



Active source: u.*.vtk database

Source
Source type: Native Mesh
Regular Grid
Resolution: 10 10 10
Data start extent: Full / Subset 0 0 0
Data end extent: Full / Subset 1 1 1
Auxiliary Grid
Auxiliary Grid: None Spacing: 0.0001
Field
Field: Default
Integration
Integration direction: Forward
Integrator: Dormand-Prince (Runge-Kutta)
Limit maximum time step: 0.1
Tolerances: max error for step < max(abstol, reltol*velocity_i) for each component i
Relative tolerance: 0.0001
Absolute tolerance: 1e-06 Fraction of Bounding Box
Termination
Operation type: Lyapunov Exponent Tensor: Right Cauchy Green
Eigen value: Largest Clamp exponent values
[✓ Limit maximum advection time i.e. FTLE]: 0.05
Limit maximum advection distance i.e. FLLE: 10
Limit maximum size i.e. FSLE: 10
Maximum number of steps: 10

Figure 5. VisIt GUI interface showing steps to illustrate finite-time Lyapunov exponents (FTLE), e.g., (**a**) active source must be set on velocity field, (**b**) sequence to plot FTLE (e.g., Lagrangian Coherent Structures, LCS), and (**c**) attributes for plotting, e.g., accuracy tolerances, solver-type, etc.

(h) To add contours of the desired scalar variable, first make sure that your *Active Source* (see Figure 5a) is set to the correct variable, then click `Add`→`Contour`→<desired variable name>. Note that you have the option to scale the contours separate from the colormap of the variable itself. Moreover, for FTLE your active source must be u and you can modify how FTLE are computed, e.g., Figure 5c; however, we used consistent values for both FTLE computations.

3.3. Guide: Running the Spectral (FFT) Method's 'bubble3' Example

In this guided walk-through, we wish to illustrate how to run a simulation and change its parameters while also observing what information is being printed to the screen. We will use the Spectral (FFT) Method's *bubble3* example and run two different examples where each uses a different fluid kinematic viscosity, ν.

To run the simulation(s), do the following:

1. First we must make sure that we are inside of MATLAB the corresponding directory for the MATLAB version of the spectral-based solver. To do this click through: `Holy_Grail`→`FFT_NS_Solver`→`MATLAB`, see Figure 6. Note that I have downloaded (or cloned) the software into my `Documents` folder.

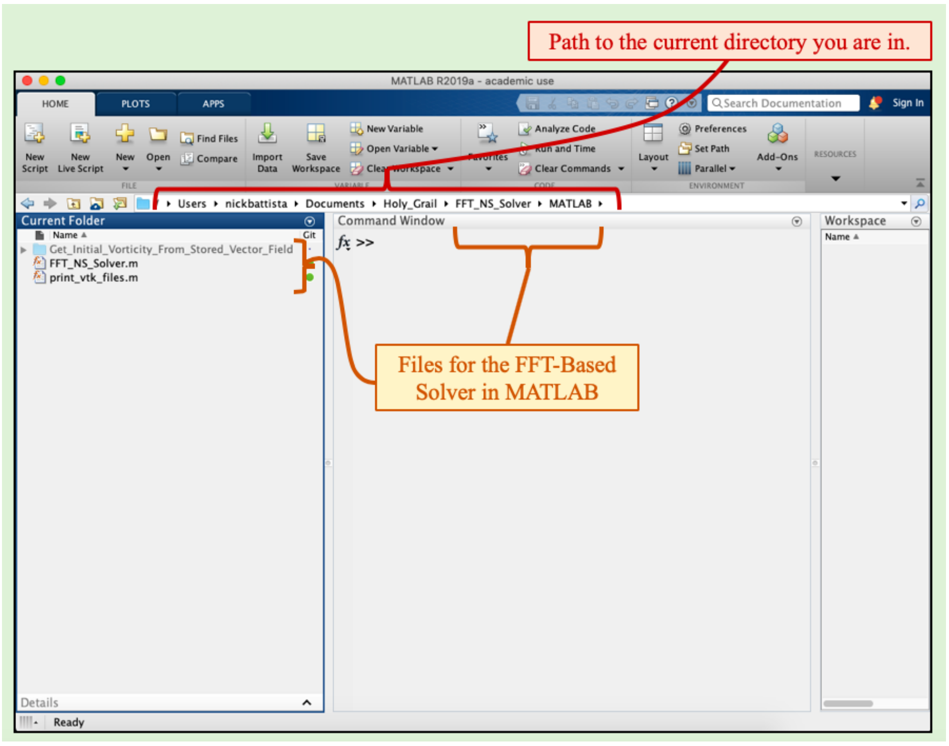
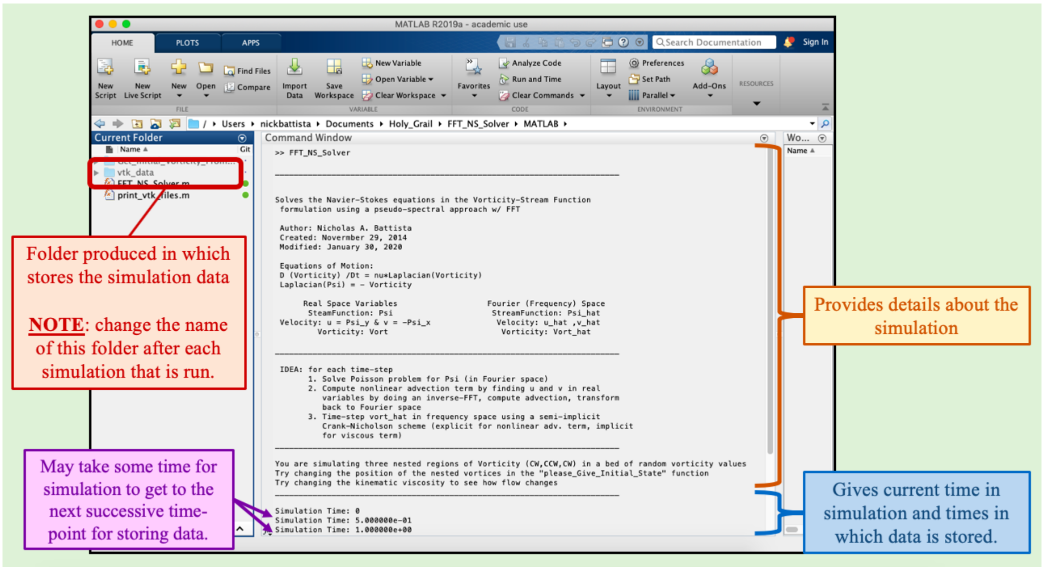


Figure 6. MATLAB GUI interface showing the path to the current directory folder for the MATLAB FFT-Based Spectral Solver as well as what files are contained within it.

2. To run this simulation, type `FFT_NS_Solver` into MATLAB's command window and click `enter`. See Figure 7 for what should print to the command window shortly thereafter. Note that the *bubble3* example is the default built-in example upon downloading the software (see Figure 3b).

   Upon starting the simulation, information regarding the simulation solver and simulation is printed to the screen. Moreover, the `current_time` within the simulation gets printed to the screen at the time-points in which the data is stored. The data is stored in the *vtk_data* folder, which is made upon starting the simulation.


MATLAB R2019a - academic use
Users ▸ nickbattista ▸ Documents ▸ Holy_Grail ▸ FFT_NS_Solver ▸ MATLAB ▸
Current Folder
vtk_data
FFT_NS_Solver.m
print_vtk_files.m
Command Window
Folder produced in which stores the simulation data
NOTE: change the name of this folder after each simulation that is run.
May take some time for simulation to get to the next successive time-point for storing data.
Provides details about the simulation
Gives current time in simulation and times in which data is stored.

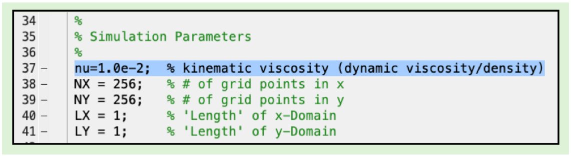
```
>> FFT_NS_Solver

 ____________________________________________________________________________


 Solves the Navier-Stokes equations in the Vorticity-Stream Function
  formulation using a pseudo-spectral approach w/ FFT

  Author: Nicholas A. Battista
  Created: Novermber 29, 2014
  Modified: January 30, 2020

  Equations of Motion:
  D (Vorticity) /Dt = nu*Laplacian(Vorticity)
  Laplacian(Psi) = - Vorticity

       Real Space Variables                  Fourier (Frequency) Space
        SteamFunction: Psi                     StreamFunction: Psi_hat
  Velocity: u = Psi_y & v = -Psi_x              Velocity: u_hat ,v_hat
          Vorticity: Vort                          Vorticity: Vort_hat

 ____________________________________________________________________________

  IDEA: for each time-step
        1. Solve Poisson problem for Psi (in Fourier space)
        2. Compute nonlinear advection term by finding u and v in real
           variables by doing an inverse-FFT, compute advection, transform
           back to Fourier space
        3. Time-step vort_hat in frequency space using a semi-implicit
           Crank-Nicholson scheme (explicit for nonlinear adv. term, implicit
           for viscous term)
 ____________________________________________________________________________

 You are simulating three nested regions of Vorticity (CW,CCW,CW) in a bed of random vorticity values
 Try changing the position of the nested vortices in the "please_Give_Initial_State" function
 Try changing the kinematic viscosity to see how flow changes
 ____________________________________________________________________________

 Simulation Time: 0
 Simulation Time: 5.000000e-01
 Simulation Time: 1.000000e+00
```

Figure 7. Showing the output on the screen when this example begins running. Note that the *vtk_data* folder is produced, in which stores the simulation data at predetermined time-points.

3. Once the simulation has finished running, we will change the *vtk_folder*'s name to *Simulation_1_Data*. This folder contains all of the data that was stored during the simulation, i.e., the vorticity, fluid velocity field, magnitude of velocity, etc.. It is imperative to change this folder name after each simulation if more simulations are to be run, or else the new data will printed over the previous simulation's data.

4. Once the folder's name has been changed, we will open the `FFT_NS_Solver.m` script to change one (or more) of the parameters. Here we will only change the fluid's kinematic viscosity, ν; however, there are other options in which you could change, such as the computational domain size and/or grid resolution, see Figure 3b. As an example change `nu` from $1.0 \times 10^{-3} \to 1.0 \times 10^{-2}$, see Figure 8. Once you have changed ν, you can close the script.

   Note that if you chose to change the grid resolution and domain size, make sure that the resolution in x and y are equal, i.e., $dx = L_x/N_x = L_y/N_y = dy$. Moreover if you increase the resolution (increase N_x, N_y) the simulations will take longer to run and the resulting data will require more storage.



```
%
% Simulation Parameters
%
nu=1.0e-2;  % kinematic viscosity (dynamic viscosity/density)
NX = 256;   % # of grid points in x
NY = 256;   % # of grid points in y
LX = 1;     % 'Length' of x-Domain
LY = 1;     % 'Length' of y-Domain
```

Figure 8. Changing the kinematic viscosity, ν, to a different value.

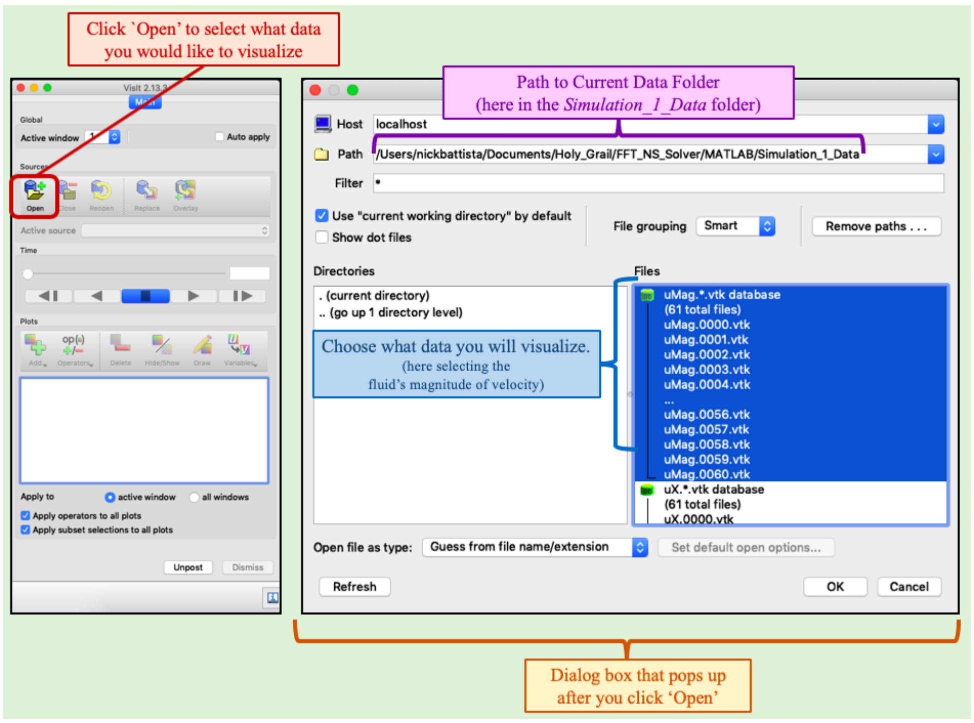
5. Finally, run this new simulation case by typing `FFT_NS_Solver` into the command window and pressing `enter`. Once that simulation has finished running, change its folder name to *Simulation_2_Data*. Now that we have a couple of simulations run, and have their corresponding data saved, we will next focus on visualizing each simulation's dynamics using the open-source software VisIt [41].

3.4. Guide: Visualizing the Spectral (FFT) Method's 'bubble3' Data

Here we will guide the user through visualizing the `bubble3` simulation data from Section 3.3. In particular we will visualize the magnitude of velocity data. Note that other Eulerian scalar data, like vorticity, the x- or y-component of velocity, etc., may be visualized analogously, as we are following the protocol from the previous section on data visualization, Section 3.2's *Visualizing the Eulerian scalar data*. To visualize the fluid's velocity vector field, please follow Section 3.2's steps for *To Visualize Velocity Vectors*.

To visualize the `uMag` data, do the following:

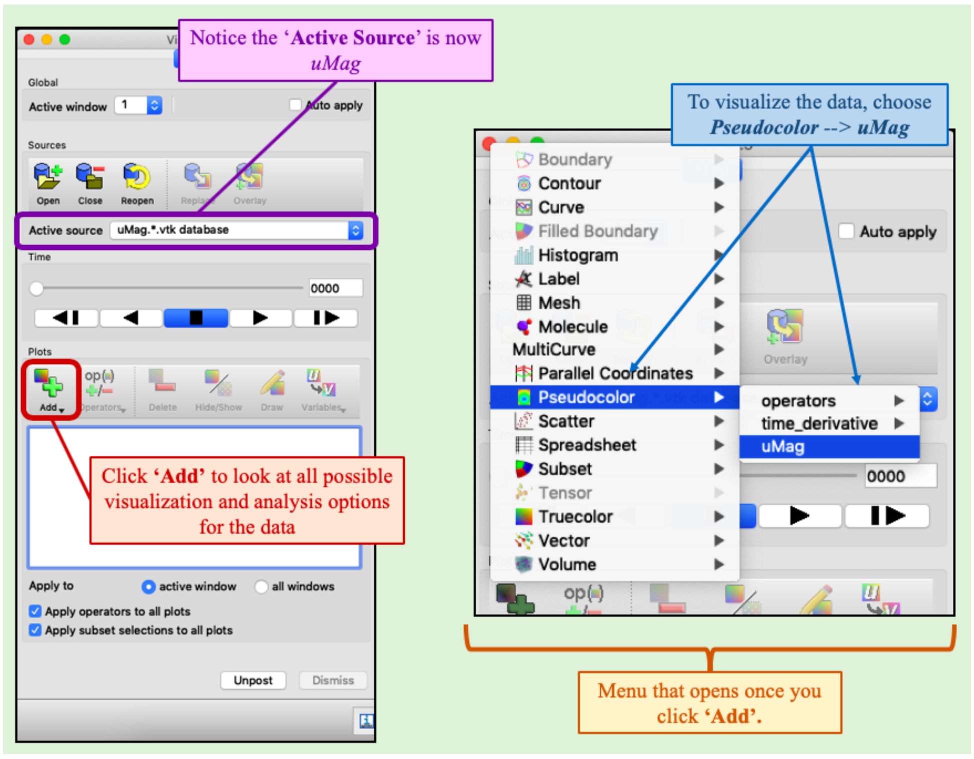
1. First we must open the desired magnitude of velocity data. Recall that the fluid's magnitude of velocity data was saved under the name *uMag*. Click `open` in the VisIt toolbar and in the dialog box that pops up, locate the desired data directory, here it is *Simulation_Data_1* from earlier, and select the `uMag` group. Figure 9 is provided as a visual aid for these steps.


Click `Open' to select what data you would like to visualize
Path to Current Data Folder (here in the *Simulation_1_Data* folder)
Choose what data you will visualize. (here selecting the fluid's magnitude of velocity)
Dialog box that pops up after you click 'Open'
Open
Directories
Files
Refresh
OK
Cancel

Host: localhost
Path: /Users/nickbattista/Documents/Holy_Grail/FFT_NS_Solver/MATLAB/Simulation_1_Data
Filter: *
Use "current working directory" by default
File grouping: Smart
. (current directory)
.. (go up 1 directory level)
uMag.*.vtk database (61 total files)
uMag.0000.vtk
uMag.0001.vtk
uMag.0002.vtk
uMag.0003.vtk
uMag.0004.vtk
...
uMag.0056.vtk
uMag.0057.vtk
uMag.0058.vtk
uMag.0059.vtk
uMag.0060.vtk
uX.*.vtk database (61 total files)
uX.0000.vtk
Open file as type: Guess from file name/extension

Figure 9. Opening specific simulation data to visualize in VisIt.

2. Although we have opened the data, nothing will appear visualized, yet. However, now `uMag` is the *Active Source* in the window, see Figure 10. We must now tell VisIt how to visualize the data. We will visualize magnitude of velocity as a background colormap, as it is a scalar quantity. To do this click 'Add' and then select `Pseudocolor` from the menu that appears, followed by `uMag`, i.e., `Add`→`Pseudocolor`→`uMag`.


Notice the '**Active Source**' is now *uMag*
Click '**Add**' to look at all possible visualization and analysis options for the data
To visualize the data, choose *Pseudocolor --> uMag*
Menu that opens once you click '**Add**'.
Open
Close
Reopen
Add
Boundary
Contour
Curve
Filled Boundary
Histogram
Label
Mesh
Molecule
MultiCurve
Parallel Coordinates
Pseudocolor
Scatter
Spreadsheet
Subset
Truecolor
Vector
Volume
operators
time_derivative
uMag

Active window: 1
Active source: uMag.*.vtk database
Apply to: active window
Apply operators to all plots
Apply subset selections to all plots

Figure 10. Opening specific simulation data to visualize in VisIt.

3. To finally visualize the data, click `Draw`. Window 1 will then show the fluid's magnitude of velocity data at time-point 0, i.e., the initial magnitude of velocity, see Figure 11.

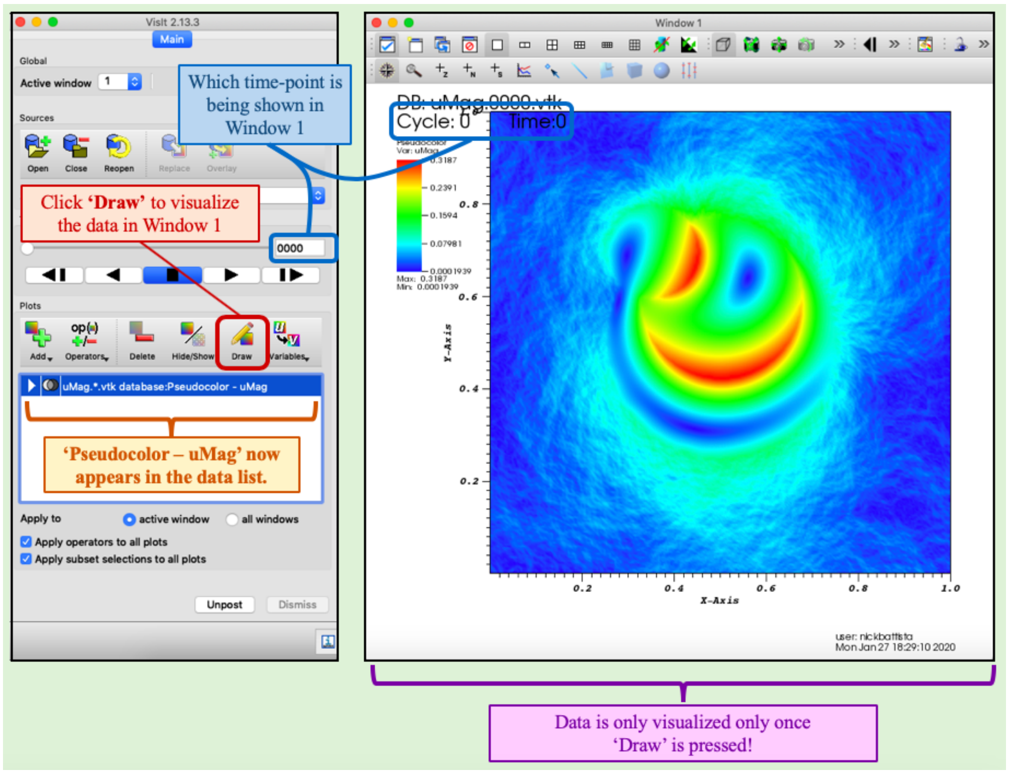

VisIt 2.13.3
Main
Which time-point is being shown in Window 1
Click 'Draw' to visualize the data in Window 1
0000
Draw
uMag.*.vtk database:Pseudocolor - uMag
'Pseudocolor – uMag' now appears in the data list.
Window 1
DB: uMag.0000.vtk
Cycle: 0
Time:0
Data is only visualized only once 'Draw' is pressed!

Figure 11. 'Drawing' the data in VisIt. The data shown in Window 1 here corresponds to time-point 0, as given from the *time slider* value.

4. Furthermore, we note that certain choices of colormaps are better for both conveying the data, but also for people who are colorblind. Generally, the default colormap, which was used in Figure 11, is a bad choice, as it is impossible for people who are red-green colorblind to interpret the field [57]. Thus, we will change the colormap to another, e.g., the `RdYlBu` (red-yellow-blue) colormap, by double-clicking on the `uMag.*.vtk database:Pseudocolor uMag` in the database list and appropriately selecting the desired colormap. Other properties may also be changed in this box, such as the colormap's scaling.

   Also, you can visualize the magnitude of velocity at the different time-points that were saved during the simulation by moving the time-slider. Figure 12 shows the data at time-point 30. Moreover, the colormap and its properties may be modified by double-clicking on `uMag.*.vtk database:Pseudocolor uMag` in the database list.

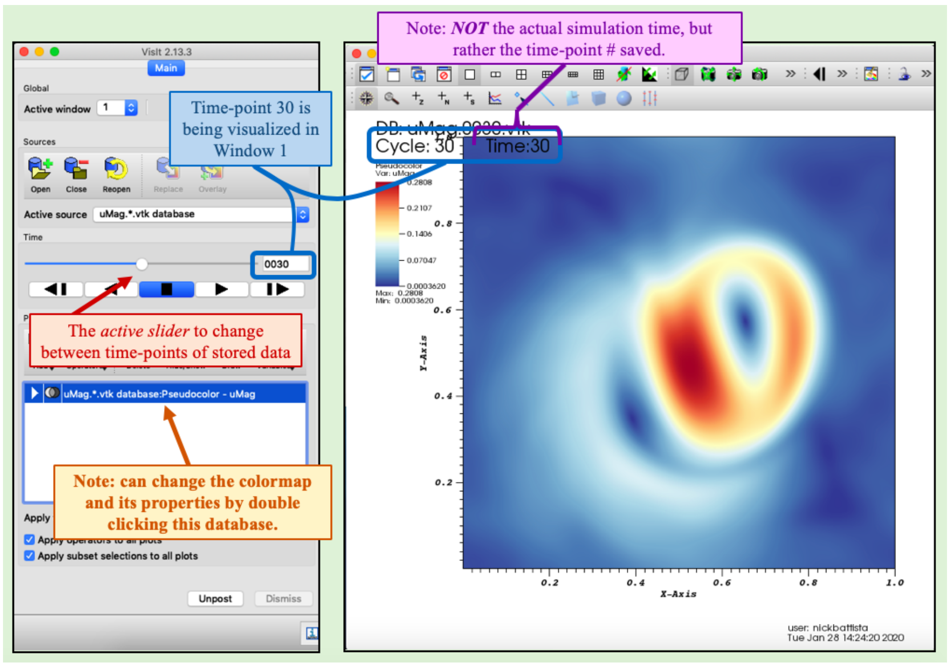

Note: *NOT* the actual simulation time, but rather the time-point # saved.
Time-point 30 is being visualized in Window 1
The *active slider* to change between time-points of stored data
Note: can change the colormap and its properties by double clicking this database.
VisIt 2.13.3
Main
Active window
Sources
Open
Close
Reopen
Replace
Overlay
Active source
uMag.*.vtk database
Time
0030
uMag.*.vtk database:Pseudocolor - uMag
Apply
Apply operators to all plots
Apply subset selections to all plots
Unpost
Dismiss
DB: uMag.0030.vtk
Cycle: 30
Time:30
X-Axis
Y-Axis
user: nickbattista
Tue Jan 28 14:24:20 2020

Figure 12. Changing the time-point to the 30th time-point stored.

5. Note that the time in the simulation does not generally correspond to the time-point stored. For example, in this case the simulation modeled 30 s of time and 60 total data time-points were stored. Thus, data was saved every 0.5 s of simulation time. Furthermore, you could repeat Steps 1–4 for data in *Simulation_2_Data* from the second simulation you ran. Instead of comparing the 30th time-point's data, we will compare the 60th, i.e., the last time-point. Repeating this entire process for the other simulation's data yields the comparison as shown in Figure 13, below.

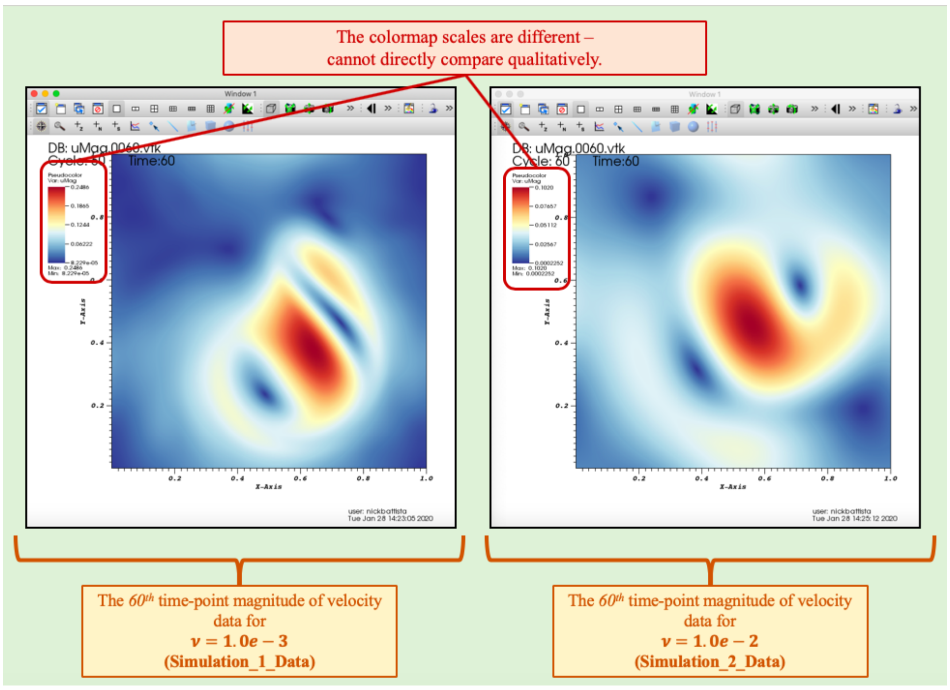
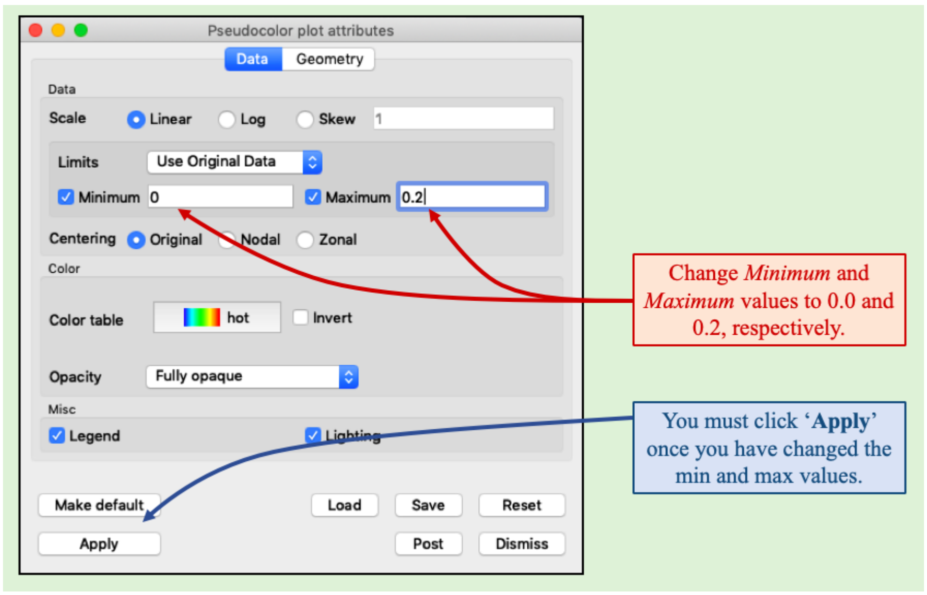

The colormap scales are different – cannot directly compare qualitatively.
DB: uMag.0060.vtk
Cycle: 60
Time:60
DB: uMag.0060.vtk
Cycle: 60
Time:60
The 60th time-point magnitude of velocity data for $\nu = 1.0e-3$ (Simulation_1_Data)
The 60th time-point magnitude of velocity data for $\nu = 1.0e-2$ (Simulation_2_Data)

Figure 13. Comparing the data between both simulations' last time-point (60th time-point stored). Note that a direct qualitative comparison is not possible because the scaling of each colormap is different.

However, we cannot qualitatively compare the magnitude of velocity between each of these simulations as they are in Figure 13. Each simulation's colormap has a different scale. In order to compare, we must make both scales equivalent.

6. To change the scale, double-click on each of the databases. This will bring up a dialog window (as described earlier) where you can change the scaling, e.g., minimum and maximum, as well as the colormap itself. Change the minimum and maximum values to 0.0 and 0.2, respectively. Figure 14 illustrates where to change the minimum and maximum for the colormap's scale. Once those values are changed, click `Apply`. Note that you must do this for both of the databases individually.


Pseudocolor plot attributes

Data | Geometry

Data

Scale: [✓ Linear] Log Skew 1

Limits: Use Original Data

[✓] Minimum 0 [✓] Maximum 0.2

Centering: [✓ Original] Nodal Zonal

Color

Color table: hot [ ] Invert

Opacity: Fully opaque

Misc

[✓] Legend [✓] Lighting

Make default Load Save Reset

Apply Post Dismiss

Change *Minimum* and *Maximum* values to 0.0 and 0.2, respectively.

You must click '**Apply**' once you have changed the min and max values.

Figure 14. Changing the magnitude of velocity's colormap scaling.

7. After changing the scale, the comparison is significantly different than when first visualized, see Figure 15.

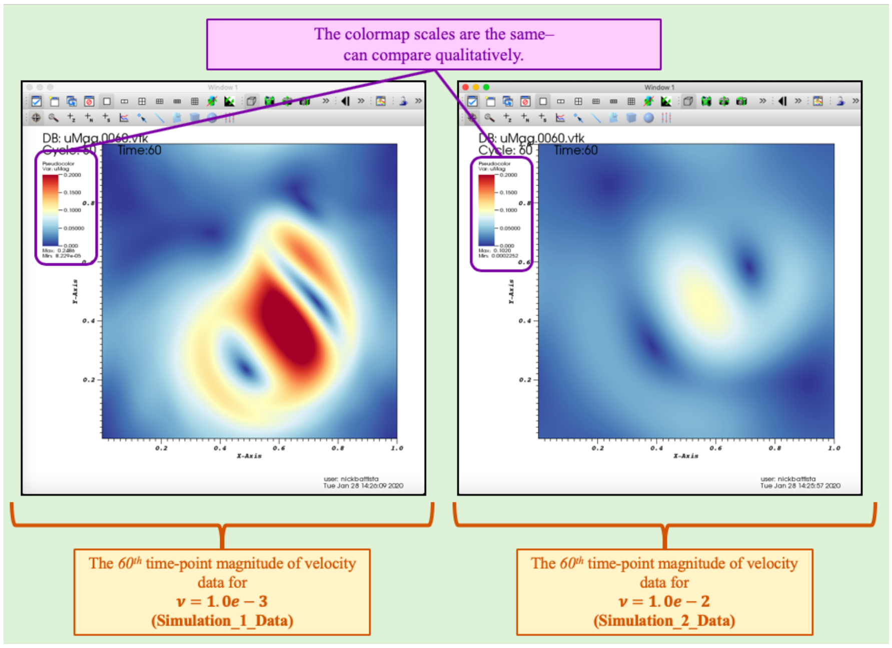

The colormap scales are the same–can compare qualitatively.
DB: uMag.0060.vtk
Cycle: 60 Time:60
Pseudocolor Var: uMag
0.2000
0.1500
0.1000
0.05000
0.000
DB: uMag.0060.vtk
Cycle: 60 Time:60
X-Axis
Y-Axis
user: nickbattista
The 60th time-point magnitude of velocity data for $\nu = 1.0e-3$ (Simulation_1_Data)
The 60th time-point magnitude of velocity data for $\nu = 1.0e-2$ (Simulation_2_Data)

Figure 15. Comparing each simulation's magnitude of velocity data at the 60th time-point. The scaling of each colormap is identical which leads to straight-forward direct qualitative comparison.

Furthermore, we also provide visualizations of other time-points and/or scalar data in Appendix C.

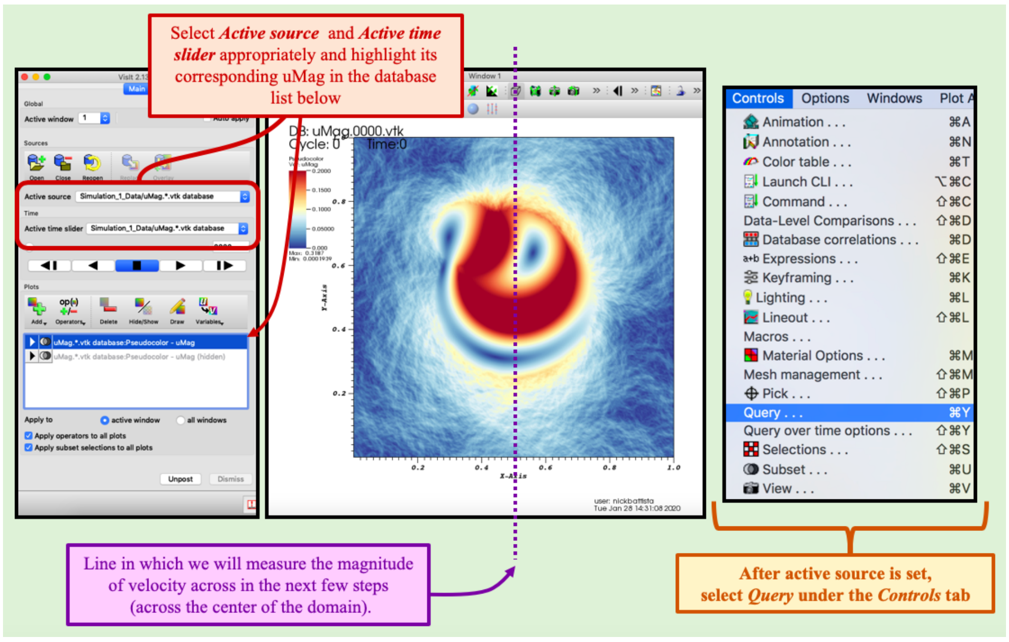
3.5. Guide: Analyzing the Spectral (FFT) Method's 'bubble3' Data

To analyze the data in VisIt, we will provide a walk-through of the steps that necessary to measure a fluid quantity (particular variable) across a line of interest within the computational domain using the Spectral (FFT) Method's `bubble3` example from Section 3.3. We will assume that you have visualized the data for the *magnitude of velocity* that corresponds to both of the simulations. In this tutorial we will compare the magnitude of velocity for both simulations across a vertical line through the center of the computational domain at the last time-point stored, i.e., the 60th time-point. See Figure 16 for an idea of where the measurement will take place.

To analyze the `uMag` data, do the following:

1. Make sure that *Active source* is the `uMag` data corresponding to the data from *Simulation_1_Data* in VisIt, see Figure 16.

   Once the *Active Source* has been appropriately selected, in the command bar, in the VisIt menu go to `Controls`→`Query` (see Figure 16).


Select ***Active source*** and ***Active time slider*** appropriately and highlight its corresponding uMag in the database list below

Line in which we will measure the magnitude of velocity across in the next few steps (across the center of the domain).

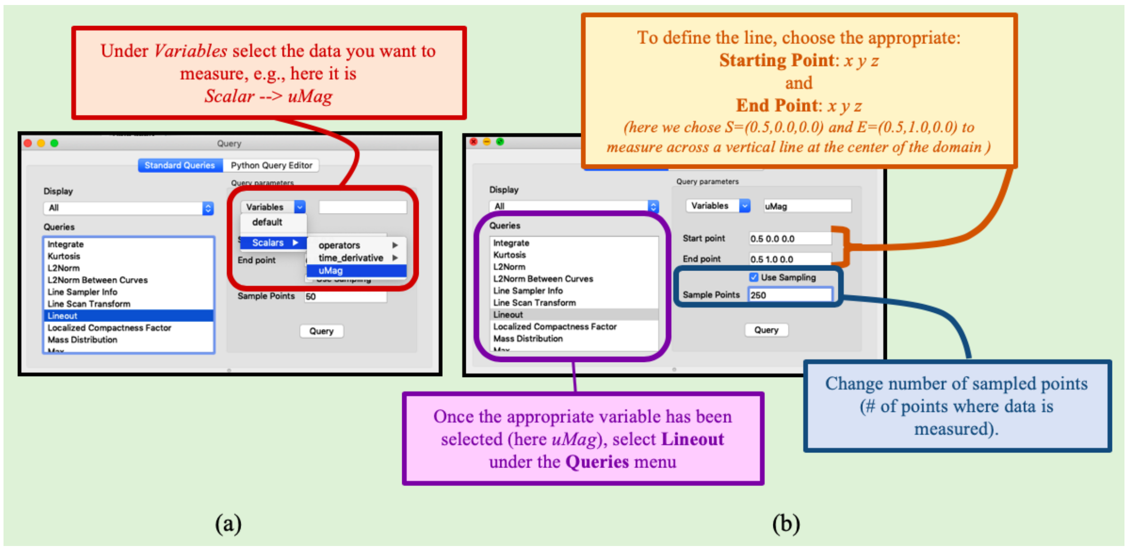
After active source is set, select *Query* under the *Controls* tab

Active source: Simulation_1_Data/uMag.*.vtk database

Active time slider: Simulation_1_Data/uMag.*.vtk database

Figure 16. To analyze data in VisIt, the *active source* and *active time slider* must be set to specific data that is desired to be analyzed, e.g., here it is the magnitude of velocity, `uMag`, for the *Simulation_1_Data*.

2. Next select `Lineout` under *Queries* and then under *Variables* select `Scalars` and then `uMag`, i.e., `Variables`→`Scalars`→`uMag`, see Figure 17.



Under *Variables* select the data you want to measure, e.g., here it is *Scalar --> uMag*

To define the line, choose the appropriate: **Starting Point**: *x y z* and **End Point**: *x y z* *(here we chose S=(0.5,0.0,0.0) and E=(0.5,1.0,0.0) to measure across a vertical line at the center of the domain)*

Once the appropriate variable has been selected (here *uMag*), select **Lineout** under the **Queries** menu

Change number of sampled points (# of points where data is measured).

Start point 0.5 0.0 0.0

End point 0.5 1.0 0.0

Use Sampling

Sample Points 250

Figure 17. Within the *Query* dialog box, select (**a**) the chosen data to measure, e.g., here it is `uMag`, and (**b**) the `Lineout` option to measure the data across a line within the domain that is defined by a starting and ending point.

3. Next we will define the line segment in which we intend to measure the data across. To do this the user gets to choose the starting point and ending point of the line segment, see Figure 17b. We will measure across a vertical line through the center of the domain, hence we choose $(x, y, z) = (0.5, 0.0, 0.0)$ and $(x, y, z) = (0.5, 1.0, 0.0)$ as the starting and ending point, respectively. We also will measure the value of `uMag` at 250 sampled points. Thus, change *Sample Points* to 250, see Figure 17b and select `Use sampling`. VisIt will automatically interpolate the data so that it is measured at number of equally-spaced sample points that is designated by the user.

4. Once you click `Query` a plot should pop up in a new *Active Window* in Visit, i.e., Active window 2, see Figure 18. If you double-click on the *Lineout(uMag)* bar in the VisIt database listing window, you can change various attributes about the line, such as its thickness or color among others. Moreover, you can use the time-slider to observe how the magnitude of velocity across the chosen line varies from time-point to time-points.

5. To get back to the colormap window, select Active Window 1, rather than 2.

6. You can repeat this procedure and overlay multiple `uMag` measured data on the same plot, each corresponding to a different simulation, e.g., repeat this process for *Simulation_2_Data*. Be sure to choose the correct data for the *Active source*.

7. If you repeat this entire process for the *Simulation_2_Data* and move the time-slider such that the data being shown in *Active window 2* corresponds to the last time-point stored (the 60th), you should see what is give in Figure 19. Also, note that we have changed each line's thickness individually.

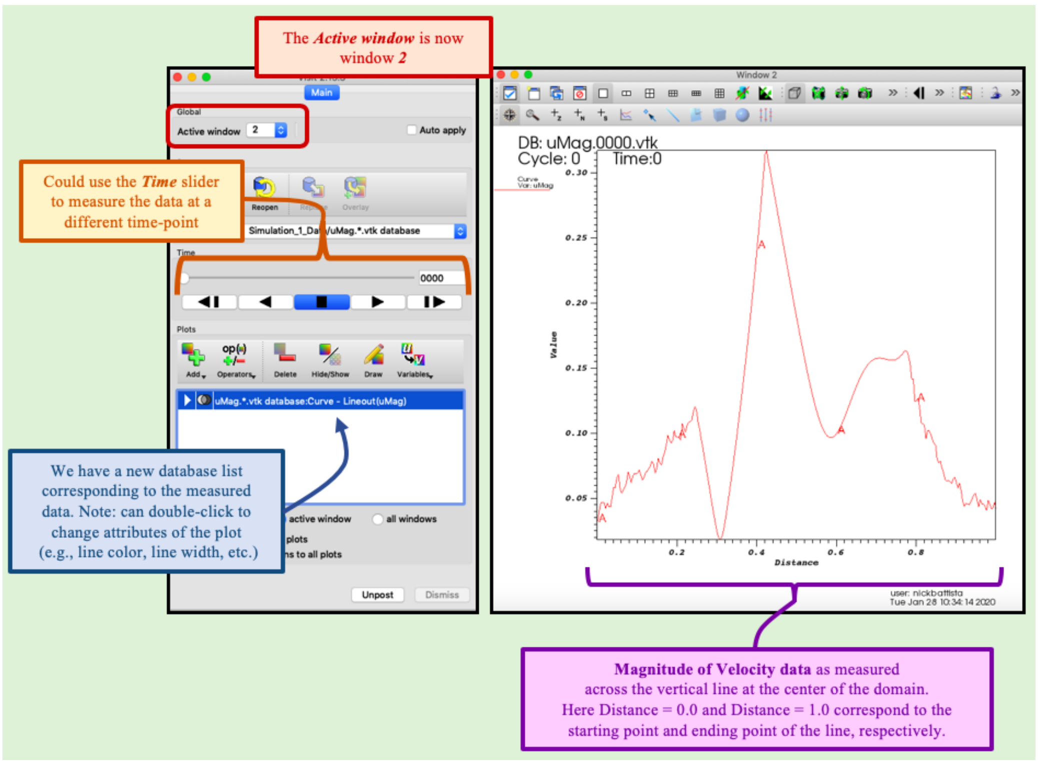

The *Active window* is now window *2*
Could use the *Time* slider to measure the data at a different time-point
We have a new database list corresponding to the measured data. Note: can double-click to change attributes of the plot (e.g., line color, line width, etc.)
Magnitude of Velocity data as measured across the vertical line at the center of the domain. Here Distance = 0.0 and Distance = 1.0 correspond to the starting point and ending point of the line, respectively.
Active window 2
Auto apply
Simulation_1_Data/uMag.*.vtk database
0000
uMag.*.vtk database:Curve - Lineout(uMag)
DB: uMag.0000.vtk
Cycle: 0 Time:0
Value
Distance
user: nickbattista
Tue Jan 28 10:34:14 2020

Figure 18. Illustrating the *Active window* change to window 2 as well as a plot of the magnitude of velocity (selected data) as measured across a vertical line at the center of the domain.

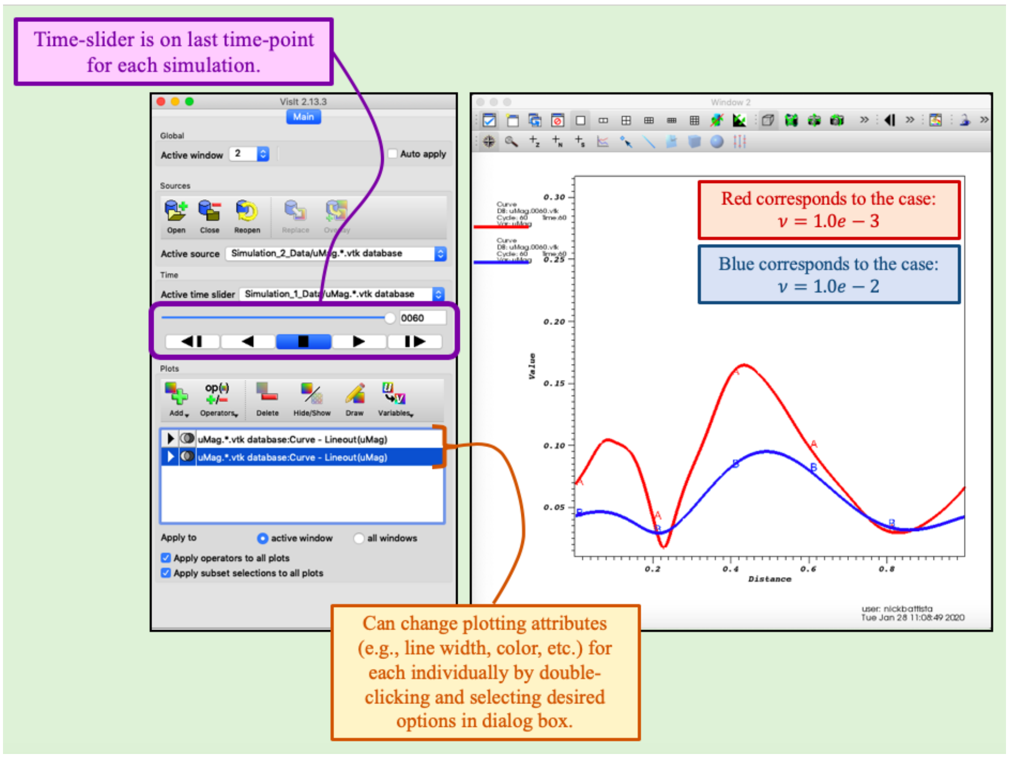

Time-slider is on last time-point for each simulation.
Red corresponds to the case: $\nu = 1.0e-3$
Blue corresponds to the case: $\nu = 1.0e-2$
Can change plotting attributes (e.g., line width, color, etc.) for each individually by double-clicking and selecting desired options in dialog box.
Active window 2
Active source Simulation_2_Data/uMag.*.vtk database
Active time slider Simulation_1_Data/uMag.*.vtk database
0060
uMag.*.vtk database:Curve - Lineout(uMag)
uMag.*.vtk database:Curve - Lineout(uMag)
Apply to active window all windows
Apply operators to all plots
Apply subset selections to all plots
Value
Distance
user: nickbattista
Tue Jan 28 11:08:49 2020

Figure 19. Comparing the measured magnitude of velocity data between both simulations at the last time-point data saved.

4. Built-in Examples (for Each Fluid Solver)

In this section we will highlight a subset of the built-in examples that can be run immediately upon download (or `git clone`) of the software. These examples were selected as they are either traditional problems in fluid dynamics or problems that could naturally lead to fruitful discussion and analysis of the underlying dynamics. The examples we will discuss are:

1. Lid-Driven Cavity Flow via the *Projection Method*
2. Circular Flow in a Square Domain via the *Projection Method*
3. Side-by-side Vorticity Region Interactions via the *Spectral (FFT) Method*
4. Evolution of Vorticity from an Initial Velocity Field via the *Spectral (FFT) Method*
5. Flow Past One or More Cylinders via the *Lattice Boltzmann Method*
6. Flow Past a Porous Cylinders via the *Lattice Boltzmann Method*

For each of these examples, we will provide the necessary simulation details, including a brief background about problem as well as the numerical parameters that were used in each simulation. Furthermore, we will also make observations regarding each simulation's dynamics, investigate some of the data, and suggest future ideas for how students could either modify or explore each example further.

These examples are available within both the MATLAB and Python implementations in the software. Note that Section 3 described how to run, visualize, and analyze the simulations, and thus the visualizations and analysis contained here can be recreated by students and other practitioners.

4.1. Cavity Flow (via Projection Method)

In this example we will use the Projection Method in the software to run a cavity flow problem. Note that this example is contained in the Projection Method folder and may be run by selecting the `'cavity_top'` option (see `line 58` in Figure 3a).

For this lid-driven cavity flow problem, the non-zero horizontal velocity on the top wall, $u = u_T$, is set to $u_T = 4.0$ m/s, while all other tangential (and normal) velocities along the boundaries are set to zero (see Figure 1a). Note that the top wall velocity is not immediately set at u_T, but rather the flow ramps up from 0 to the preset u_T along this wall, to avoid instantaneous acceleration artifacts. For the simulations shown below with $Re = 4000$, use the computational parameters listed in Table 3. The Reynolds Number was set by defining the characteristic length scale to be $L = L_x$, the horizontal length of the domain, and the characteristic velocity to be $V = u_T$, e.g., $Re = \frac{\rho L_x u_T}{\mu}$. To change the Reynolds Number, the dynamic viscosity was varied. For $Re = 4000, 400, 40$ and 4, we used $\mu = 1, 10, 100$, and 1000 $\text{kg}/(\text{m}\cdot\text{s})$, respectively. Note that for the cases with $Re = 4$ and 40, we decreased the time-step to $dt = 5\times10^{-5}$ s to ensure numerical stability of solutions. If we had not, errors would have been magnified every time-step of the simulation and/or the numerical solutions may have blown up. In CFD it is common that one may need to vary the time-step depending on the Re or other parameters of the system. Situations in which very small time-steps need to be taken to ensure numerical stability of solutions are known as *stiff equations*, for more information please see [43].

Table 3. Numerical parameters for the Cavity simulation for $Re = 4000$.

Parameter	Variable	Units	Value
Domain Size	$[L_x, L_y]$	m	$[1, 2]$
Spatial Grid Resolution	$[N_x, N_y]$		$[128, 256]$
Spatial Grid Size	$dx = dy$	m	$L_x/N_x = L_y/N_y$
Time Step Size	dt	s	10^{-3}
Total Simulation Time	T	s	6
Fluid Density	ρ	kg/m^3	1000
Fluid Dynamic Viscosity	μ	$\text{kg}/(\text{m}\cdot\text{s})$	1

Upon having run the cavity model for $Re = 4000$, we visualized its corresponding simulation data using VisIt [41], as illustrated by Figure 20, which gives colormaps of vorticity, magnitude of velocity, velocity in the horizontal direction throughout the domain, and pressure, as well as depictions of the velocity field, both as a scaled and non-scaled quantity. For the scaled velocity field snapshot, the scale factor was equal to the largest magnitude of velocity across the entire computational domain. In the non-scaled plot, streamlines are also given. Streamlines show the path that a passive particle would take in the flow at a particular moment in time. The data shown here was taken at $t = 6.0$ s, when the simulation ended.

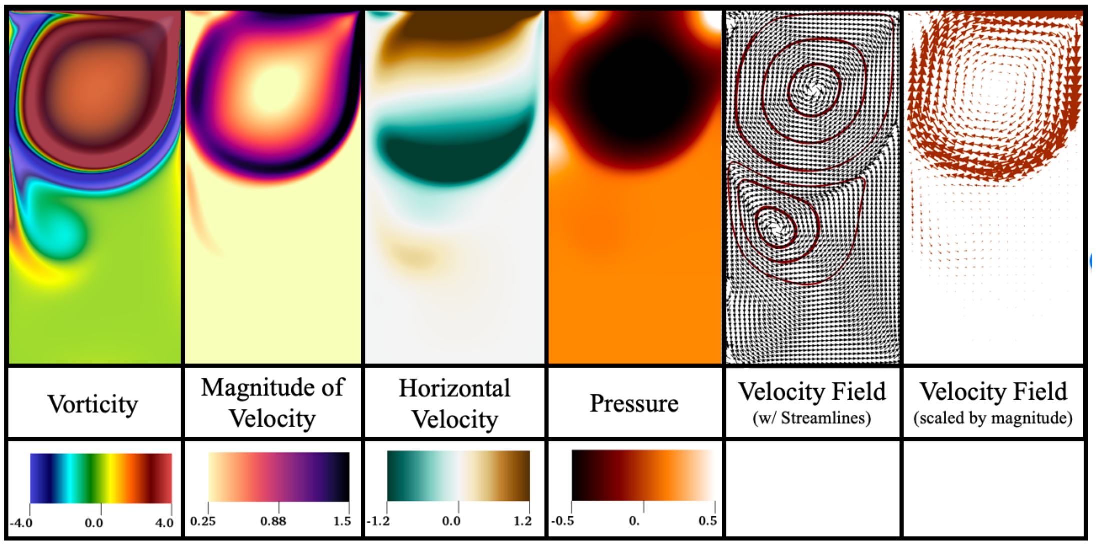


Figure 20. Illustration of all the simulation data produced from the projection method example of cavity flow (at $Re = 4000$). This snapshot was taken at 6.0 s, as the simulation ended.

A fully-formed vortex developed by that time near the top of the cavity, while a smaller vortex appears to be forming below (see Vorticity pane in Figure 20). Snapshots illustrating the formation of such vortices from this case of $Re = 4000$ are presented in Figure 21, which gives a colormap of vorticity and the velocity vector field. As fluid is moving along the top edge of the domain from left-to-right, the fluid nearest to the top begins moving in the same direction. Eventually fluid down in the cavity begins moving in the same direction as well, until it reaches the right boundary, where it must move downwards. It is easier for the fluid to move downward because of the faster flows moving towards the right above it, due to the boundary condition. As the fluid moves downward along the wall, fluid towards the middle of the cavity begins moving downwards as well. In tandem, with the fluid moving left-to-right along the top and top-to-bottom along the right, these fluid patterns initiate the formation of a clockwise-spinning vortex.

Now that there is a vortex spinning clockwise, it causes the fluid below the vortex (moving right-to-left) to begin moving right-to-left until it reaches the wall on the left and an analogous situation occurs to the above, except an oppositely-spinning (counter-clockwise) vortex begins to form. Recall that these vortices can both be traced back to the fluid moving along the top of the domain left-to-right. Within the region that the second vortex forms, much of the energy (velocity) has dissipated away, resulting in a smaller, less strong vortex.

We can explicitly measure the horizontal velocity descending down the cavity. This data is presented in Figure 22, which gives the horizontal velocity vs. depth in the cavity during four different snapshots in the simulation. We also measure the horizontal velocity across three different lines descending into the cavity. Near the surface of the cavity (at depth = 0 m), the velocity is equal to the boundary condition. As one descends into the cavity, there are spikes in both the positive and negative horizontal velocities. These spikes correspond to locations near the edges of the vortices, where there is faster moving fluid.

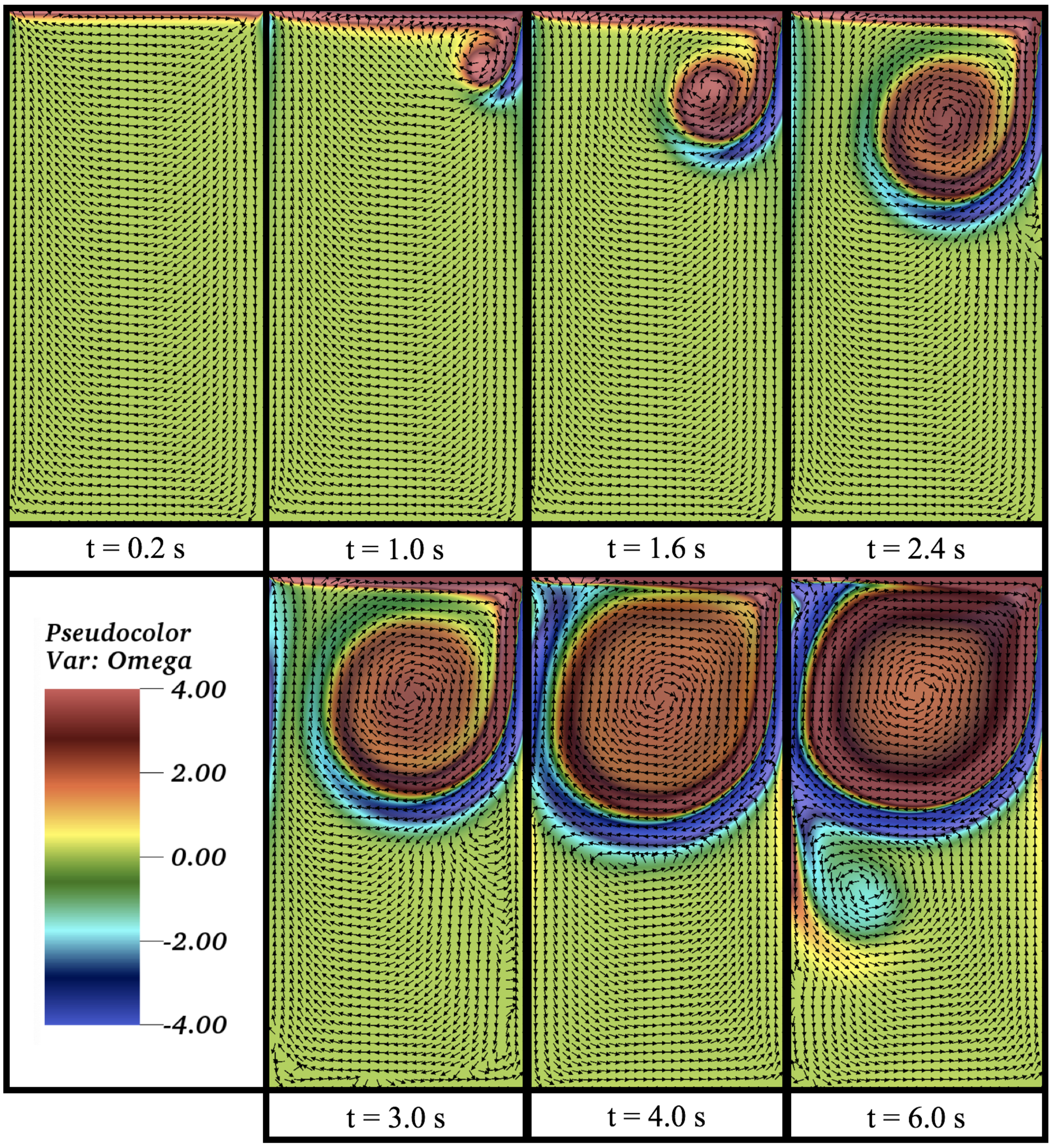


Figure 21. Snapshots showing the evolution of vortical flow patterns during a simulation with $Re = 4000$. The background colormap is of vorticity and the velocity vector field is also given.

Finally, one can change the Re by varying the dynamic viscosity, μ to observe how vortex formation changes within the cavity, as well as, how quickly energy gets dissipated in higher viscosity settings. Recall that if μ is changed to 40 or 4 the time-step, dt, was adjusted for as described above. Figure 23 illustrates how vortex formation changes within the cavity for different Re at three different snapshots during the simulations. Moreover, it gives the horizontal velocity vs. cavity depth, as measured across the middle of the cavity. Note that the $Re = 4$ and $Re = 40$ data virtually overlaps in the figure, and velocity (and system's energy) quickly dissipates to zero going further into the cavity.

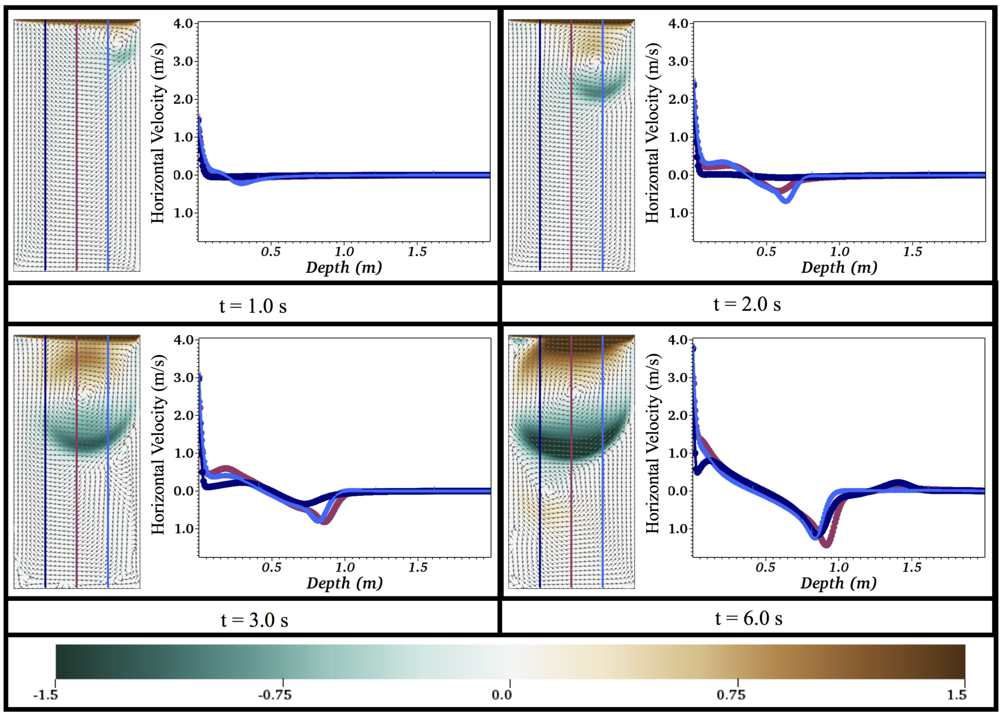


Figure 22. Snapshots of horizontal velocity measurements, when measured down the cavity in three places for $Re = 4000$.

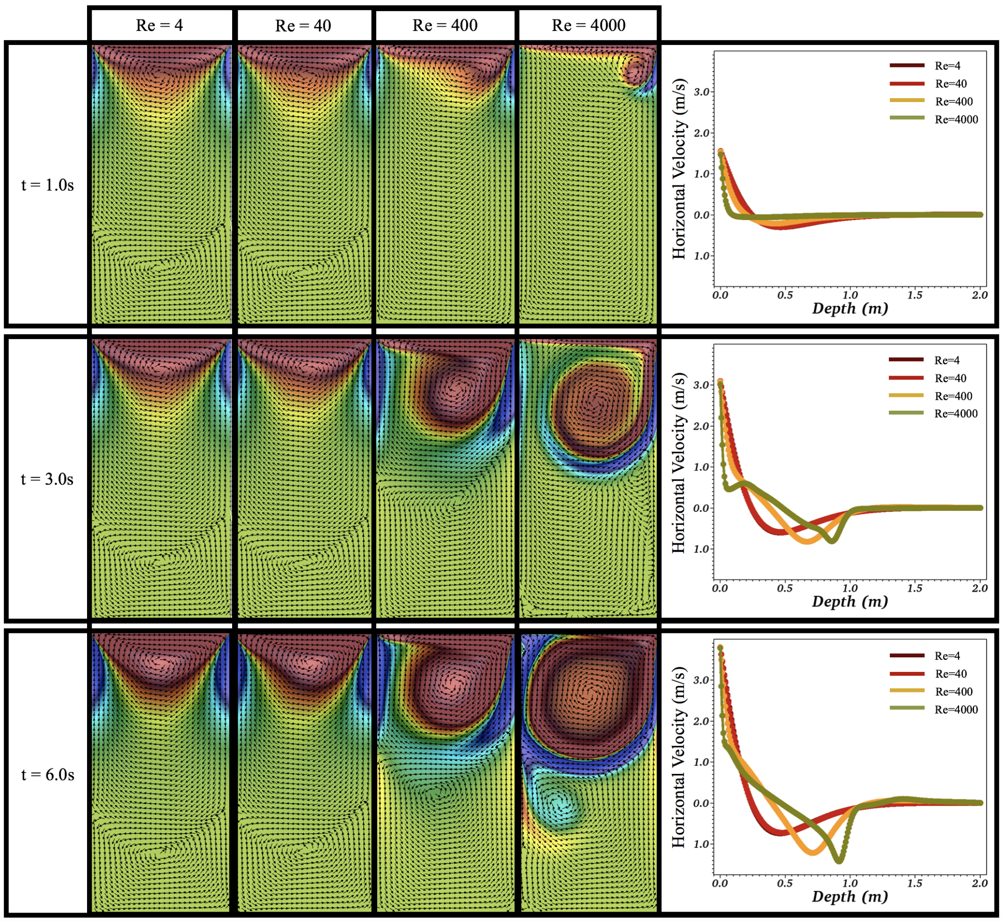


Figure 23. A comparison of vorticity, velocity field, and horizontal velocity measurements down the center of the cavity between cases of $Re = 4, 40, 400$, and 4000.

Students may elect to try the following:

1. Recreate the above results for cavity flow with the geometric and simulation parameters given in Table 3 for one or more Re.
2. Change the domain width (cavity width) to observe differences as the cavity gets wider for a given Re.
3. Vary the lid-velocity to observe differences (e.g., varying Re for different velocities as opposed to differing viscosities, μ).

4.2. Circular Flow in a Square Domain (via Projection Method)

In this example we will use the Projection Method in the software to run an example involving circular flow in a square domain. Note that this example is in the Projection Method folder and may be run by selecting the `'whirlwind'` option (see `line 58` in Figure 3a).

In this case, all the tangential boundary conditions along the computational domain were set to be $U = u_{top} = -u_{bot} = v_{left} = -v_{right} = 1.0$ m/s, see Figure 1b. As mentioned in Section 4.1 above, the tangential wall velocities are not immediately set to U, but rather the flow ramps up along each edge from 0 to the preset $u_{top}, u_{bot}, v_{left}$, or v_{right} along this wall, to avoid instantaneous acceleration artifacts. For the simulation shown below with $Re = 4000$, use the computational parameters listed in Table 4. Note that as $Re = \frac{\rho L U}{\mu}$, with $L = L_x = L_y$ and $U = 1.0$ m/s, one can vary the dynamic viscosity, μ, to change the Re. If you decide to make Re smaller (e.g., increasing μ), you may also need to decrease the time-step, dt, significantly to ensure numerical stability, as mentioned in Section 4.1. The requirement that very small time-steps are necessary to ensure numerical stability denotes what are called *stiff equations*. For more information on stiff equations, please see [43].

Table 4. Numerical parameters for the whirlwind simulation for $Re = 4000$.

Parameter	Variable	Units	Value
Domain Size	$[L_x, L_y]$	m	$[1, 1]$
Spatial Grid Resolution	$[N_x, N_y]$		$[256, 256]$
Spatial Grid Size	$dx = dy$	m	$L_x/N_x = L_y/N_y$
Time Step Size	dt	s	10^{-3}
Total Simulation Time	T	s	24
Fluid Density	ρ	kg/m^3	1000
Fluid Dynamic Viscosity	μ	$\text{kg}/(\text{m} \cdot \text{s})$	0.25

Having run the circular flow simulation for $Re = 4000$, we visualized its corresponding simulation data using VisIt [41], see Figure 24, which gives colormaps of vorticity, magnitude of velocity, velocity in the horizontal and vertical directions, and the finite-time Lyapunov exponent (FTLE). The FTLE can be used to find the rate of separation in the trajectories of two infinitesimally close parcels of fluid. Maxima in the FTLE have been used to determine approximations to the true Lagrangian Coherent Structures (LCSs). LCSs are used to determine distinct flow structures in the fluid [58–61] can be used to divide the fluid's complex dynamics into distinct regions to better understand transport properties of flow [62–64]. True LCSs would require knowledge of the infinite-time Lyapunov exponent (ITLE); however, the FTLE can serve as a proxy to the ITLE. In this paper, we computed the forward-time FTLE field, whose maximal ridges give approximate LCSs corresponding to regions of repelling fluid trajectories and whose low values give rise to regions of attraction [61]. Thus, the FTLE can be used to help find regions of high fluid mixing [58,59,61,65,66].

Figure 24 also provides contours for each quantity. It also includes a depiction of the fluid velocity field, as scaled by the maximum in the entire domain's magnitude of velocity, which also includes its associated streamlines. Recall that streamlines are curves that depict instantaneous tangent lines along direction of the flow velocity. They show the direction that a neutrally-buoyant particle would travel at a particular snapshot in time. These are different than contours (also known as level-sets or isolines), which are curves where a function has a constant value. This snapshot was taken at $t = 24.0$ s, when the simulation ended.

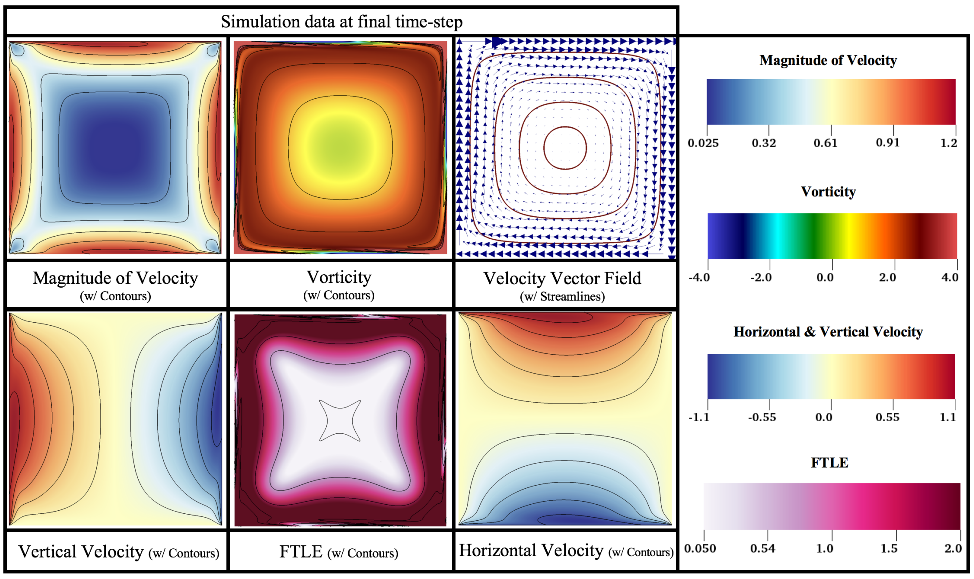


Figure 24. Illustration of some of the data produced for the circular flow example, using the projection method. The data shown is from the final time-step for $Re = 4000$.

As the velocity boundary conditions continually increased, the interior of the domain began to move clockwise, in the same direction as the flow at the boundaries (see Figure 1b). The colormaps and contours in each velocity-related panel reveal that flow speeds decrease towards the interior of the domain. Moreover, as all the boundary conditions are uniform, the vertical and horizontal velocity plots are identical under a 90 degree rotation. Also, the vorticity plot illustrates that vorticity is still low by the ending of the simulation in the center of the domain, even though flow across the whole domain is rotating clockwise. The FTLE plot suggests that the majority of fluid mixing occurs near the edge of the domain, rather than the interior, as higher values of FTLE suggest nearby fluid trajectories move away from each other at higher rates. This is due to larger spatial gradients in the velocity in these regions, where flow is moving at different speeds in different directions.

We also present data comparing simulations for $Re = 400, 1000$, and 4000 (for $\mu = 2.5, 1.0$, and $0.25\ \mathrm{kg}/(\mathrm{m}\cdot\mathrm{s})$, respectively). Figure 25 illustrates colormaps of the magnitude of velocity (with corresponding contours) at various time-points during the simulation. Every colormap uses the same scaling in the figure. Higher velocities appear to creep into the interior of the domain quicker in the lower Re cases. By $t = 24$ s it appears that the $Re = 400$ and 1000 cases look almost identical, the $Re = 4000$ case tells a different story - the interior of the domain is still moving considerable slower than the other two cases. This might seem counter-intuitive at first, as one might generally think that higher Re tends to lead to more fluid motion, especially when we are lowering viscosity to increase Re. However, this is due to differences in time-scales for the diffusion of momentum through the fluid. The viscous diffusion time can be thought of as the time it takes for a fluid parcel to diffuse a particular distance on average. If $\tilde{t}$ is the diffusion time, $\nu = \mu/\rho$ is the kinematic viscosity, and $\tilde{l}$ is the mean-squared distance, then

$$\tilde{t} \sim \frac{\tilde{l}}{\nu} \sim \frac{1}{\nu},$$

if all of the simulation geometries are identical and only μ (and hence ν) is varied. This concept is based off viewing diffusion as a random walk process [67]. Hence for the simulations presented here the time-scales vary between

$$\tilde{t}_{Re=400} \sim \frac{1}{2.5/1000} = 400\text{ s} \quad \text{and} \quad \tilde{t}_{Re=4000} \sim \frac{1}{0.25/1000} = 4000\text{ s}.$$

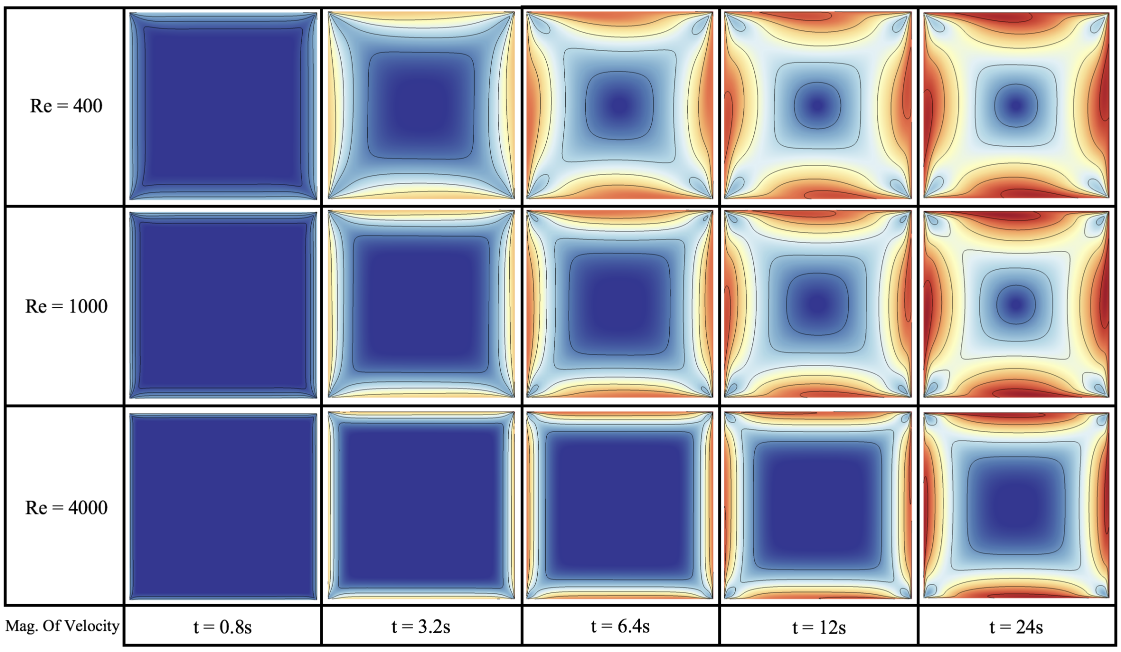


Figure 25. Snapshots showing the evolution of the magnitude of velocity between cases of $Re = 400, 1000$, and 4000.

Therefore in the $Re = 4000$ case, the diffusion time-scale is $10x$ longer than that of the $Re = 400$ case, and thus the dynamics evolve much slower in the $Re = 4000$ case! Moreover, this phenomena is shown in Figure 26, which illustrates the fluid velocity field along with its associated streamlines at the same snapshots in time.

This is more apparent if we compare flow profiles within the domain. Figure 27 compares the horizontal flow profile for $Re = 400, 1000$, and 4000 across the vertical mid-line of the domain. As mentioned above the dynamics in the $Re = 4000$ case evolves about $4x$ and $10x$ slower than the $Re = 1000$ and $Re = 400$ cases, respectively. By 24.0 s, the flow profiles in the $Re = 400$ and $Re = 1000$ cases look almost identical, while the flow profile in the $Re = 4000$ resembles that of the the flow profile in the $Re = 1000$ at $\sim$6.4 s—approximately a factor of 4 different in time. Furthermore Figure 28 shows the flow profiles for all three cases of Re when the diffusion of momentum has reached the same depth. This occurs at different simulation times due do the variations in viscous diffusion time-scales.

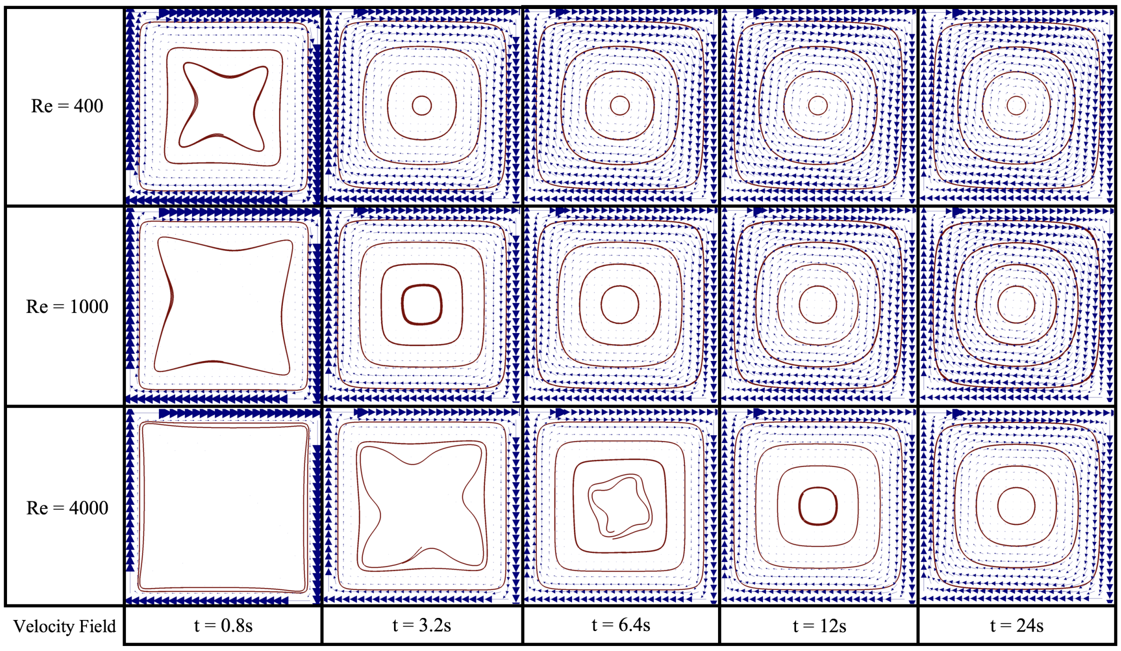


Figure 26. Snapshots showing the velocity field's evolution between cases of $Re = 400, 1000$, and 4000. Streamlines of the velocity field are also illustrated.

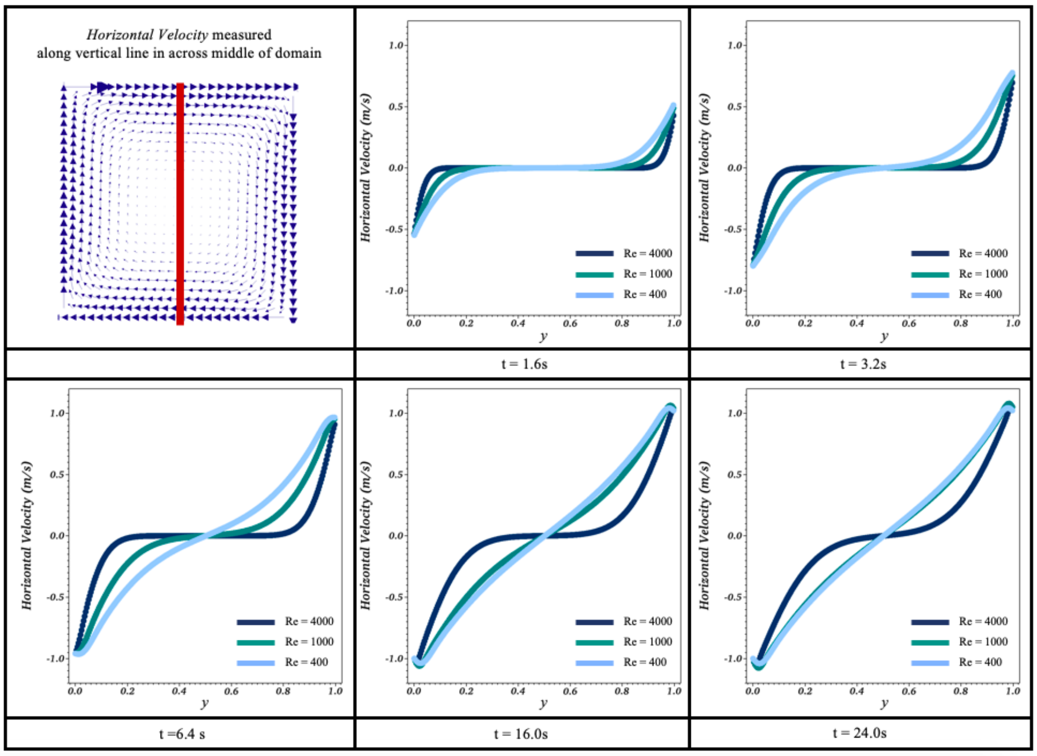


Figure 27. Snapshots showing horizontal velocity measurements across a vertical line centered in the domain among cases for $Re = 400, 1000$, and 4000.

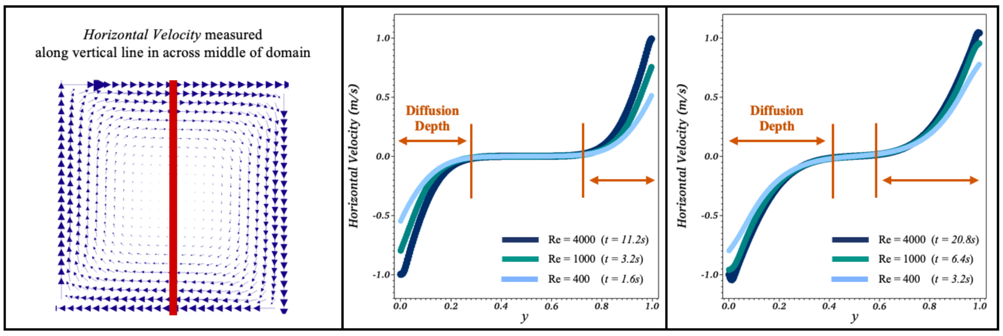


Figure 28. Snapshots showing horizontal velocity measurements across a vertical line centered in the domain among cases for $Re = 400, 1000$, and 4000 in which similar momentum diffusion depths have been reached.

Students may elect to try the following:

1. Recreate the above results with the parameters listed in Table 4
2. Change the computational geometry, e.g., rectangular instead of square
3. Vary the input velocity along each side of the domain to make them non-uniform
4. Vary the *Re* by changing dynamic viscosity for suggestion (2) or (3)

4.3. Side-by-Side Voritices (via Spectral Method)

In this example we will use a FFT-based fluid solver in the software to run a simulation of multiple regions of vorticity interacting with one another. Note that the first example given here can be run by going into in the *FFT_NS_Solver* script selecting the `'qtrs'` option (see `line 47` in Figure 3b). The vorticity is initialized as in Figure 2a. We will also highlight the `'half'` option in this section as well.

Recall that for this solver, an initial vorticity configuration is needed. For the simulations of four interacting vorticity regions, the *CW* and *CCW* vorticity were initialized as $\omega = -1$ and 1 rad/s, respectively. In other cases with only two interacting voricity regions, the strengths of each regions were varied between $-1, -0.5, 0.5$ and 1 rad/s, as appropriate. For the simulations shown below with either 2 or 4 interacting vorticity regions, use the computational parameters listed in Table 5.

Table 5. Numerical parameters for the 4 interacting vortices simulation.

Parameter	Variable	Units	Value
Domain Size (4 case)	$[L_x, L_y]$	m	$[1, 1]$
Spatial Grid Resolution (4 case)	$[N_x, N_y]$		$[512, 512]$
Spatial Grid Size	$dx = dy$	m	$L_x/N_x = L_y/N_y$
Domain Size (2 case)	$[L_x, L_y]$	m	$[1, 0.5]$
Spatial Grid Resolution (4 case)	$[N_x, N_y]$		$[512, 256]$
Spatial Grid Size	$dx = dy$	m	$L_x/N_x = L_y/N_y$
Time Step Size	dt	s	10^{-2}
Total Simulation Time	T	s	5
Fluid Kinematic Viscosity	$\nu = \rho/\mu$	m^2/s	0.001

Figure 29 provides the simulation data for the case with 4 interacting vorticity regions, as described above. It presents colormaps for vorticity, magnitude of velocity, horizontal velocity, vertical velocity, and the finite-time Lyapunov exponent (FTLE), which is used to determine approximations to the true Lagrangian Coherent Structures (LCS) to which can be used to distinguish fluid mixing regions [58,59,61,65,66]. Their corresponding contours are also given. Recall that contours are curves to which the value of a quantity is constant. Figure 29 also gives a snapshot of the velocity vector field. This data is from the last snapshot of the simulation at $t = 5.0$ s.

The top two and bottom two vorticity regions are initialized the same, e.g., clockwise (CW) and counterclockwise (CCW), respectively, by this point in the simulation they are beginning to lean in the direction of rotation. Moreover, it can be seen that fluid is moving quickly horizontally through the top, middle, and bottom of the domain, as illustrated by the magnitude of velocity plot. When this is compared to the horizontal velocity plot, one observes that the fluid along the top and bottom of the domain both are moving left-to-right, while fluid moving horizontally through the middle of the domain is moving right-to-left. These directions align with the direction of the spinning vortices in each of these horizontally-aligned regions—top, middle, and bottom, which can be seen in the velocity vector field depiction.

The high values of FTLE illustrate regions of higher fluid mixing, as higher FTLE suggest faster rates of separation of close fluid blobs. In Figure 29 these are the areas between the top-two and bottom-two vorticity regions. Within these regions there is intense shearing among the vertical flow velocity, see the vertical velocity plot. Red regions in the vertical velocity plot indicate fluid moving upwards while blue corresponds to downward fluid motion. Moreover, from the nature of the FFT-solver, the periodic boundary conditions, illustrate that along vertical walls of the domain on opposite sides there are regions where fluid is moving the the opposite direction. Hence the large FTLE values on the left and ride side of the top-two and bottom-two vortices. There is significantly less mixing in the top, middle, and bottom horizontal strips across the domain because fluid is generally moving unidirectionally in these regions with lower spatial gradients.

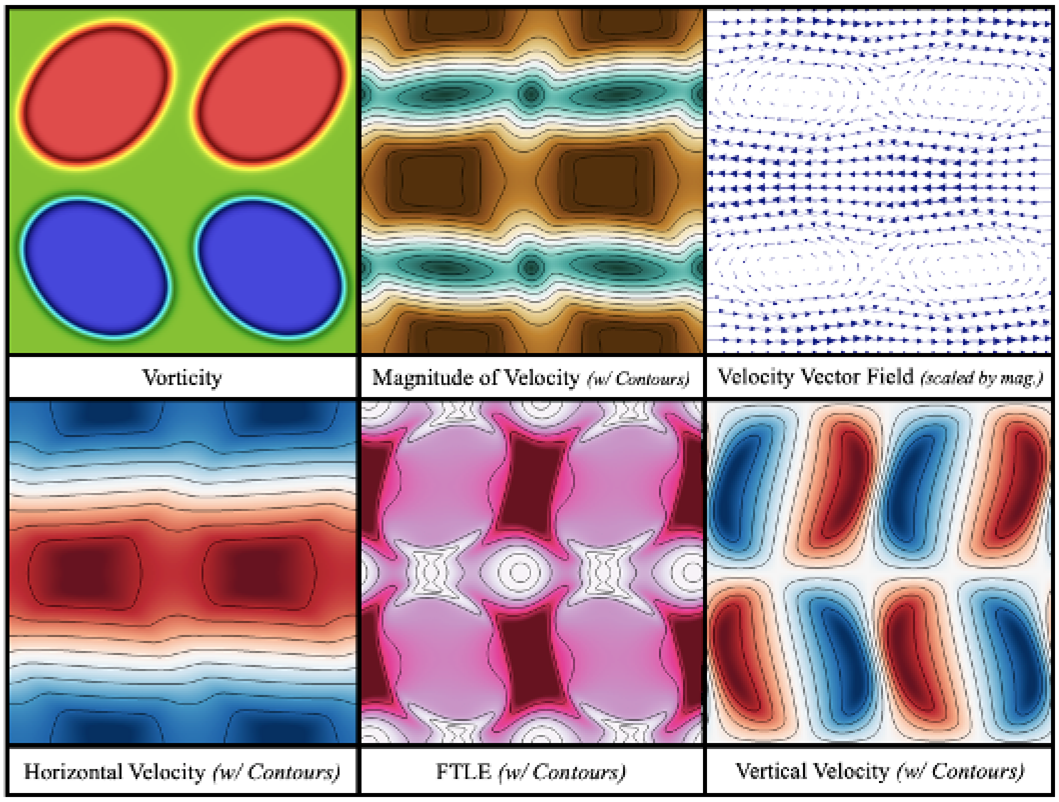


Figure 29. Illustrations of the simulation data produced during the case of 4 interacting vorticity regions. The data shown is from the last time-step.

One could elect to change the size or strength of a subset of these vorticity regions to simulate asymmetric vorticity interactions in this configuration. However, rather than study 4 interacting vorticity regions, we compared 2 interacting regions at a time, and varied the strength (magnitude of vorticity), size, and initial vorticity values (spinning-direction) between them, see Figure 30. Figure 30 provides comparison data of snapshots at $t = 6.0$ s in each simulation for different quantities—either vorticity with velocity vectors, magnitude of velocity with its corresponding contours, and FTLE with its corresponding contours. We use the language that a "strong" vorticity region has a $|\omega| = 1$ rad/s, while a "weaker" region has a vorticity of $|\omega| = 0.5$ rad/s. Each initial size of the vorticity regions had a radius of 0.15 m or 0.3 m.

In general, one can notice the following:

1. When same-spinning and size vorticity regions interact, more fluid mixing occurs directly between them (1st row)
2. When oppositely-spinning and same size vortex regions interact, there is enhanced vertical fluid flow between the vorticity regions, but less mixing between them (2nd row)
3. When oppositely-spinning vorticity regions of different strengths interact, the stronger region (whose magnitude of vorticity is greater), influences fluid flow to move in the same direction further away from it. There is still some enhanced vertical flow through the region between the regions (3rd row)
4. If there are asymmetric region sizes and strengths, where the larger region is weaker (smaller magnitude of vorticity), the region with the larger vorticity magnitude may be able to keep the other region from influencing much of the flow around it, while imposing its flow direction on the weaker region (4th row)
5. Two regions of vorticity with the same sign, but differing strengths, leads to enhanced fluid mixing between the vortices, while flow around the smaller region are significantly less than the other (5th row)
6. If there are asymmetric region sizes but uniform strengths, enhanced vertical flow will be seen between the regions, although the larger vorticity region exerts more influence on flow patterns closer to the smaller region (6th row)
7. If there are asymmetric region sizes and strengths, where the smaller region is weaker (smaller magnitude of vorticity), enhanced vertical flow will be seen between the regions, although the stronger vorticity region exerts more influence closer to the smaller region (7th row)

Students may elect to try the following:

1. Recreate the above results for interacting side-by-side vorticity regions with the geometric and simulation parameters given in Table 5 or for different ν or computational domains.
2. Add more vorticity regions or make the case with four asymmetric in terms of placement and/or size.
3. Change the distance between the vorticity regions to observe how magnitude of velocity or FTLE contours change.

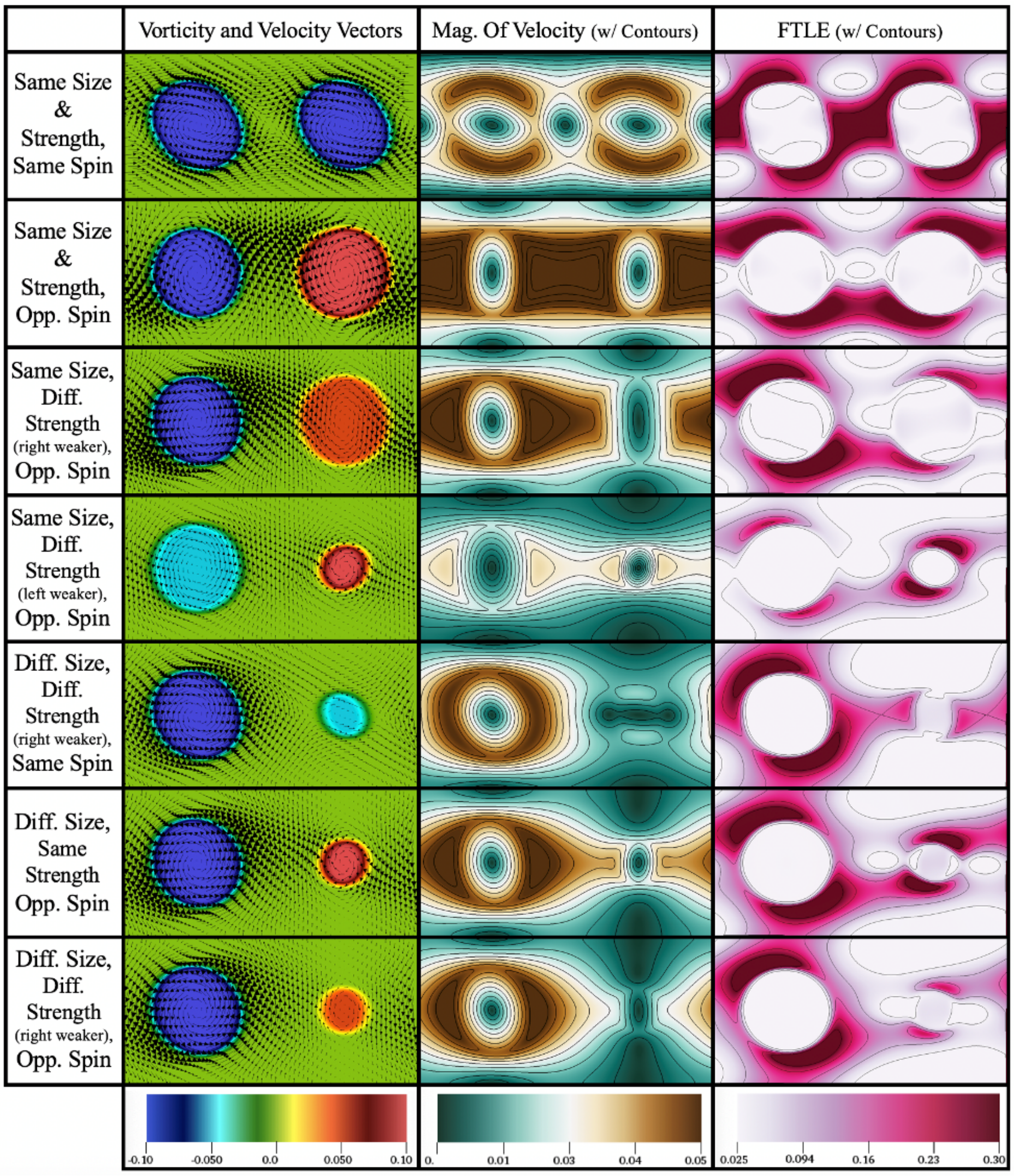


Figure 30. Vorticity and velocity field (**left column**) and magnitude of velocity with its associated contours (**right column**) for cases of two vorticity regions with different sizes, strengths, and vorticity initialization.

4.4. Evolution of Vorticity from an Initial Velocity Field

In this example using a spectral (FFT) solver, we begin with a snapshot of the velocity vector field from an independent CFD simulation, see Figure 31a,b. Note that this snapshot was taken from an open-source fluid-structure interaction example of a pulsing cartoon heart [36]. We used a stored time-point's velocity field data, to which we extracted the horizontal and vertical velocity components, and then computed its associated vorticity at that particular time-step, see Figure 31c. We note that there are numerical errors in computing the vorticity, ω, as a centered finite difference scheme [43] was used to compute each first-order partial derivative in calculating vorticity, i.e., $\omega = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$. Moreover, since this fluid solver uses periodic boundary conditions, we computed the partial derivatives at each boundary using data from the opposite side of the computational domain. Note this example may be run by selecting the `'jets'` option in the *FFT_NS_Solver* script (see `line 47` in Figure 3b).

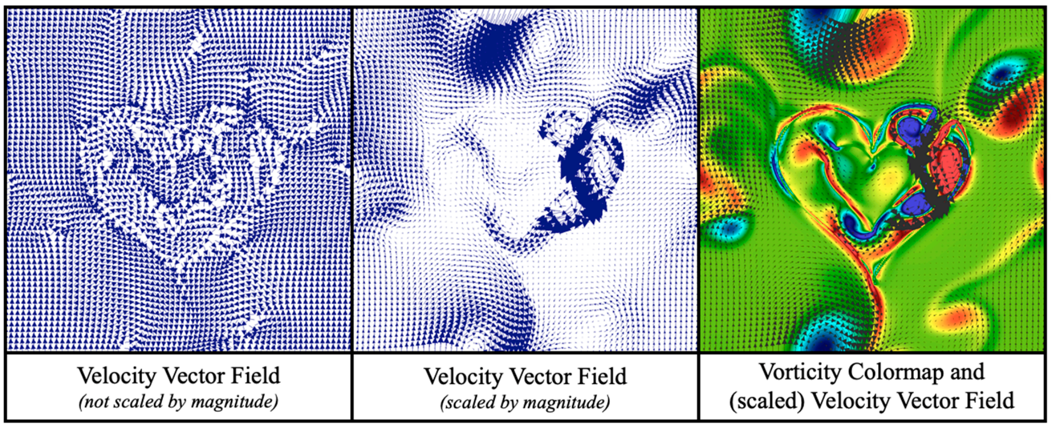


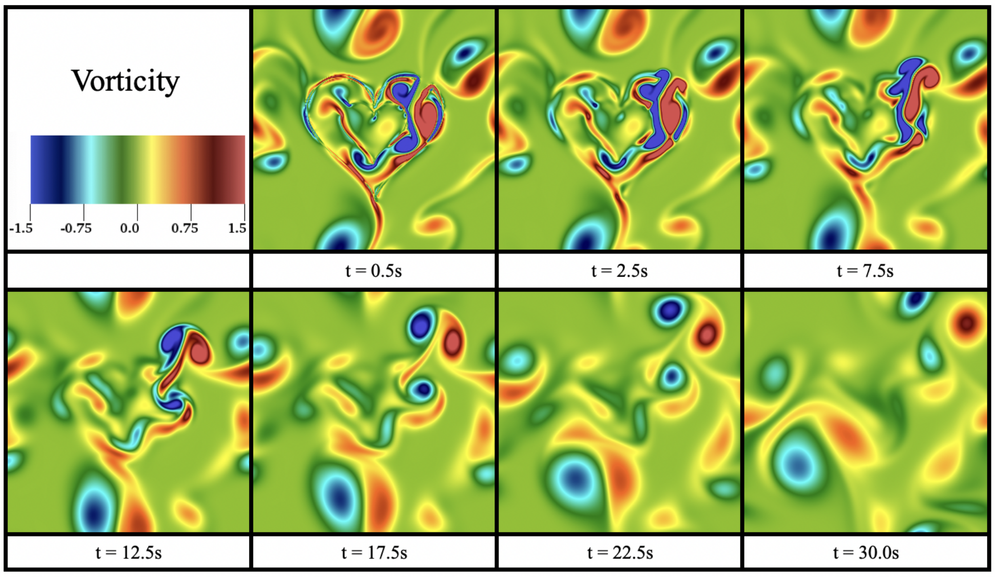
Figure 31. (**a**,**b**) Unscaled and scaled velocity vector field, respectively, that was used to define the initial fluid vorticity (**c**).

After having computed the vorticity from a velocity field, the simulation could begin. This simulation used the computational parameters listed in Table 6. We further note that the velocity field had an original grid resolution of $[N_x, N_y] = [512, 512]$, so we down-sampled it to a $[256, 256]$ grid for the example shown here. Furthermore, as the original velocity field was computed from an independent fluid-structure simulation, we comment that there is no complex, moving boundary within the computational domain; however, at the initial point you can see the remnants of the coinciding heart structure, see Figure 31.

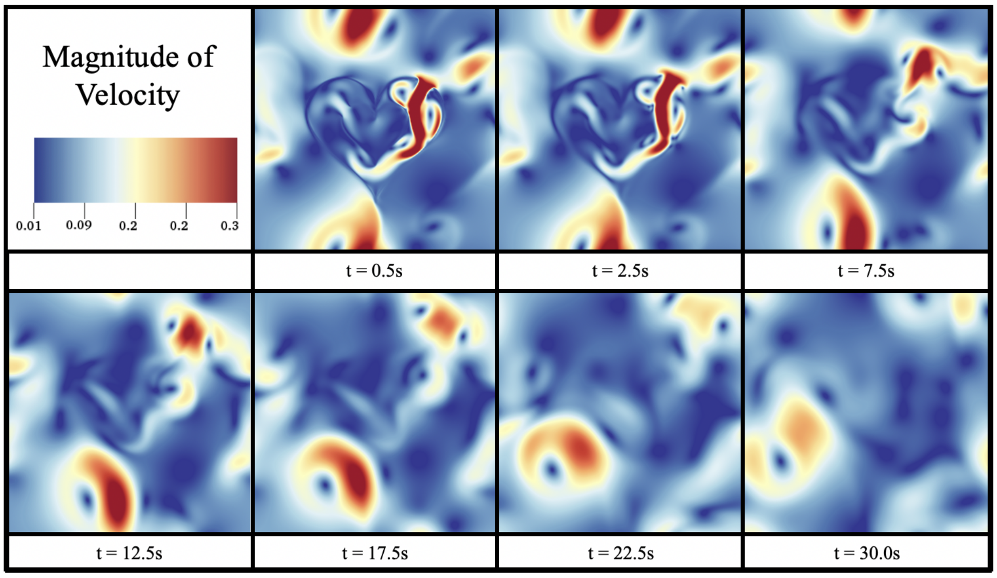
Table 6. Numerical parameters for case of evolving vorticity from an initial velocity field.

Parameter	Variable	Units	Value
Domain Size	$[L_x, L_y]$	m	$[1, 1]$
Spatial Grid Resolution	$[N_x, N_y]$		$[256, 256]$
Spatial Grid Size	$dx = dy$	m	$L_x/N_x = L_y/N_y$
Time Step Size	dt	s	10^{-2}
Total Simulation Time	T	s	50
Fluid Kinematic Viscosity	$\nu = \mu/\rho$	m^2/s	0.001

Figure 32 provides snapshots over the simulation to illustrate how the vorticity and magnitude of velocity evolve. Both Figure 32a,b illustrate the effect of periodic boundary conditions along each edge of the domain, e.g., vortices moving through the top or right side of the domain continue through the bottom or left side of the domain, respectively. Furthermore, in the snapshots provided, the overall vorticity and magnitude of velocity appears to dissipate within the domain. This is because there are no source terms (or explicit boundary conditions) providing additional flow (energy) into the system to induce more flow. Although the original velocity vector field that defined the initial vorticity of the system came from a simulation of pulsing hearts, without the heart moving and thus pushing on the fluid, the fluid will eventually stop moving and reach zero velocity due to the fluid's viscous nature. Therefore initializing the vorticity out of a time-point from another simulation's velocity field, is akin to studying the evolution of the fluid's motion from a singular impulse in the flow.



(a)



(b)

Figure 32. The evolution of (**a**) the fluid vorticity and (**b**) magnitude of velocity during the course of the simulation.

Students may elect to try the following:

1. Recreate the above results with the parameters listed in Table 6 or for different ν.
2. Take a velocity field from one of the Projection method examples, e.g., from Section 4.1 or Section 4.3, and use it to initialize the vorticity for a simulation using this spectral (FFT) method.
3. Find how long it takes the simulation to reach a $1/10$ or $1/100$ of the original maximum magnitude of velocity.
4. Discover how that time varies with changes in viscosity, ν.

4.5. Flow Past One or More Cylinders (via Lattice Boltzmann)

Flow past objects is one of the most heavily studied problems in aerodynamics and thus is a standard example used in many fluid dynamics courses [68–70]. While many texts only consider the problem for inviscid flows and use potential flow methods to solve for exact solutions [69,70], here we include effects of viscosity to illustrate transitions to vortical flow, and upon doing so, a subset of the possible flow separation cases. Furthermore, flow past cylinders is also still an active area of contemporary research [71–77].

We performed simulations for both a single cylinder as well as multiple cylinders. The computational geometry for a single cylinder case is given in Figure 33. Figure 33a gives the boundary conditions considered for the problem for flow past a rigid cylinder contained within a rigid channel. The walls of the channel are rigid and so bounce-back boundary conditions are used. The ends of the channel use periodic boundary conditions, e.g., what goes out the right side, comes in through the left. Note that this is the first example discussed in which uses a mix of explicit boundary conditions and periodic boundary conditions. Moreover, the cylinder itself is modeled as a rigid structure and so bounce-back conditions are used on it. Figure 33b provides visual details how the inflow is modeled in the simulation. Every iteration, the horizontal velocity at the inflow is increased by an amount of ΔU_x to drive fluid flow towards the right.

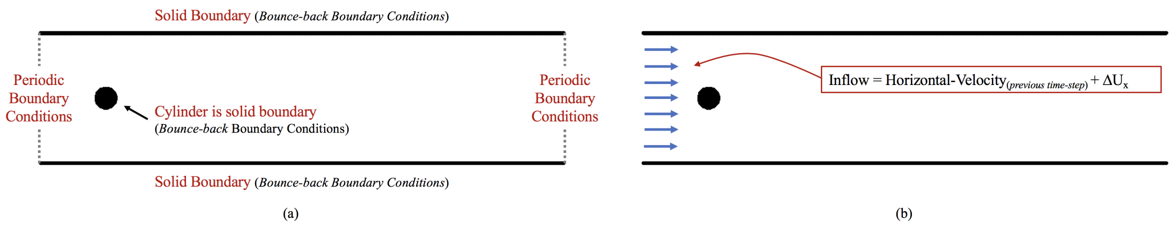

Solid Boundary (Bounce-back Boundary Conditions)
Periodic Boundary Conditions
Cylinder is solid boundary (Bounce-back Boundary Conditions)
Periodic Boundary Conditions
Solid Boundary (Bounce-back Boundary Conditions)
(a)
Inflow = Horizontal-Velocity$_{(previous\ time\text{-}step)}$ + ΔU_x
(b)

Figure 33. The geometrical setup for flow past cylinders using the Lattice Boltzmann method. (**a**) The boundary conditions specified on all sides of the domain and on the cylinder within the channel and (**b**) the inflow condition.

All simulation parameters (fluid, grid, and geometry) for both cases involving either a single cylinder or multiple rigid cylinders are given in Table 7. Note that the cases with multiple cylinders simulate flow around cylinders with larger radii, use a larger value for ΔU_x, and a different final number of steps and grid resolutions. Furthermore, due to these differences we are not comparing the case of one single cylinder to the case with multiple cylinders.

Table 7. Numerical parameters for flow past one or more cylinders using the Lattice Boltzmann Method.

Parameter	Variable	Units	Value
Domain Size	$[L_x, L_y]$	m	$[2, 0.5]$
Spatial Grid Resolution (1 Cylinder)	$[N_x, N_y]$		$[640, 160]$
Spatial Grid Resolution (Multiple Cylinders)	$[N_x, N_y]$		$[512, 128]$
Spatial Grid Size	$dx = dy$	m	$L_x/N_x = L_y/N_y$
'Density' Initialization	ρ		0.01
Relaxation Parameter	τ		0.53
Total Simulation Steps (1 Cylinder)	N		56000
Total Simulation Steps (Multiple Cylinders)	N		5500
Incremental inflow velocity increase (1 cylinder)	Δu_X		0.00125
Incremental inflow velocity increase (Multiple Cylinders)	ΔU_x		0.01
Radii (1 Cylinder)	r	$\%L_y$	$7.5\%\ L_y$
Radii (Multiple Cylinders)	r	$\%L_y$	$12.5\%\ L_y$

The simulation data obtained at step $n = 49,600$ is given in Figure 34. It presents colormaps (and corresponding contours) for vorticity, magnitude of velocity, horizontal velocity, vertical velocity, and the finite time Lyapunov exponent (FTLE), to which knowledge of the approximate Lagrangian

Coherent Structures (LCS) can be extracted [58,59,61,65,66]. We attribute regions with high FTLE to regions of high fluid mixing, as higher FTLE suggests that nearby fluid parcels separate at a much faster rate than those in lower FTLE regions. From the vorticity panel, significant flow separation is seen as vortices are being shed off the cylinder and flow pushes past it left-to-right. Oppositely spinning vortices are shed off cylinder one after another. Moreover, we observe that there is significantly more horizontal fluid motion than vertical within the channel. Since flow is being pushed left-to-right from the inflow condition, it would require more energy for fluid to move vertically, than to simply. . . go with the flow. Figure 35 provides snapshots of vorticity (left) and FTLE (right) during the simulation to highlight how smooth flow developed into vortical flow patterns once the horizontal velocity increased past a threshold. Such threshold appears to have occurred around simulation step $\sim 47,000$.

In particular, flow instabilities began to manifest by step $46,000$, as seen by both the vorticity and FTLE snapshots. The majority of fluid mixing occurred in large contours behind the cylinder up to this point. Once flow separation occurs, some mixing occurs in the wake in small patches, but no geometrically long regions of higher fluid mixing are present, as seen by the contours. Note that in the last two panels, for steps $49,200$ and $49,600$, large FTLE-valued regions do not correspond to locations of shed vortices, rather higher FTLE valued regions appear between oppositely spinning vortices.

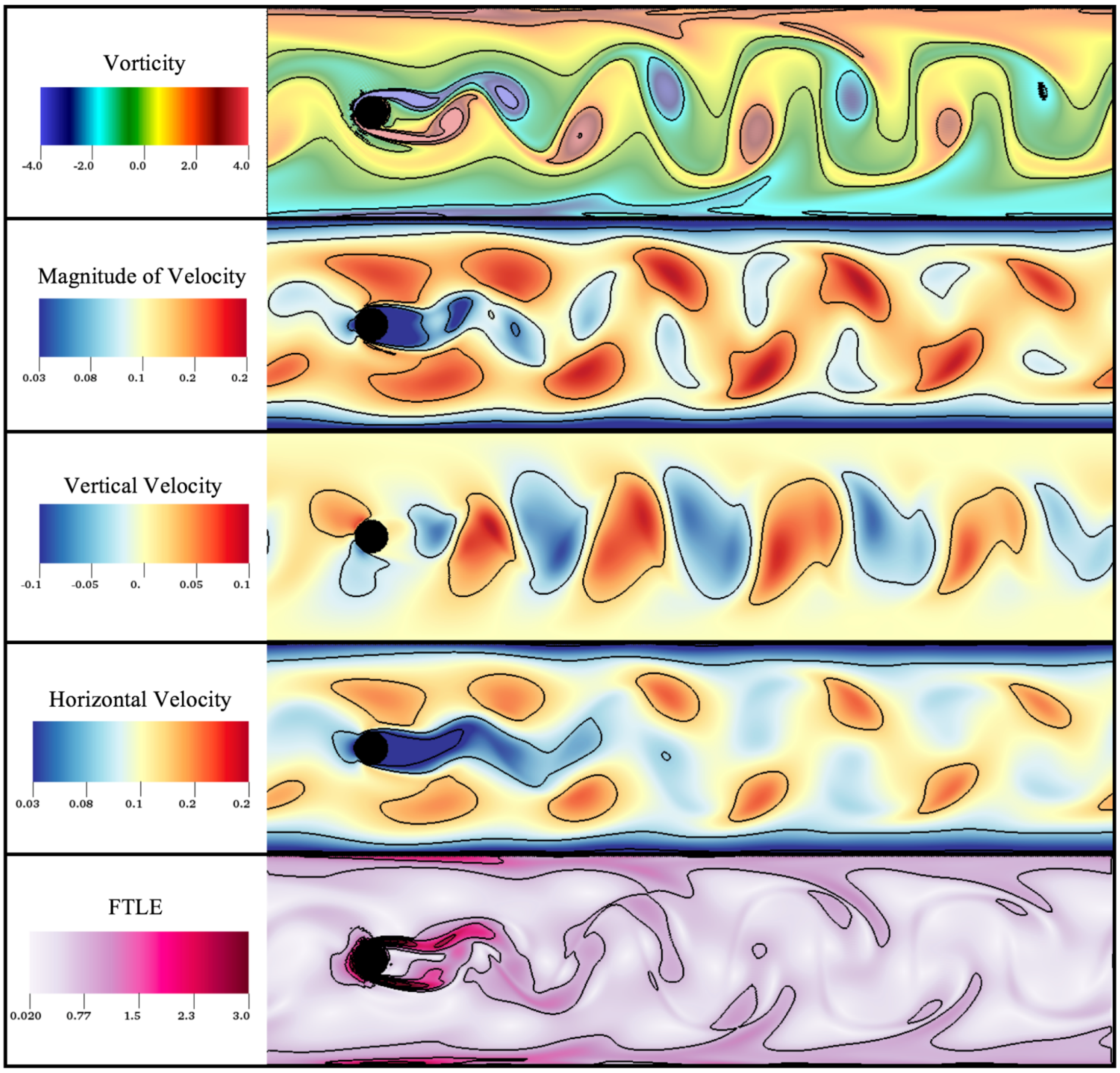


Figure 34. Illustrations of the flow field at the end of a simulation with one cylinder in the channel, showing colormaps of vorticity, magnitude of velocity, vertical and horizontal velocity, and finite-time Lyapunov exponent.

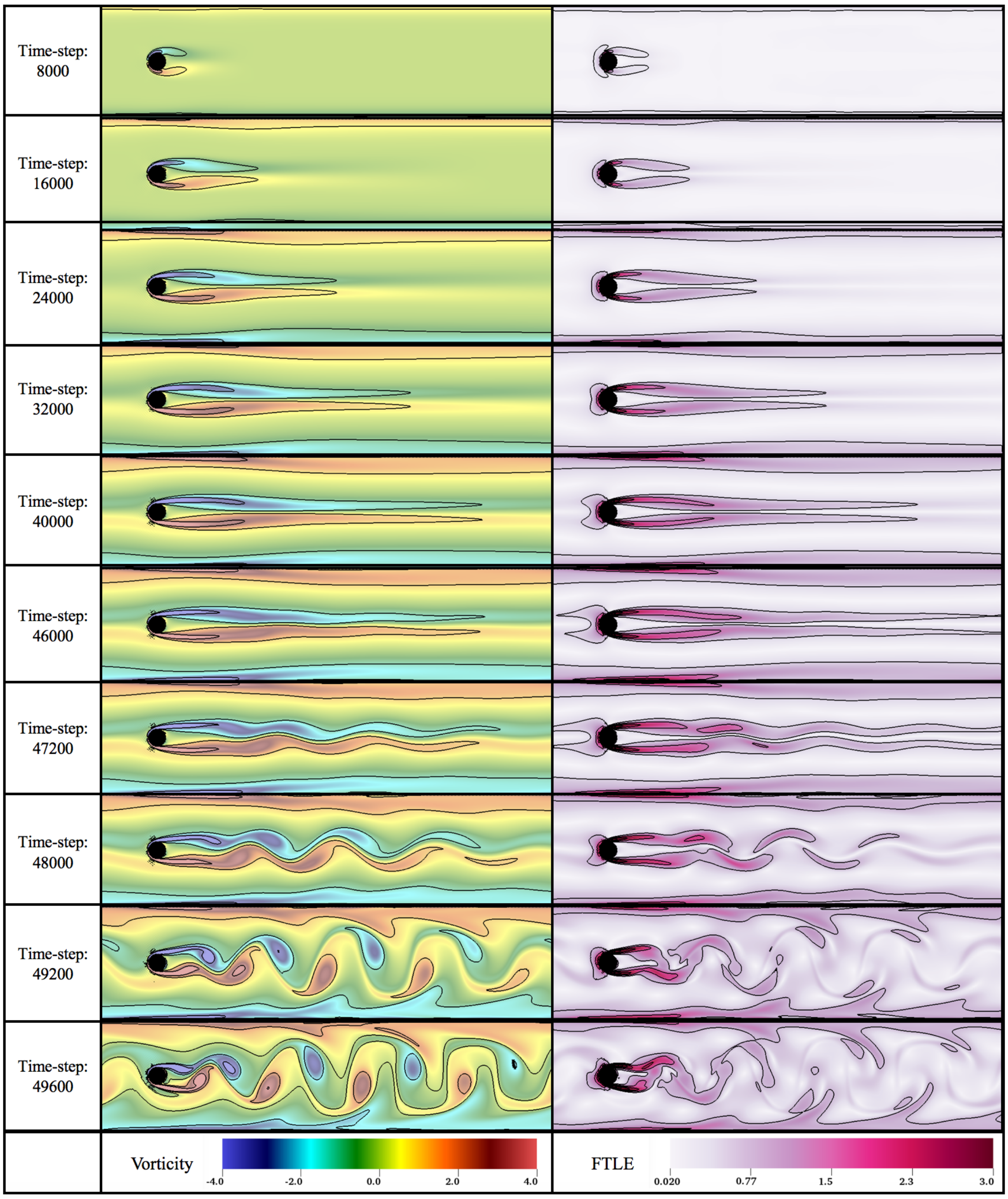
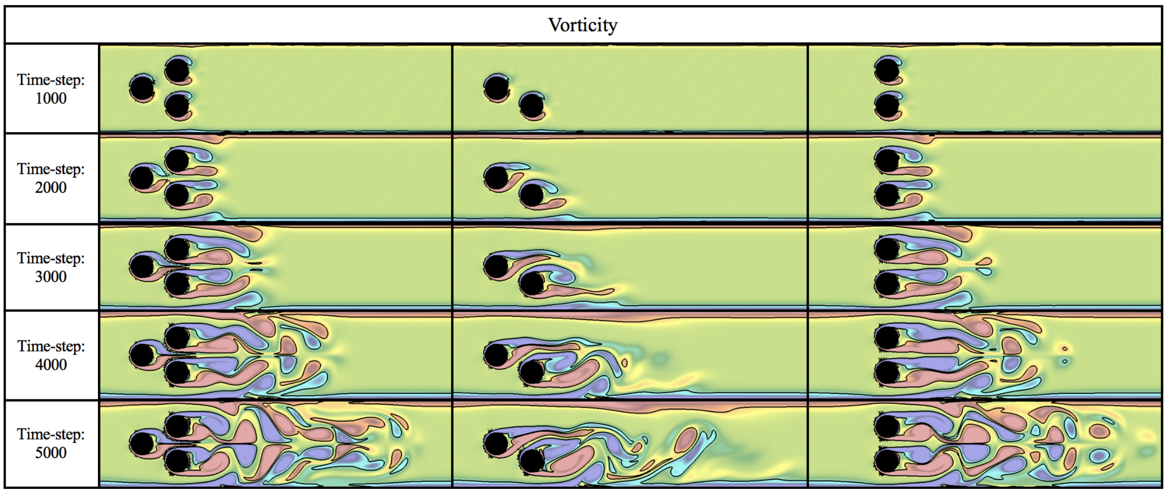


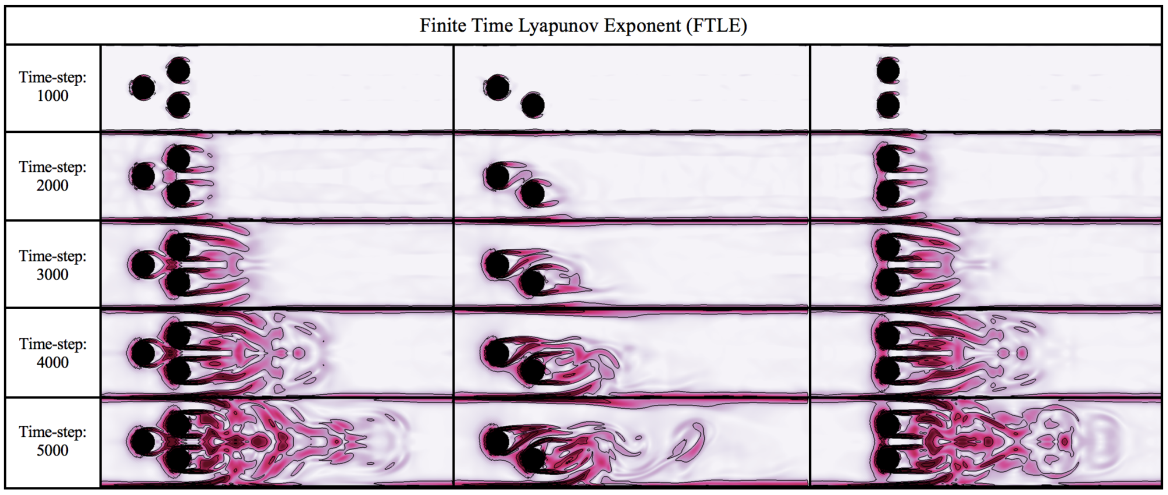
Figure 35. Illustrations depicting the flow's evolution to vortex shedding using vorticity (**left column**) and finite-time Lyapunov exponent (**right column**) with their contours, respectively.

Finally we performed simulations with multiple cylinders in a channel. Each cylinder in these simulations had a radii 66% larger than the single cylinder shown above. We performed three separate simulations with different configurations of the cylinders, each based off a triangle formation. Figure 36 illustrates how flow evolved for each geometric configuration, in particular for (a) vorticity and (b) FTLE.

Vortices are observed shedding off each cylinder, possibly at an enhanced rate due to flow interactions with the other cylinders (see suggested activity below). In the symmetric configurations, e.g., the triangle-formation and vertical-line formation, flow symmetry is preserved, while in the case with only one trailing cylinder, flow asymmetries arise. FTLE plots at the last step shown (5000) illustrate different geometrical configurations not only can lead to different patterns of fluid mixing, but also enhanced mixing, even among cases with only two cylinders. The vertically-aligned cylinder case appears to elicit more downstream mixing in the wake than the case of asymmetrically placed cylinders; however, this may also be an artifact of overall geometry of the system, e.g., the size of the cylinders and the proximity of cylinders to the channel walls.



(a)



(b)

Figure 36. Snapshots from simulations of multiple cylinders in different configurations showing (**a**) vorticity and (**b**) finite-time Lyapunov exponents with their contours, respectively. Note that the colormaps are the same as used in Figure 35.

Students may elect to try the following:

1. Recreate the above results with the parameters listed in Table 7
2. Using the radius for the larger geometry case, run a simulation with only one cylinder present in the middle and compare vorticity and FTLE evolution plots.
3. Vary τ (relaxation parameter relate to viscosity) to test the system's sensitivity to τ
4. Change the number, placement, or size of each cylinder in the domain.
5. Change the shape of the object in the channel, e.g., try an ellipse or airfoil-like shape

4.6. Flow Past a Porous Cylinder (via Lattice Boltzmann)

While flow around solid objects is a popular problem in aerodynamics, flow around porous objects has only recently begun to be investigated within in the past few decades, using either high fidelity numerical simulation or experiments [78–82]. Here, we present an example of flow past a porous cylinder using the LBM. The cylinder geometry, computational setup, and inflow/boundary conditions are identical to the single cylinder case of Section 4.5 (see Table 7). The only difference being that the cylinder geometry itself is porous.

The porosity, or void fraction (VF), of the cylinder was chosen to be $VF \in \{0\%, 10\%, 25\%, 50\%, 62.5\%\}$. To model the void fraction, each grid cell inside the cylinder was given a random number between $[0, 1]$ and then depending on the specific VF, if that grid cell's randomly assigned number was above the VF in decimal form, it was assigned as a solid boundary. For example, for $VF = 25\%$, each grid cell with a randomly assigned number greater than 0.25 was declared a solid boundary point. Figure 37 depicts the porous cylinders considered for each VF case studied below.

Although, the cylinder is no longer uniformly solid, in this example we will not focus our efforts to study flow through the cylinder itself. For example, in Figure 37's $VF = 10\%$ case, there are holes in the interior of the cylinder domain that are not connected to the fluid region outside of the cylinder. Those void regions will not influence the flow outside of the cylinder, as they are blocked from outside fluid penetrating them. However, if you wanted to study the fluid dynamics through a porous structure, you could design a specific void network structure within the cylinder or in another desired geometry. What we will emphasize here is that the porous regions connected to the outside of the cylinder cause asymmetries to develop in the overall flow pattern at an accelerated rate as compared to the non-porous (solid) case. Such asymmetries then lead to quicker vortex shedding.

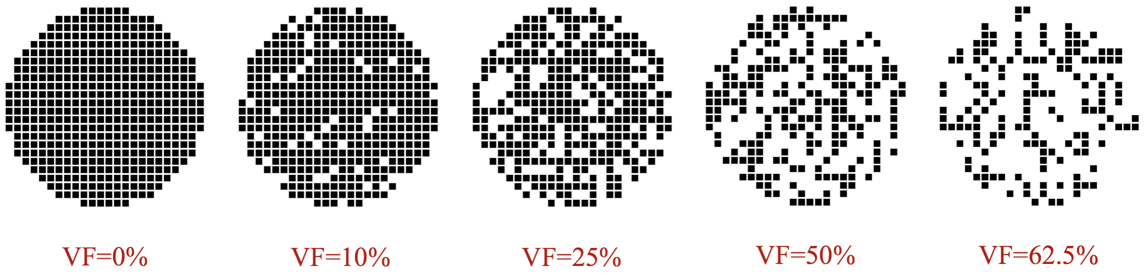


Figure 37. The porous cylinders considered in this section for different void fractions, $VF = \{0, 10, 25, 50, 62.5\}$. Note that the porous structures were initialized randomly within the cylinder, so there may not be open networks through the cylinder from one side to the other. The slight geometrical perturbations are sufficient enough to initiate quicker transitions to vortex shedding in each case of porous cylinder than the case of $VF = 0$.

Figure 38 shows the evolution of vortex shedding in cases of differing porosities (void fractions) of $\{0\%, 10\%, 25\%, 50\%, 62.5\%\}$. Every case shows that more porosity leads to faster development of vortex shedding. Furthermore, even the case with $VF = 10\%$ has pronounced vortices being shed before the $VF = 0\%$ case even shows any signs of significant flow asymmetry. The increased porosity accelerates flow asymmetries to manifest, leading to vortex shedding occurring earlier on. This example highlights how small perturbations in fluid dynamics problems (here small differences in geometric structure) can lead to the system having significantly different time-scales for flow structures to develop or for a bifurcation in the resulting dynamics to emerge.

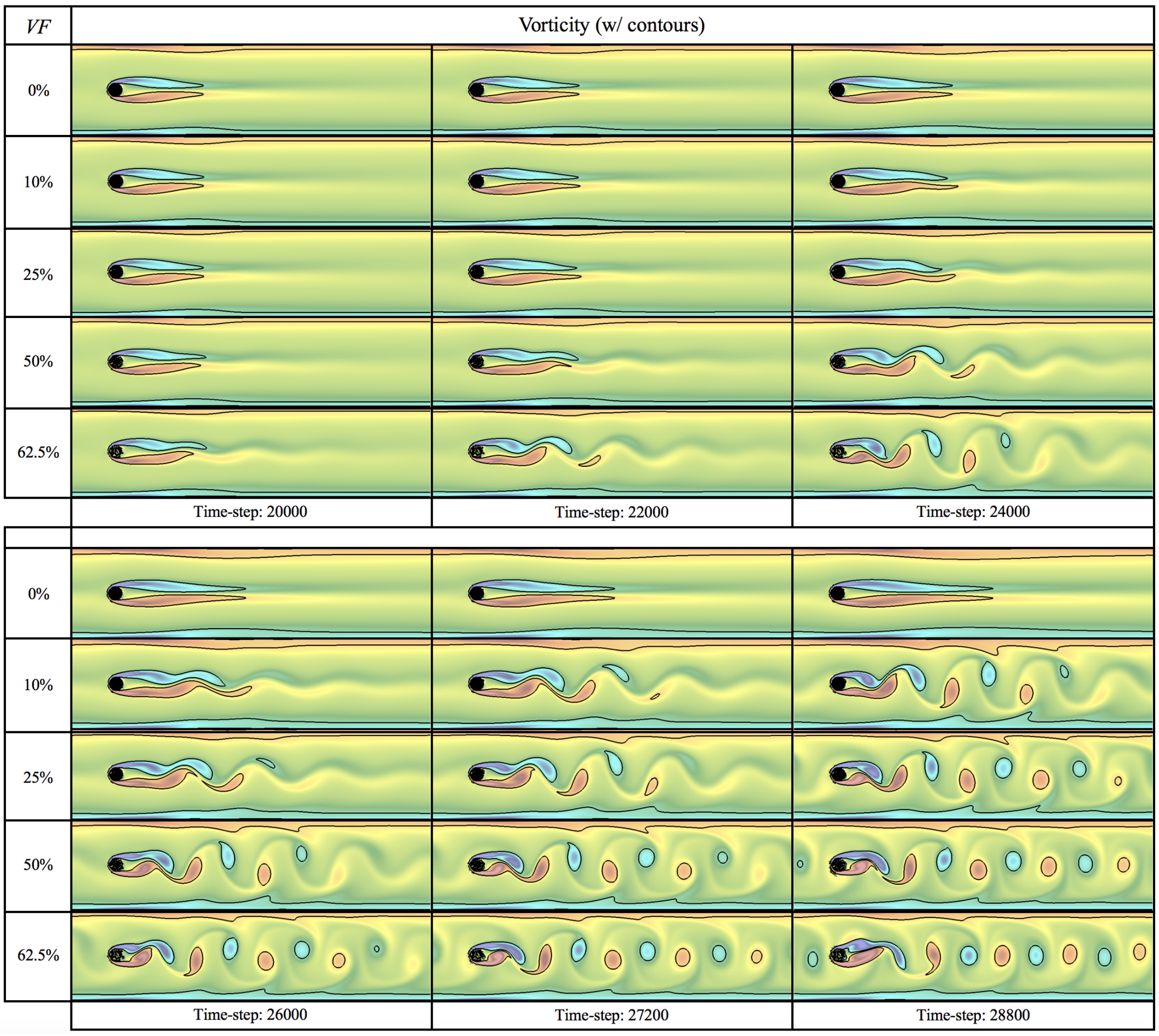


Figure 38. Snapshots illustrating vorticity (with its contours) among cases of differing void fractions, VF.

Students may elect to try the following:

1. Recreate the above results with the parameters listed in Table 7
2. Create the accompanying FTLE plots and discuss how fluid mixing changes due to the porous cylinder
3. Vary τ (relaxation parameter relate to viscosity) to test the system's sensitivity to τ
4. Increase the number of porous cylinder in the domain and vary their placement or size
5. Change the shape of the porous object in the channel, e.g., try an ellipse or airfoil-like shape
6. Create a specific porous network structure through the cylinder and test how flows directly through a porous cylinder may affect vortex shedding

5. Discussion

In this work we provide software that was developed for the specific purpose to make CFD accessible to undergraduate (and graduate) students in order to provide them an opportunity to perform traditional and contemporary CFD simulations as a valuable learning experience. To that end all the fluid solvers contained within were written in both MATLAB and Python, two languages that are commonly familiar to most science and engineering students [24–28,31]. Students also have the opportunity to modify existing examples or create their own examples within the software to test hypothesis or for stimulate further scientific curiosity. To that extent, we provided an overview of how the code is structured (Section 3) and offered interesting variations to each of the examples that we showcased in this paper (Section 4).

Not only can students run CFD simulations, they also have the chance the practice the art of scientific visualization, and use open-source software that is used by many researchers (VisIt [41] or Paraview [42]), to visualize the corresponding data. These software packages are both open-source and have easy-to-use and learn graphical user interfaces (GUIs) that streamline the visualization process. They can also be used to perform data analysis as well, which further reduces the computational learning curve for students to be able probe the data produced. We provided step-by-step guides on how to use VisIt for these tasks in Section 3.

By performing their own CFD experiments, fluid dynamics students have the opportunity to vary parameters (e.g., fluid scale (*Re*), boundary conditions, geometry, etc.) of a given system and observe the resulting dynamics. More specifically, they are able to decrypt how the dynamics may vary across parameter regimes, both qualitatively and quantitatively. This grants them the ability to challenge their own developing intuition of the resulting flow physics to further accelerate their learning. They also gain more computer experience and witness first-hand some of the advantages that complex computer simulations offer.

If students wish to go a step further and study the underlying mathematical structure of the code, we provide insightful comments everywhere possible within each fluid solver script. To complement this, we provided a detailed mathematical overviews of each fluid solver used here in Appendix B. While it is not our mission here to teach students how implement their own numerical fluid solvers, we believe this work contributes to providing a foundation for capturing the importance (and utility) of different fluid solver schemes. For each solver that was introduced for particular applications in Sections 2.1–2.3, we give a high-level overview of the method. If students wish to continue learning about numerical methods for solving the fluid equations or numerical methods for partial differential equations in greater depth, we suggest Barba and G. Forsyth's *CFD Python: the 12 steps to Navier-Stokes equations* [33], L.A. Barba and O. Mesnard's *Aero Python: classical aerodynamics of potential flow using Python* [34], both of which use Jupyter Notebooks [38], or Pawar and San's [39] modules for developing fluid solvers in the Julia programming language [40].

In this day and age, as the importance of computer literacy increases rapidly [83–86], integrating more hands-on computer-based activities into course structures will be of critical importance. However, students also seeing successful execution of code is paramount [85]. Here we provide gateway software for students to dive into the world of CFD, in familiar programming environments. While proprietary software offers immense advantages to running simulations and analyzing data, students may see a disconnect between the programming knowledge they've acquired and such polished software. We hope to provide them a unique opportunity to see fluid solvers written in familiar, non-intimidating environments that they may be able to tweak and modify comfortably. Our hope is to inspire further ownership of their learning and to stimulate more interest in (lucrative, transferable) computer programming skills.

Supplementary Materials: The following are available at http://www.mdpi.com/2311-5521/5/1/28/s1.

Funding: N.A.B. was funded and supported by the NSF OAC-1828163, the TCNJ Support of Scholarly Activity (SOSA) Grant, the TCNJ Department of Mathematics and Statistics, and the TCNJ School of Science.

Acknowledgments: The author would like to thank Laura Miller for introducing him to the joys of computational fluid dynamics. He would also like to thank Christina Battista, Robert Booth, Christina Hamlet, Alexander Hoover, Matthew Mizuhara, Arvind Santhanakrishnan, Emily Slesinger, and Lindsay Waldrop for comments and discussion. He also would like to sincerely thank the Reviewers, especially Reviewer 2, whose above and beyond effort in providing incredibly thorough, insightful, and constructive feedback led to the manuscript becoming significantly stronger.

Conflicts of Interest: The author declares no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CFD	Computational Fluid Dynamics
Re	Reynolds Number
BCs	Boundary Conditions
LBM	Lattice Boltzmann Method
VF	Void Fraction (Porosity)
GPU	Graphics Processing Unit
CPU	Central Processing Unit

Appendix A. Instructor Resources

Teaching Resources:

Associated supplemental files contain movies and codes pertaining to all the simulations detailed in this paper. It encompasses the following:

1. **suiteCFD_Supplement.pptx/.pdf**: presentations which may be used in class; slides that tell the story of the paper. Note that the *.pptx* file has embedded movies in *.mp*4 format.
2. **Images**: directory containing images (simulation snapshots, data) pertaining to each simulation shown in the manuscript.
3. **Movies**: directory containing movies (*.mp*4 format) pertaining to each simulation shown in the manuscript.
4. **suite-CFD-Software-02-06-2019.zip**: zip-file containing all fluid solver codes used in the manuscript (as of 6 February 2020).
5. Note that all codes used in the manuscript can also be found at: https://github.com/nickabattista/Holy_Grail.
6. Visualization software used: VisIt (https://visit.llnl.gov/) (v. 2.12.3).

Appendix B. Select CFD Algorithms

In this appendix we will discuss various numerical algorithms for studying fluid dynamics, namely a projection method [2–5], a spectral methods solver based on the Fast Fourier Transform (FFT) [7–9], and the lattice Boltzmann method [11,12]. Codes are available to test these methods at https://github.com/nickabattista/Holy$_$Grail/. Their corresponding simulation data is saved in the *.vtk* format, as to allow for visualization and analysis using open-source programs, such as VisIt [41] or ParaView [42].

Appendix B.1. Projection Methods

The projection was first introduced by Chorin in 1967 [2] and independently a year later by Temam [4] to solve the incompressible, Navier-Stokes equations [2]. The key feature of this method is that it uses operator splitting and Helmholtz-Hodge decomposition to decouple the velocity and the pressure fields, making it possible to explicitly solve the incompressible, Navier-Stokes equations in only a few steps.

We will begin by introducing some notation. We denote $\mathbf{u}_{ij}^n = \mathbf{u}^n(x_i, y_j)$ to be the velocity field field at time-step n and spatial location (x_i, y_j), where $\{x_i\}_{i=0}^{N_x}$ and $\{y_j\}_{j=0}^{N_y}$ are the x and y values of the discretized rectangular (Eulerian) computational grid. Furthermore, $\mathbf{u}^n$ represents the velocity field at time-step n in its entirety, e.g., not at one specific spatial location.

In the first step, an auxiliary (intermediate) velocity field is computed by ignoring any dependencies on the pressure. This is essentially an operator split. This velocity field found will not be divergence-free, and hence the necessary incompressible condition will not be satisfied (Equation (2)). In discretization terms, this step takes the form of

$$\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} = -\left(\mathbf{u}^n \cdot \nabla\right)\mathbf{u}^n + \nu\Delta\mathbf{u}^n, \tag{A1}$$

where $\mathbf{u}^*$ is an intermediate (auxiliary) velocity field, which is not divergence-free. The second step is known as the projection step, where the pressure gets reintroduced to give a final velocity field, $\mathbf{u}^{n+1}$ that satisfies the incompressibility condition (Equation (2)). This discretized step takes the following form,

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^*}{\Delta t} = -\frac{1}{\rho}\nabla p^{n+1}, \tag{A2}$$

where p^{n+1} is the pressure field at the next time-step. Because it requires an updated pressure term, we must first find such a pressure. To do this we recall Helmholtz-Hodge Decomposition [87,88], which says any vector field, that is twice continuously differentiable on a bounded domain, say $\mathbf{v}$, can be decomposed into a solenoidal part (divergence-free) and an irrotational part (curl-free), i.e.,

$$\mathbf{v} = \mathbf{v}_{sol} + \mathbf{v}_{irr} = \mathbf{v}_{sol} + \nabla\phi, \tag{A3}$$

where $\mathbf{v}_{sol}$ is the solenoidal part and $\mathbf{v}_{irr}$ is the irrotational part. We note that an irrotational vector field can be written as the gradient of a scalar, e.g., $\mathbf{v}_{irr} = \nabla\phi$, where ϕ is some scalar function (sometimes ϕ is referred to as a *potential*).

Note that if we take the divergence of (A3), we obtain,

$$\nabla \cdot \mathbf{v} = \Delta\phi. \tag{A4}$$

It is then possible to find the divergence-free part of the vector field $\mathbf{v}$ by solving the above Poisson problem in (A4). This motivates the form of the second step for a projection method given in (A2). However, to find the pressure, we take a divergence of (A2) and note that we require that $\mathbf{u}^{n+1}$ be divergence-free, i.e.,

$$\nabla \cdot \mathbf{u}^{n+1} = 0. \tag{A5}$$

Taking the divergence of (A2) and requiring the condition in (A5), we obtain the following Poisson problem for the pressure, p^{n+1}, in terms of the intermediate velocity field $\mathbf{u}^*$,

$$\frac{\Delta t}{\rho}\Delta p^{n+1} = \nabla \cdot \mathbf{u}^*. \tag{A6}$$

Note that this equation can be solved explicitly. Hence once p^{n+1} is found, we can then solve for $\mathbf{u}^{n+1}$ using (A2), e.g.,

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \frac{\Delta t}{\rho}\nabla p^{n+1}. \tag{A7}$$

Appendix B.2. Spectral Methods via Fast Fourier Transform (FFT)

For the spectral (FFT) method fluid solver we choose to work in the vorticity formulation of the viscous, incompressible Navier-Stokes equations. Here we will begin by deriving such equations from the previously written viscous, incompressible Navier-Stokes equations, i.e., Equations (1) and (2),

$$\rho\left[\frac{\partial\mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right] = -\nabla p + \mu\Delta\mathbf{u}$$

and

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

We will use a lot of identities from vector calculus. First, recall the following identity for any vector $\mathbf{F}$,

$$(\mathbf{F}\cdot\nabla)\mathbf{F} = (\nabla\times\mathbf{F})\times\mathbf{F} + \frac{1}{2}\nabla(\mathbf{F}\cdot\mathbf{F}).$$

Substituting the above identity into the conservation of momentum equation (Equation (1) and dividing by ρ gives us,

$$\frac{\partial\mathbf{u}}{\partial t} + (\nabla\times\mathbf{u})\times\mathbf{u} + \frac{1}{2}\nabla(\mathbf{u}\cdot\mathbf{u}) = -\frac{1}{\rho}\nabla p + \nu\Delta\mathbf{u}, \tag{A8}$$

where $\nu = \frac{\mu}{\rho}$ is the kinematic viscosity. Note that $\mathbf{u}\cdot\mathbf{u}$ is a scalar quantity, and thus we define it to be $U^2 = \mathbf{u}\cdot\mathbf{u}$. Moreover, we also get to define a new quantity called the vorticity,

$$\boldsymbol{\omega} = \nabla\times\mathbf{u}. \tag{A9}$$

It is tempting to think of $\boldsymbol{\omega}$ as describing the global rotation of the fluid, but this is misleading. Although many flows can be characterized by local regions of intense rotation, such as smoke rings, whirlpools, tornadoes, or even the red spot on Jupiter, some flows have no global rotation, but do have vorticity. Vorticity describes the *local* spinning of a fluid near a fixed point in space, as seen by an observer in the Eulerian framework.

Substituting the definitions of vorticity and U^2 into Equation (A8), we obtain

$$\frac{\partial\mathbf{u}}{\partial t} + \boldsymbol{\omega}\times\mathbf{u} + \frac{1}{2}\nabla(U^2) = -\frac{1}{\rho}\nabla p + \nu\Delta\mathbf{u}.$$

Now taking the curl of the above equation we get an equation for the evolution of the vorticity, yields

$$\frac{\partial\boldsymbol{\omega}}{\partial t} + \nabla\times(\boldsymbol{\omega}\times\mathbf{u}) = \nu\Delta\boldsymbol{\omega}, \tag{A10}$$

since $\nabla\times(\Delta\mathbf{u}) = \Delta(\nabla\times\mathbf{u}) = \Delta\boldsymbol{\omega}$. Furthermore we note that the pressure terms drop out as the resulting force from pressure only acts perpendicular to the surface of a fluid blob and not parallel to it, i.e., $\nabla\times(\nabla\Phi) = 0$ for any scalar field Φ.

The viscous, incompressible Navier-Stokes equations can thus be written as follows,

$$\frac{\partial\boldsymbol{\omega}}{\partial t} + \nabla\times(\boldsymbol{\omega}\times\mathbf{u}) = \nu\Delta\boldsymbol{\omega} \tag{A11}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{A12}$$

Shortly we will introduce a vector potential to strive towards introducing a streamfunction, ψ, into our formulation, but first we will use a vector calculus identity to re-write Equation (A11) into a more traditional looking advection-diffusion equation. Using the following identity from vector calculus,

$$\nabla\times(\mathbf{A}\times\mathbf{B}) = (\nabla\cdot\mathbf{B} + \mathbf{B}\cdot\nabla)\mathbf{A} - (\nabla\cdot\mathbf{A} + \mathbf{A}\cdot\nabla)\mathbf{B}, \tag{A13}$$

we can mathematically massage Equation (A11) into the following form,

$$\frac{\partial\boldsymbol{\omega}}{\partial t} + (\mathbf{u}\cdot\nabla)\boldsymbol{\omega} = (\boldsymbol{\omega}\cdot\nabla)\mathbf{u} + \nu\Delta\boldsymbol{\omega}. \tag{A14}$$

Note that the evolution equation for vorticity, Equation (A14), now looks like an advection-diffusion equation, but with an additional extra term, $(\boldsymbol{\omega}\cdot\nabla)\mathbf{u}$. For $2D$ flows, recall that $\mathbf{u} = (u, v, 0)$ and hence $\boldsymbol{\omega} = (0, 0, \omega)$. Moreover, in 2$D$ flows, all partial derivatives with respect to z are zero; hence the $(\boldsymbol{\omega} \cdot \nabla)\mathbf{u}$ term becomes zero, e.g.,

$$(\boldsymbol{\omega} \cdot \nabla)\mathbf{u} = \left(0\frac{\partial}{\partial x} + 0\frac{\partial}{\partial y} + \omega\frac{\partial}{\partial z} \right) \mathbf{u} = \omega\frac{\partial \mathbf{u}}{\partial z} = 0.$$

Thus, in 2D, we have the following form of the momentum equation in terms of vorticity

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + (\mathbf{u} \cdot \nabla)\boldsymbol{\omega} = \nu \Delta \boldsymbol{\omega}. \tag{A15}$$

Note that the form of Equation (A15) suggests that if $\nu \equiv 0$ and if $\boldsymbol{\omega} = 0$ everywhere at any moment in time, then $\boldsymbol{\omega} = 0$ for all future time. Moreover, since $\omega = \nabla \times \mathbf{u} = 0$, we have irrotational flow. Thus, we would be studying an incompressible *potential flow* problem [70].

Next we introduce the streamfunction, ψ, as part of the vector potential for $\mathbf{u}$,

$$\mathbf{u} = \nabla \times \psi \hat{k}. \tag{A16}$$

Note that if the streamfunction, $\psi = \psi(x, y)$, is known, it is possible to extract the components of the 2D fluid velocity field, $\mathbf{u} = (u, v)$, from it e.g.,

$$u = \frac{\partial \psi}{\partial y} \quad \text{and} \quad v = -\frac{\partial \psi}{\partial x}. \tag{A17}$$

Furthermore, taking the curl of (A16), we are able to get a Poisson problem for ψ in terms of $\boldsymbol{\omega}$,

$$\Delta \psi = -\omega, \tag{A18}$$

where we have used the following vector calculus identity,

$$\nabla \times \nabla \times \mathbf{A} = \nabla(\nabla \cdot \mathbf{A}) - \nabla^2 \mathbf{A}, \tag{A19}$$

and the fact that $\frac{\partial \psi}{\partial z} = 0$.

At this point, the idea is that if we are able to solve for the streamfunction, ψ, from the vorticity, ω, we can then get the fluid velocity ,$\mathbf{u}$, and it will automatically satisfy the incompressibility condition by definition of the vector potential, i.e., Equation (A16). In essence this is the algorithm; however, within this algorithm, we will work as much as possible in the Fourier frequency space, granted to us by taking the Fast Fourier Transform (FFT). Before diving into the 4 main steps of this scheme, we will introduce some notation involving Discrete Fourier Transforms (DFT) and hence FFT. Note that the FFT yields the same results as the DFT but does so in a more computationally efficient, i.e., *fast*, manner.

- Taking the Discrete Fourier Transform (DFT) of a set of complex numbers produces another set of complex numbers. The notation $\mathscr{F}\{\mathbf{z}\}$ denotes taking the Discrete Fourier Transform of a set of N-values, $\{z_n\}_{n=0}^{N-1}$, e.g., for $k = 0, 1, \ldots, N-1$, we define

$$\hat{z}_k = \mathscr{F}\{\mathbf{z}\} = \sum_{n=0}^{N-1} z_n \, e^{-2\pi i \frac{kn}{N}}.$$

- Variables with a hat, such as $\hat{z}_k$, denote variables that have been transformed into frequency space via the Discrete Fourier Transform. Moreover, the indices k are known as the DFT's wave-numbers.

- We can also take the Inverse Discrete Fourier Transform (IDFT) to return our quantities of interest from frequency space to real space. We denote the IDFT as

$$z_n = \mathscr{F}^{-1}\{\hat{\mathbf{z}}\} = \frac{1}{N}\sum_{k=0}^{N-1} \hat{z}_k \, e^{-2\pi i \frac{kn}{N}},$$

  for all $n = 0, 1, 2, \ldots, N-1$.
- Next we will define the discretized quantities in the algorithm. A quantity such as f_{ij}^n denotes that quantity's value at the nth time-step at spatial location (x_i, y_j) within the rectangular computational grid. Hence we have a set of numbers $\{f_{ij}^n\}_{i=0,j=0}^{N_x-1,N_y-1}$ (or matrix that changes as $n \to n+1$), and can transform it in analogous manner using a DFT.
- We define K_X and K_Y to be matrices of the DFT's wave-numbers, where $K_X, K_Y \in \mathbb{R}^{N_x \times N_y}$. They are defined as:

$$K_X = \begin{bmatrix} 0 & 1 & 2 & \cdots & N_y - 1 \\ 0 & 1 & 2 & \cdots & N_y - 1 \\ \vdots & & \vdots & & \vdots \\ 0 & 1 & 2 & \cdots & N_y - 1 \end{bmatrix} \text{ and } K_Y = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 1 \\ 2 & 2 & 2 & \cdots & 2 \\ \vdots & & \vdots & & \vdots \\ N_x - 1 & N_x - 1 & N_x - 1 & \cdots & N_x - 1 \end{bmatrix}.$$

  Note that each row of the matrix K_X is a vector that we denote $\mathbf{k}_X$ with components $\mathbf{k}_X = (0, 1, 2, \ldots, N_y - 1)^T$. Similarly each column of the matrix K_Y is a vector that we denote $\mathbf{k}_Y = (0, 1, 2, \ldots, N_x - 1)$.
- In order to benefit from the FFT we will assume our spatial grid has a resolution of $N_x \times N_y$, where both N_x and N_y are powers of 2 [89].

We will now dive into the the 4 main steps in this spectral (FFT) method's algorithm. They are as follows:

1. *Update the streamfunction to the current time-step, n:*

   From the previous time-step's vorticity, ω^n, we can solve the Poisson problem (Equation (A18)) for the streamfunction at the current time-step, ψ^n, i.e.,

$$\hat{\psi}_{ij}^n = \frac{\hat{\omega}_{ij}^n}{k_{X_i}^2 + k_{Y_j}^2}, \tag{A20}$$

   where k_{X_i} and k_{Y_j} are the Fourier wave-numbers, e.g., the ith and jth components of the vectors $\mathbf{k}_X$ and $\mathbf{k}_Y$, respectively.

2. *Obtain the components of velocity and the gradient of vorticity:*

   With the newly updated $\hat{\psi}^n$ as well as $\hat{\omega}^n$ from the previous time-step, we are able to compute the components of the velocity field at the current time-step, $\mathbf{u}^n = (u^n, v^n)$. To do this, we take derivatives of the streamfunction and vorticity in frequency space. This then gives us the components of velocity (see Equation (A17)) and the gradient of vorticity, in frequency space respectively. We can then use the IDFT to transform these quantities back into in real space, e.g.,

$$u^n = \mathscr{F}^{-1}\left\{2\pi i \, K_Y \circ \hat{\psi}^n\right\} \tag{A21}$$
$$v^n = \mathscr{F}^{-1}\left\{-2\pi i \, K_X \circ \hat{\psi}^n\right\} \tag{A22}$$
$$\omega_x^n = \mathscr{F}^{-1}\left\{2\pi i \, K_X \circ \hat{\omega}^n\right\} \tag{A23}$$
$$\omega_y^n = \mathscr{F}^{-1}\left\{2\pi i \, K_Y \circ \hat{\omega}^n\right\} \tag{A24}$$

where the operation $A \circ B$ between two matrices of equal size is called the Hadamard product. The Hadamard product is element-wise multiplication, e.g., if A and B are matrices of the same size, for all components i, j, $(A \circ B)_{ij} = A_{ij}B_{ij}$.

3. *Compute the advection term in frequency space from Equation (A15):.*

   Once you have the velocity field (u^n, v^n) and partial derivatives of vorticity $\omega_x^n = \frac{\partial \omega^n}{\partial x}$ and $\omega_y = \frac{\partial \omega^n}{\partial y}$ at the current time-step, it is now possible to compute the advection term from Equation (A15), i.e., $\mathbf{u} \cdot \nabla \boldsymbol{\omega}$. We define $F^n_{\text{adv}_{ij}}$ to be the above advection term, and hence get that

$$F^n_{\text{adv}_{ij}} = u^n_{ij} \cdot \omega^n_{x_{ij}} + v^n_{ij} \cdot \omega^n_{y_{ij}}. \tag{A25}$$

   Furthermore, we can apply the DFT to Equation (A25) to transform it into frequency space, e.g.,

$$\hat{F}^n_{\text{adv}_{ij}} = \mathscr{F}\left\{ F^n_{\text{adv}_{ij}} \right\}. \tag{A26}$$

   This will allows us to update vorticity to the $(n+1)^{st}$ time-step using Equation (A15) in frequency space.

4. *Update the vorticity to next time-step:*

   Finally we use the Crank-Nicholson scheme to update the vorticity to the next time-step, $\hat{\omega}^{n+1}$,

$$\hat{\omega}^{n+1}_{ij} = \frac{\left[1 + \frac{\nu \Delta t}{2}\left(K^2_{X_{ij}} + K^2_{Y_{ij}}\right)\right]\hat{\omega}^n_{ij} - \Delta t\, \hat{F}^n_{\text{adv}_{ij}}}{1 - \frac{\nu \Delta t}{2}\left(K^2_{X_{ij}} + K^2_{Y_{ij}}\right)} \tag{A27}$$

   Note that this method is semi-implicit; we explicitly discretize the advection term, while we implicitly discretize the diffusion term. The Crank-Nicholson scheme is second order accurate in time and space [51], and is unconditionally stable for an array of parabolic problems of the type $w_t = aw_{xx}$ [52].

   Now that the vorticity has been updated, $\omega^n \rightarrow \omega^{n+1}$, you can repeat this process again.

Appendix B.3. Lattice Boltzmann Methods

As mentioned in Section 2.3, the Lattice Boltzmann method (LBM) does not explicitly (or implicitly) solve the viscous, incompressible Navier-Stokes equations, rather it uses discrete Boltzmann equations to model the fluid dynamics. In a nutshell tracks fictitious particles of fluid flow, thinking of the problem more as a transport equation, e.g.,

$$\frac{\partial f}{\partial t} + \mathbf{u} \cdot \nabla f = \Omega, \tag{A28}$$

where $f(\mathbf{x}, t)$ is the particle distribution function, i.e., a probability density, $\mathbf{u}$ is the fluid particle's velocity, and Ω is what is called the collision operator. However, rather than have these particles moving in a Lagrangian framework, the Lattice Boltzmann method simplifies this assumption and restricts the particle movements to nodes of a lattice. While we will only discuss a two dimensional implementation of the LBM, three dimensional implementations follow analogously.

From the assumption restricting the fluid particles to reside on a lattice, there are only 9 possible directions that a particle could potentially stream, or pass information, along to. These directions are either horizontal (left/right) or vertical (up/down) or forward or backward along both diagonal directions, as well as, staying at rest on its current node. These directions are illustrated in Figure A1, and these streaming velocities, $\{\mathbf{e}_i\}$, are called the *microscopic velocities*. The directions illustrated in Figure A1 is commonly called the *D*2*Q*9 Lattice Boltzmann Model.

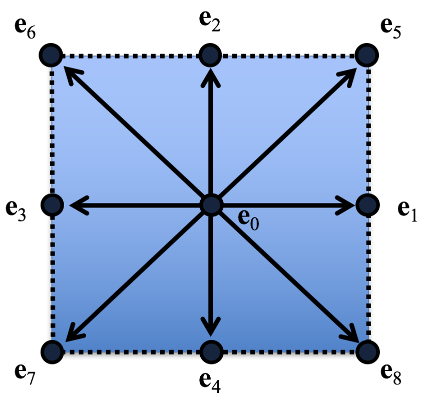


Figure A1. Figure illustrating the possible streaming directions, $\{\mathbf{e}_i\}$ for the D2Q9 Lattice Boltzmann model.

Every point on the lattice has is a probability function, $f(\mathbf{x}, t)$, associated with it. Accounting for the possibility of moving in only 9 directions, we rewrite the probability function as its discretized counterpart, $f_i(\mathbf{x}, t)$, where f_i now gives the probability of streaming in a particular direction $\mathbf{e}_i$. Using this discretization, we can define the macroscopic fluid density to be the sum of all possible f_i, e.g.,

$$\rho(\mathbf{x}, t) = \sum_{i=0}^{8} f_i(\mathbf{x}, t). \tag{A29}$$

Similarly, we can define the *macroscopic velocity* of the fluid as an average of the microscopic velocities in each direction weighted by their associated particle distribution functions f_i using (A29),

$$\mathbf{u}(\mathbf{x}, t) = \frac{1}{\rho} \sum_{i=0}^{8} c f_i(\mathbf{x}, t)\mathbf{e}_i, \tag{A30}$$

where $c = \frac{\Delta x}{\Delta t}$ and is referred to as the lattice speed. The key elements that are left to discuss are exactly what it means to *stream* the particle distributions, f_i, as well as what it meant by the *collision*, Ω. However, they both are encompassed within the steps in the LBM algorithm, so we will explicitly define these procedures while also describing the algorithm. The steps are detailed below:

1. The first step is to stream the particle densities to propagate in each direction. Explicitly you calculate the following intermediate particle density, f_i^*,

$$f_i^*(\mathbf{x} + c\mathbf{e}_i\Delta t, t + \Delta t) = f_i^n(\mathbf{x}, t), \tag{A31}$$

where n is the time-step and where for each direction i, you would in practice compute

$$\begin{aligned}
&f_1^*(x_i, y_j) = f_1^n(x_{i-1}, y_j), && f_2^*(x_i, y_j) = f_2^n(x_i, y_{j-1}), && f_3^*(x_i, y_j) = f_3^n(x_{i+1}, y_j),\\
&f_4^*(x_i, y_j) = f_4^n(x_i, y_{j+1}), && f_5^*(x_i, y_j) = f_5^n(x_{i-1}, y_{j-1}), && f_6^*(x_i, y_j) = f_6^n(x_{i+1}, y_{j-1})\\
&f_7^*(x_i, y_j) = f_7^n(x_{i+1}, y_{j+1}), && f_8^*(x_i, y_j) = f_8^n(x_{i-1}, y_{j+1}), && f_9^*(x_i, y_j) = f_9^n(x_i, y_j)
\end{aligned} \tag{A32}$$

This idea of streaming is shown in Figure A2.

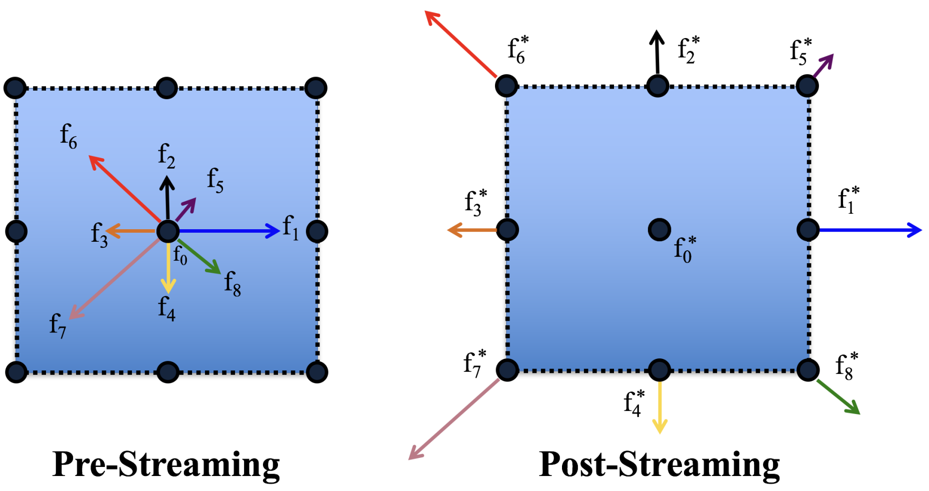


Figure A2. Figure illustrating the idea of streaming by showing color correlated particle probability functions, f_i, before the streaming process and post-streaming, f_i^*.

2. The second step involves finding what is referred to as the equilibrium distribution. This step is a part of the *collision* step, where you want to relax the particle density distributions towards a local equilibrium. The local equilibrium is denoted $f_i^{eq}(\mathbf{x},t)$. First we must compute macroscopic the properties (density and velocity) from the intermediate particle distributions f_i^* using (A29) and (A30).

   Once we have these quantities, we can now define the equilibrium distributions, f_i^{eq}. We note that there are many equilibrium distributions one could use in practice; however, each depends on your specific model and its assumptions. The Lattice Boltzmann method implemented here uses what is called the Bhatnagar-Gross-Krook (BGK) collision mode [90]. The BGK collision model is useful for simulating single phase flows [12] and is most often the classic model to use for solving the incompressible, viscous Navier-Stokes equations, although it can also be useful for simulating compressible flows at low Mach numbers [11]. See [11] for a good review of the BGK model. The BGK model's equilibrium distribution can be written as follows

$$f_i^{eq}(\mathbf{x},t) = w_i\rho + \rho s_i(\mathbf{u}(\mathbf{x},t)), \tag{A33}$$

   where w_i is a weight and $s_i(\mathbf{u}(\mathbf{x},t))$ is defined as

$$s_i(\mathbf{u}(\mathbf{x},t)) = w_i\left[3\frac{\mathbf{e}_i\cdot\mathbf{u}}{c} + \frac{9}{2}\frac{(\mathbf{e}_i\cdot\mathbf{u})^2}{c^2} - \frac{3}{2}\frac{\mathbf{u}\cdot\mathbf{u}}{c^2}\right]. \tag{A34}$$

   The corresponding weights, w_i are given as

$$w_i = \begin{cases} \frac{4}{9} & i=0 \\ \frac{1}{9} & i\in\{1,2,3,4\} \\ \frac{1}{36} & i\in\{5,6,7,8\} \end{cases}. \tag{A35}$$

3. Finally we compute the collision step associated with the BGK model as follows

$$f_i^{n+1} = f_i^* - \frac{f_i(\mathbf{x},t) - f_i^{eq}(\text{bf})}{\tau}, \tag{A36}$$

   where τ is the relaxation parameter and intuitively is related to the viscosity of the fluid, i.e.,

$$\nu = \frac{2\tau - 1}{6}\frac{\Delta x^2}{\Delta t}. \tag{A37}$$

Before we mention how to handle boundary conditions we will briefly discuss some of the advantages of the LBM. One of the biggest advantages of LBM is its implementation lends itself toward massive GPU or CPU parallelization. Due to parallelization it can be an incredibly fast way of solving fluid problems that are coupled with equations that model heat transfer or chemical processes [91]. Moreover, the algorithm also prides itself for the ability to compute flows through complex geometries and porous structures rather easily and efficiently [55]. From the structure of the streaming step, one can easily prescribe boundary conditions and regions in the grid where fluid is not allowed to flow easily. For our considerations here we only will introduce what are referred to as *bounce-back boundary conditions* [55].

The bounce-back boundary conditions are used to enforce no-slip conditions; however, as we will show, they are not only used on the edges of the domain, but can be implemented on the interior to create complex geometries. In a nutshell the incoming streaming directions of the distribution functions are simply reversed when they hit a boundary node. This idea is depicted in Figure A3. In practice, one can simply mask these boundary points on the domain using boolean logic.

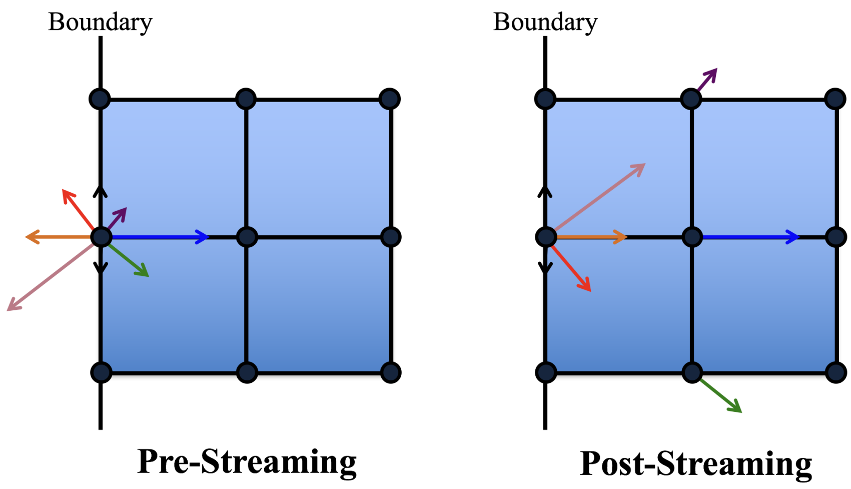


Figure A3. Illustration of bounce-back boundary conditions. During the pre-streaming step there are microscopic velocities set on the boundary and then they are reversed during the streaming step

Appendix C. Extra Visualizations of Data from Section 3: Spectral (FFT) Method's `bubble3` Example

These visualizations are to complement those already presented in the guided tutorials from Section 3. We will also give a bit of background for the simulation as well. This example used the spectral (FFT-based) fluid solver in the software and models multiple regions of vorticity 'overlapping' at the beginning. Note that first example given here can be run by going into in the *FFT_NS_Solver* script selecting the `'bubble3'` option. The vorticity is initialized as in Figure A4. Recall that counterclockwise (*CCW*) and clockwise (*CW*) correspond to regions of uniform vorticity, where vorticity initialized as a positive or negative constant for *CCW* and *CW*, respectively.

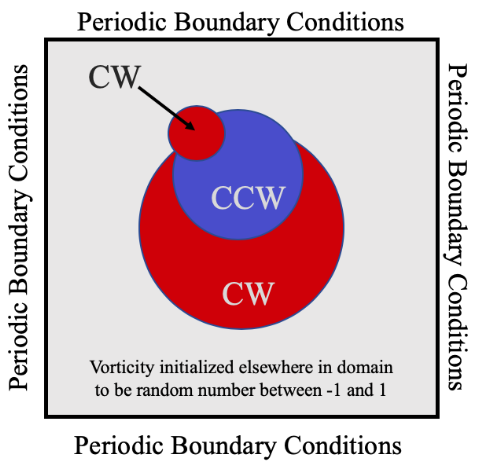

Periodic Boundary Conditions
Periodic Boundary Conditions
Periodic Boundary Conditions
Periodic Boundary Conditions
CW
CCW
CW
Vorticity initialized elsewhere in domain to be random number between -1 and 1

Figure A4. Illustrations of the boundary conditions and vorticity initialization for the cases of the `bubble3` example.

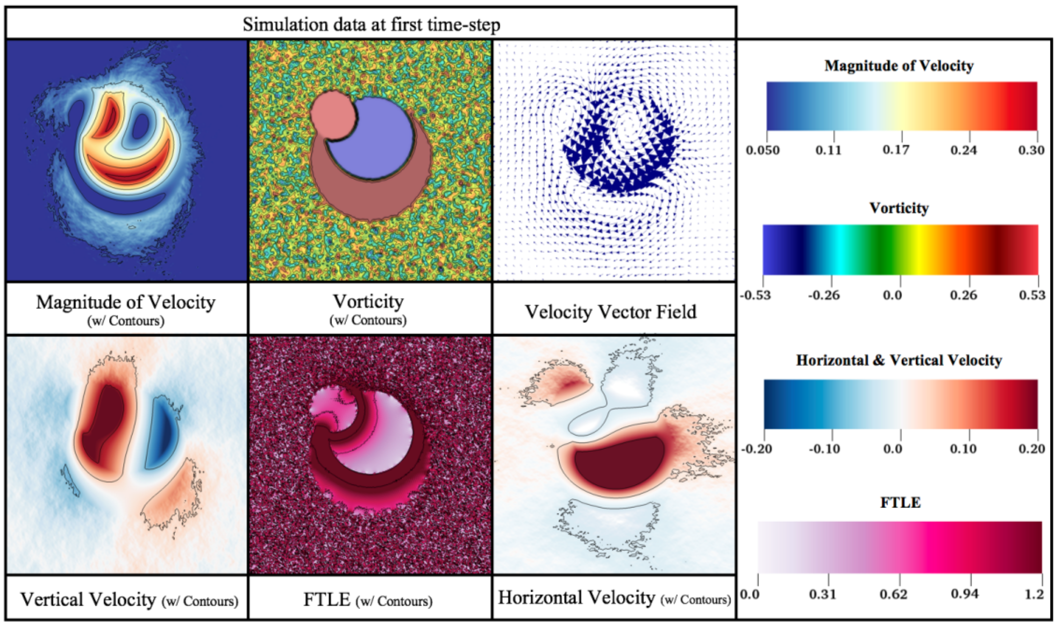
In this example, three circular regions of vorticity are placed, partially on top of one another. The largest region begins with a uniform value of $+0.4$, followed by the smaller regions with -0.5 and $+0.5$, respectively. The remainder of the computational domain is initialized with a random value of vorticity between $[-1, 1]$. We note that initializing a random values of vorticity will generally not satisfy the incompressibility condition (Equation (2)); however, here we do it only to illustrate that the solver is able to handle initial random noise. This simulation used the computational parameters found in Table A1.

Table A1. Numerical parameters for case with 'overlapping' vorticity regions

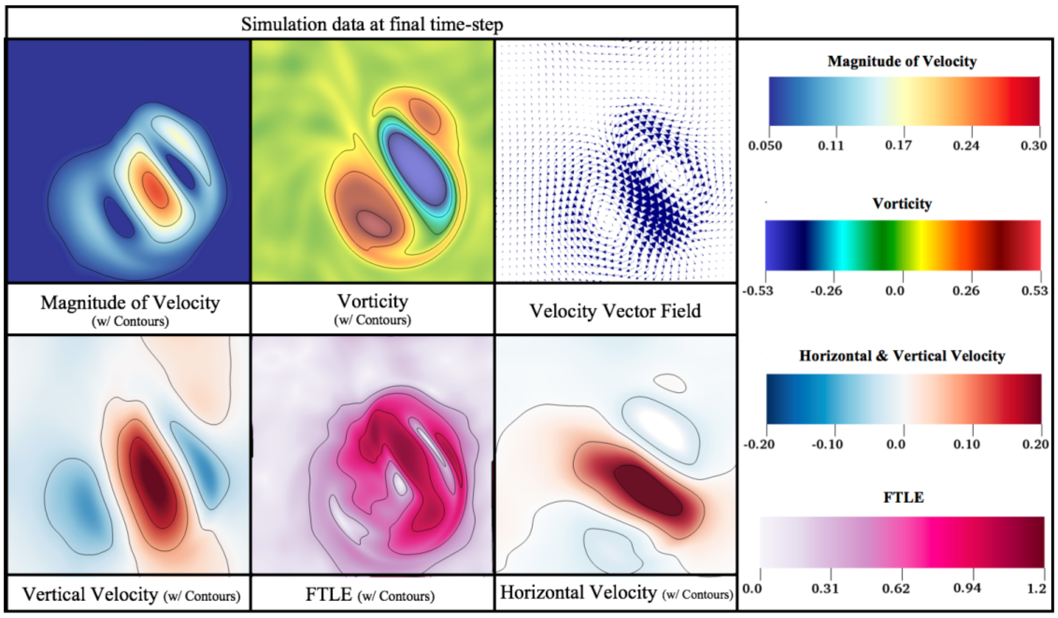
Parameter	Variable	Units	Value
Domain Size	$[L_x, L_y]$	m	$[1, 1]$
Spatial Grid Resolution	$[N_x, N_y]$		$[256, 256]$
Spatial Grid Size	$dx = dy$	m	$L_x/N_x = L_y/N_y$
Time Step Size	dt	s	10^{-2}
Total Simulation Time	T	s	30
Fluid Kinematic Viscosity	$\nu = \mu/\rho$	m^2/s	0.001

Figure A5 provides the simulation data at the beginning of the simulation (a) and at the last time-step (b). It presents colormaps (and corresponding contours) for vorticity, magnitude of velocity, horizontal velocity, vertical velocity, and the finite-time Lyapunov exponent (FTLE). It also gives a snapshot of the velocity vector field. The last snapshot of the simulation was from $t = 30.0$ s.

From Figure A5, it is evident that the random vorticity values at the start of the simulation eventually interact and dissipate in the flow, as observed in the vorticity panel. Initially there appears to be a lot of noise in the background vorticity (it was initialized to be random between $[-1,1]$) and by the end it appears virtually averaged out. Moreover, due to the background vorticity noise at the beginning, there are a lot of tiny patches of oppositely moving fluid. This leads to an initial background of high FTLE values, which suggests there is significant fluid mixing occurring. Similarly to vorticity, by the end, the background has significantly less mixing (e.g., smaller FTLE values) overall in areas away from the interacting vortical structures, which started off as overlapping vorticity regions. The area of high mixing in the FTLE panel at the last time-step is in a region where there is a lot of oppositely moving fluid, both horizontally as well as vertically.



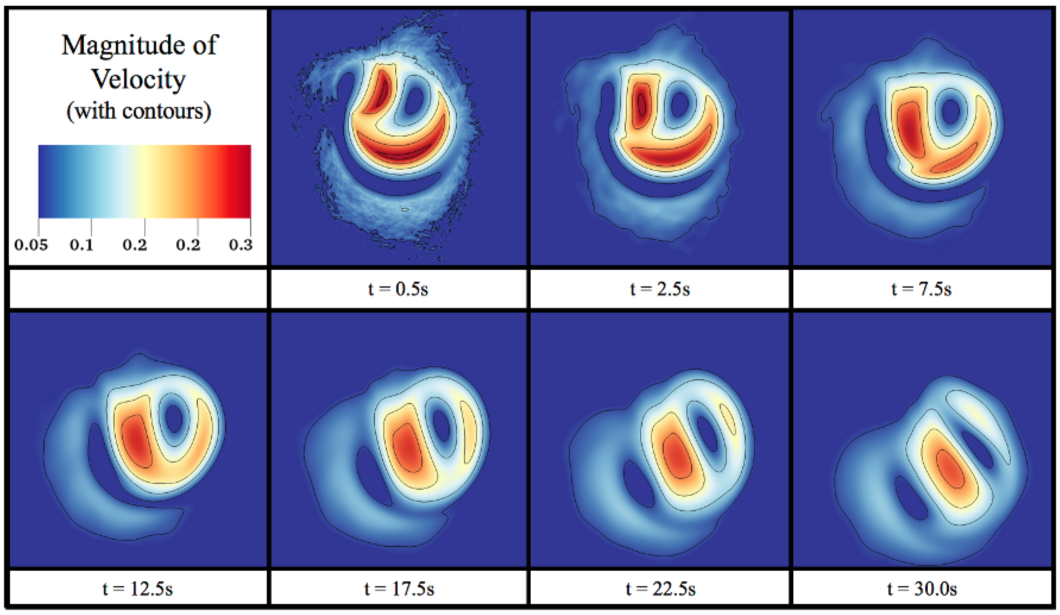
(a)



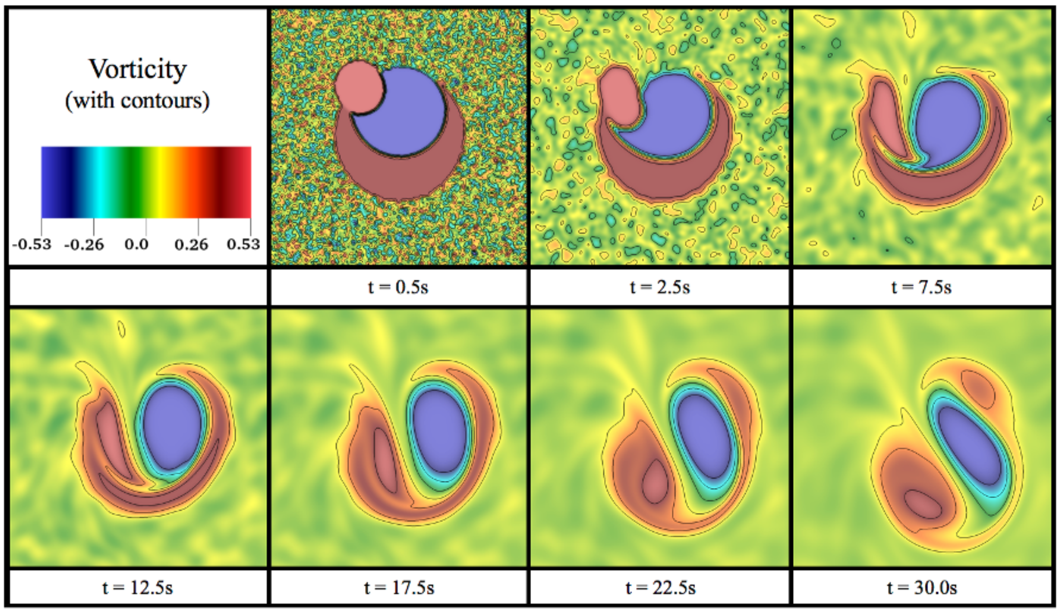
(b)

Figure A5. Simulation data from the case of overlapping vorticity regions during (**a**) the first time-step and (**b**) the final time-step.

Figure A6 provides snapshots over the simulation to illustrate how the magnitude of velocity and vorticity evolved over time. Figure A6a shows that the overlapping regions of vorticity induce the highest flows overall, as quantified by magnitude of velocity. That is, although the background was initialized to random values between [−1,1], which may include values that are higher than initial vorticity of the overlapping regions (+0.4, −0.5, and +0.5 for largest to smallest, respectively), it did not significantly contribute to bulk flow within the domain. Moreover, as suggested earlier, the initial random vorticity configuration quickly dissipates itself out, see Figure A6b.



(a)



(b)

Figure A6. The evolution of (**a**) the magnitude of velocity (with its contours) and (**b**) vorticity (with its contours) during the course of the simulation.

Students may elect to try the following:

1. Recreate the above results with the parameters listed in Table A1 or for different ν or computational domains.
2. Change the initial vorticity in each 'overlapping' region.
3. Change the placement of where each vorticity region is.
4. Modify the example to have more/fewer overlapping regions of vorticity.
5. Try initializing a completely random background vorticity without any other vorticity structures.

References

1. Fefferman, C.L. Existence and Smoothness of the Navier-Stokes Equation. In *The Millenium Prize Problems*; Clay Mathematics Institute: Cambridge, MA, USA, 2006; pp. 57–67.
2. Chorin, A.J. The numerical solution of the Navier-Stokes equations for an incompressible fluid. *Bull. Am. Math. Soc.* **1967**, *73*, 928–931. [CrossRef]
3. Chorin, A.J. Numerical Solution of the Navier-Stokes Equations. *Math. Comp.* **1968**, *22*, 745–762. [CrossRef]
4. Temam, R. Une méthode d'approximation de la solution des équations de Navier-Stokes. *Bull. Soc. Math. Franc.* **1968**, *96*, 115–152. [CrossRef]
5. Brown, D.L.; Cortez, R.; Minion, M.L. Accurate Projection Methods for the Incompressible Navier-Stokes Equations. *J. Comp. Phys.* **2001**, *168*, 464–499. [CrossRef]
6. Griffith, B.E. An accurate and efficient method for the incompressible Navier-Stokes equations using the projection method as a preconditioner. *J. Comput. Phys.* **2009**, *228*, 7565–7595. [CrossRef]
7. Costa, B. Spectral Methods for Partial Differential Equations. *CUBO Math. J.* **2004**, *6*, 1–32.
8. Uecker, H. A short ad hoc introduction to spectral methods for parabolic PDE and the Navier-Stokes equations, 2009. In Proceedings of the Lecture given at International Summer School Modern Computational Science, Oldenburg, Germany, 16–28 August 2009.
9. Suzuki, M. Fourier-Spectal Methods For Navier-Stokes Equations in 2D, 2014. Available online: http://www.math.mcgill.ca/gantumur/math595f14/NSMashbat.pdf (accessed on 29 June 2019).
10. Hardy, J.; Pomeau, Y.; de Pazzis, O. Time evolution of a two-dimensional classical lattice system. *Phys. Rev. Lett.* **1973**, *31*, 276–279. [CrossRef]
11. Chen, S.; Doolen, G.D. Lattice Boltzmann method for fluid flows. *Annu. Rev. Fluid Mech.* **1998**, *30*, 282–300. [CrossRef]
12. Succi, S. *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*; Oxford University Press: Oxford, UK, 2001.
13. Stern, F.; Xing, T.; Yarbrough, D.B.; Rothmayer, A.; Rajagopalan, G.; Prakashotta, S.; Caughey, D.; Bhaskaran, R.; Smith, S.; Hutchings, B.; et al. Hands-On CFD Educational Interface for Engineering Courses and Laboratories. *J. Eng. Edu.* **2006**, *95*, 63–83. [CrossRef]
14. Cummings, R.; Morton, S., Computational Aerodynamics Goes to School: A Course in CFD for Undergraduate Students. In Proceedings of the 43rd AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, USA, 10–13 January 2005; AIAA: Reston, VA, USA,2005. [CrossRef]
15. Ormiston, S.J. Incorporating CFD into the undergraduate Mechanical Engineering Programme at the University of Manitoba. In Proceedings of the Ninth Annual Conference of the CFD Society of Canada: CFD2001, Waterloo, ON, Canada, 27–29 May 2001; Schneider, G., Ed.; CFD Society of Canada: Waterloo, ON, Canada, 2001; pp. 333–337.
16. Aung, K. Design and Implementation of an Undergraduate Computational Fluid Dynamics (Cfd) Course, 2003. In Proceedings of the 2003 American Society for Engineering Education Annual Conference, Nashville, TN, USA, 22–25 June 2003. Available online: https://peer.asee.org/design-and-implementation-of-an-undergraduate-computational-fluid-dynamics-cfd-course.pdf (accessed on 9 September 2019)
17. Stern, F.; Xing, T.; Yarbrough, D.; Rothmayer, A.; Rajagopalan, G.; Otta, S.P.; Caughey, D.; Bhaskaran, R.; Smith, S.; Hutchings, B.; et al. Development of hands-on CFD educational interface for undergraduate engineering courses and laboratories. In Proceedings of the 2004 American Society for Engineering Education Annual Conference & Exposition, Salt Lake City, UT, USA, 20–23 June 2004; American Society for Engineering Education: Washington, DC, USA, 2004; pp. 1526–1555.
18. Adair, D. Incorporation of Computational Fluid Dynamics into a Fluid Mechanics Curriculum. In *Advances in Modeling of Fluid Dynamics*; Liu, C., Ed.; IntechOpen: London, UK, 2012; Chapter 5, pp. 97–122.
19. Stern, F.; Yoon, H.; Yarbrough, D.; Okcay, M.; Oztekin, B.U.; Roszelle, B. Hands-on integrated CFD educational interface for introductory fluids mechanics. *Int. J. Aerodyn.* **2012**, *2*, 339–371. [CrossRef]
20. Ray, B.; Bhaskaran, R. Integrating Simulation into the Engineering Curriculum: A Case Study. *Int. J. Mech. Eng. Edu.* **2013**, *41*, 269–280. [CrossRef]
21. Eldredge, J.D.; Senocak, I.; Dawson, P.; Canino, J.; Liou, W.; LeBeau, R.; Hitt, D.; Rumpfkeil, M.; Cummings, R. A Best Practices Guide to CFD Education in the Undergraduate Curriculum. *Int. J. Aerodyn.* **2014**, *4*, 200–236. [CrossRef]

22. Heron, P.; McNeill, L. Phys21: Preparing Physics Students for 21st-Century Careers (A Report by the Joint Task Force on Undergraduate Physics Programs), 2016. American Physical Society and the American Association of Physics Teachers. Available online: https://www.compadre.org/JTUPP/report.cfm (accessed on 7 January 2020).
23. Heron, P.; McNeill, L. Preparing Physics Students for 21st-Century Careers. *Phys. Today* **2017**, *70*, 38.
24. Fefferman, C.L. A Comparison of C, MATLAB, and Python as Teaching Languages in Engineering. In *Computational Science—ICCS 2004*; Bubak, M., van Albada, G.D., Sloot, P.M., Dongarra, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 1210–1217.
25. Spencer, R.L. Teaching computational physics as a laboratory sequence. *Am. J. Phys.* **2005**, *73*, 151–153. [CrossRef]
26. Peng, L.; Bao, L.; Huang, M. Application of Matlab/Simulink Software in Physics. In *High Performance Networking, Computing, and Communication Systems*; Wu, Y., Ed.; Springer: Berlin/Heidelberg, Germany, 2011; Chapter 21, pp. 140–146.
27. Sangwin, C.J.; O'Toole, C. Computer programming in the UK undergraduate mathematics curriculum. *Int. J. Math. Edu. Sci. Technol.* **2017**, *48*, 1133–1152. [CrossRef]
28. Wang, Y.; Hill, K.J.; Foley, E.C. Computer programming with Python for industrial and systems engineers: Perspectives from an instructor and students. *Comput. Appl. Eng. Educ.* **2017**, *25*, 800–811. [CrossRef]
29. MATLAB. *version 8.5.0 (R2015a)*; The MathWorks Inc.: Natick, MA, USA, 2015.
30. Van Rossum, G. *Python*; version 3.5. 2015. Available online: https://www.python.org (accessed on 31 August 2019).
31. Carey, M.A.; Papin, J.A. Ten simple rules for biologists learning to program. *PLoS Comput. Biol.* **2018**, *14*, e1005871. [CrossRef]
32. Battista, N.A.; Strickland, W.C.; Miller, L.A. IB2d: A Python and MATLAB implementation of the immersed boundary method. *Bioinspir. Biomim.* **2017**, *12*, 036003. [CrossRef]
33. Barba, L.A.; Forsyth, G. CFD Python: The 12 steps to Navier-Stokes equations. *J. Open Source Edu.* **2018**, *1*, 21. [CrossRef]
34. Barba, L.A.; Mesnard, O. Aero Python: Classical aerodynamics of potential flow using Python. *J. Open Source Edu.* **2019**, *2*, 45. [CrossRef]
35. Battista, N.A.; Mizuhara, M.S. Fluid-Structure Interaction for the Classroom: Speed, Accuracy, Convergence, and Jellyfish! *arXiv* **2019**, arXiv:1902.07615.
36. Battista, N. Fluid-structure Interaction for the Classroom: Interpolation, Hearts, and Swimming! *SIAM Rev.* **2018**, in press.
37. Battista, N.A.; Strickland, W.C.; Barrett, A.; Miller, L.A. IB2d Reloaded: A more powerful Python and MATLAB implementation of the immersed boundary method. *Math. Methods Appl. Sci.* **2018**, *41*, 8455–8480. [CrossRef]
38. Kluyver, T.; Ragan-Kelley, B.; Pérez, F.; Granger, B.; Bussonnier, M.; Frederic, J.; Kelley, K.; Hamrick, J.; Grout, J.; Corlay, S.; et al. Jupyter Notebooks—A publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*; Loizides, F., Schmidt, B., Eds.; IOS Press: Amsterdam, The Netherlands: 2016; pp. 87–90.
39. Pawar, S.; San, O. CFD Julia: A Learning Module Structuring an Introductory Course on Computational Fluid Dynamics. *Fluids* **2019**, *4*, 159. [CrossRef]
40. Bezanson, J.; Edelman, A.; Karpinski, S.; Shah, V.B. Julia: A fresh approach to numerical computing. *SIAM Rev.* **2017**, *59*, 65–98. [CrossRef]
41. Childs, H.; Brugger, E.; Whitlock, B.; Meredith, J.; Ahern, S.; Pugmire, D.; Biagas, K.; Miller, M.; Harrison, C.; Weber, G.H.; et al. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*; Bethel, E.W., Childs, H., Hansen, C., Eds.; Chapman and Hall/CRC: Boca Raton, FL USA, 2012; pp. 357–372.
42. Ahrens, J.; Gerveci, B.; Law, C. *ParaView: An End-User Tool for Large Data Visualizations*; Elsevier: Atlanta, GA, USA, 2005.
43. Burden, R.L.; Faires, J.D. *Numerical Analysis*, 5th ed.; Prindle, Weber and Schmidt: Boston, MA USA, 1993.
44. Minion, M.L. Higher-Order Semi-Implicit Projection Methods. In *Numerical Simulations of Incompressible Flows*; Hafez, M.M., Ed.; World Scientific Publishing Company: Waterloo, ON, Canada, 2003; pp. 126–140.

45. Guermond, J.L.; Minev, P.; Shen, J. An overview of projection methods for incompressible flows. *Comp. Methods Appl. Mech. Eng.* **2006**, *195*, 6011–6045. [CrossRef]
46. Almgren, A.S.; Aspden, A.J.; Bell, J.B.; Minion, M.L. On the Use of Higher-Order Projection Methods for Incompressible Turbulent Flow. *SIAM J. Sci. Comput.* **2013**, *35*, B25–B42. [CrossRef]
47. Bell, J.B.; Colella, P.; Glaz, H.M. A second order projection method for the incompressible Navier-Stokes equations. *J. Comput. Phys.* **1989**, *85*, 257–283. [CrossRef]
48. Atkinson, K.E. *An Introduction to Numerical Analysis*, 2nd ed.; John Wiley & Sons: Hoboken, NJ, USA, 1989.
49. Trefethen, L.N. *Spectral Methods in MATLAB*; SIAM: Philadelphia, PA, USA, 2001.
50. Battista, N.A. Spectrally Accurate Initial Data in Numerical Relativity. Master's Thesis, Rochester Institute of Technology, Rochester, NY, USA, 2010.
51. Crank, J.; Nicholson, P. A practical method for numerical evaluation of solutions of partial differential equations of the heat conduction type. *Proc. Camb. Phil. Soc.* **1947**, *43*, 50–67. [CrossRef]
52. Thomas, J.W. *Numerical Partial Differential Equations: Finite Difference Methods*; Springer: New York, NY, USA, 1995.
53. Battista, N.A.; Baird, A.J.; Miller, L.A. A Mathematical Model and MATLAB Code for Muscle-Fluid-Structure Simulations. *Integr. Comp. Biol.* **2015**, *55*, 901–911. [CrossRef]
54. Zhang, J. Lattice Boltzmann method for microfluidics: Models and applications. *Microfluid. Nanofluidics* **2011**, *10*, 1–28. [CrossRef]
55. Bao, Y.B.; Meskas, J. Lattice Boltzmann Method for Fluid Simulations, 2011. Available online: http://www.cims.nyu.edu/~billbao/report930.pdf (accessed on 19 September 2019).
56. Tu, J.; Yeoh, G.H.; Liu, C. *Computational Fluid Dynamics*, 3rd ed.; Butterworth-Heinemann: Oxford, UK, 2018.
57. Ishihara, I. Tests for color blindness. *Am. J. Ophthal.* **1918**, *1*, 457. [CrossRef]
58. Shadden, S.C.; Lekien, F.; Marsden, J.E. Definition and properties of Lagrangian coherent structures from finite-time Lyapunov exponents in two-dimensional aperiodic flows. *Physica D* **2005**, *212*, 271–304. [CrossRef]
59. Shadden, S.C. Lagrangian Coherent Structures: Analysis of Time Dependent Dynamical Systems Using Finite-Time Lyapunov Exponent, 2005. Available online: https://shaddenlab.berkeley.edu/uploads/LCS-tutorial/index.html (accessed on 19 September 2019).
60. Shadden, S.C.; Katija, K.; Rosenfeld, M.; Marsden, J.E.; Dabiri, J.O. Transport and stirring induced by vortex formation. *J. Fluid Mech.* **2007**, *593*, 315–331. [CrossRef]
61. Haller, G.; Sapsis, T. Lagrangian coherent structures and the smallest finite-time Lyapunov exponent. *Chaos* **2011**, *21*, 023115. [CrossRef]
62. Shadden, S.C.; Dabiri, J.O.; Marsden, J.E. Lagrangian analysis of fluid transport in empirical vortex ring flows. *Phys. Fluids* **2006**, *18*, 047105. [CrossRef]
63. Lukens, S.; Yang, X.; Fauci, L. Using Lagrangian coherent structures to analyze fluid mixing by cilia. *Chaos* **2010**, *20*, 017511. [CrossRef]
64. Cheryl, S.; Glatzmaier, G.A. Lagrangian coherent structures in the California Current System—Sensitivities and limitations. *Geophys. Astrophys. Fluid Dyn.* **2012**, *106*, 22–44.
65. Haller, G. Finding finite-time invariant manifolds in two-dimensional velocity fields. *Chaos* **2000**, *10*, 99–108. [CrossRef]
66. Haller, G. Lagrangian Coherent Structures. *Annu. Rev. Fluid Mech.* **2015**, *47*, 137–162. [CrossRef]
67. Truskey, G.A.; Yuan, F.; Katz, D.F. *Transport Phenomena in Biological Systems*; Pearson Prentice Hall Bioengineering: Upper Saddle River, NJ, USA, 2004.
68. Rayleigh, L. On the flow of compressible fluid past an obstacle. *Lond. Edinb. Dublin Philos. Mag. J. Sci.* **1916**, *32*, 1–6. [CrossRef]
69. Acheson, D.J. *Elementary Fluid Dynamics*; Oxford University Press: Oxford, UK, 1990.
70. Batchelor, G.K. *An Introduction to Fluid Dynamics*; Cambridge University Press: Cambridge, UK, 2000.
71. Morton, C.; Yarusevych, S. Vortex shedding in the wake of a step cylinder. *Phys. Fluids* **2010**, *22*, 083602. [CrossRef]
72. Bao, Y.; Wu, Q.; Zhou, D. Numerical investigation of flow around an inline square cylinder array with different spacing ratios. *Comput. Fluids* **2012**, *55*, 118–131. [CrossRef]
73. Carini, M.; Gianetti, F.; Auteri, F. On the origin of the flip-flop instability of two side-by-side cylinder wakes. *J. Fluid Mech.* **2014**, *742*, 552–576. [CrossRef]

74. Younis, M.Y.; Alam, M.M.; Zhou, Y. Flow around two non-parallel tandem cylinders. *Phys. Fluids* **2016**, *28*, 125106. [CrossRef]
75. Gao, Y.; Chen, W.; Wang, B.; Wang, L. Numerical simulation of the flow past six-circular cylinders in rectangular configurations. *J. Mar. Sci. Technol.* **2019**, 1–25. [CrossRef]
76. Ji, C.; Cui, Y.; Xu, D.; Yang, X.; Srinil, N. Vortex-induced vibrations of dual-step cylinders with different diameter ratios in laminar flows. *Phys. Fluids* **2019**, *31*, 073602.
77. Ji, C.; Yang, X.; Yu, Y.; Cui, Y.; Srinil, N. Numerical simulations of flows around a dual step cylinder with different diameter ratios at low Reynolds number. *Eur. J. Mech. B/Fluids* **2019**, in press. [CrossRef]
78. Fransson, J.H.; Konieczny, P.; Alfredsson, P.H. Flow around a porous cylinder subject to continuous suction or blowing. *J. Fluids Stuct.* **2004**, *19*, 1031–1048. [CrossRef]
79. Chen, X.; Yu, P.; Winoto, S.; Low, H. Numerical analysis for the flow past a porous square cylinder based on the stress-jump interfacial-conditions. *Int. J. Num. Meth. Heat Fluid Flow* **2008**, *18*, 635–655. [CrossRef]
80. Naito, H.; Fukagata, K. Numerical simulation of flow around a circular cylinder having porous surface. *Phys. Fluids* **2012**, *24*, 117102. [CrossRef]
81. Shahsavari, S.; Wardle, B.L.; McKinley, G.H. Interception efficiency in two-dimensional flow past confined porous cylinders. *Chem. Eng. Sci.* **2014**, *116*, 752–762. [CrossRef]
82. Ledda, P.G.; Siconolfi, L.; Viola, F.; Gallaire, F.; Camarri, S. Suppression of von Kármán vortex streets past porous rectangular cylinders. *Phys. Rev. Fluids* **2007**, *3*, 103901. [CrossRef]
83. Gupta, G.K. Computer literacy: Essential in today's computer-centric world. *ACM SIGCSE Bull.* **2005**, *38*, 115–119. [CrossRef]
84. Shein, E. Should everybody learn to code? *Commun. ACM* **2014**, *57*, 16–18.
85. Sterling, L. Coding in the curriculum: Fad or foundational? *ACER Res. Conf.* **2016**, *4*, 72–84.
86. Baker, M. Scientific computing: Code alert. *Nature* **2017**, *541*, 563–565. [CrossRef]
87. Helmholtz, H. Uber Integrale der hydrodynamischen Gleichungen, welcher der Wirbelbewegungen entsprechen. *J. Reine Angew. Math.* **1858**, *55*, 25–50.
88. Bladel, J. *On Helmholtz's Theorem in Finite Regions*; Midwestern Universities Research Association: Madison, WI, USA, 1958.
89. Cooley, J.W.; Tukey, J.W. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.* **1965**, *19*, 297–301. [CrossRef]
90. Bhatnagar, P.L.; Gross, E.P.; Krook, M. A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems. *Phys. Rev.* **1954**, *94*, 511–525. [CrossRef]
91. Asinari, P. Multi-Scale Analysis of Heat and Mass Transfer in Mini/Micro Structures. Ph.D. Thesis, Energy Engineering, Politecnico di Torino, Turin, Italy, 2005.