INCOMPRESSIBLE LID-DRIVEN CAVITY FLOW SIMULATION

Christopher McCormick
M.S. Aerospace Engineering
University of California, Los Angeles
chrismccormick@g.ucla.edu
+1 (949) 444-9470

ABSTRACT

Developing an incompressible flow solver is difficult due to complex mathematics and intense computational requirements. The governing equations in fluid mechanics are partial differential equations that must be solved in space and time. With that being said, having the capability of simulating custom fluid systems is very beneficial because it allows for further fluid mechanics research and helps inform the development of engineering designs. Additionally, writing an incompressible flow solver enables an individual to explore numerical methods and gain a deeper understanding of fluid mechanics by discretizing and solving the governing equations. In this work, we write the code for an incompressible lid-driven cavity flow solver. First, we investigate and verify the accuracy and stability properties of different finite differencing and time-stepping methods. We then define a domain and construct a method of visualizing different flow properties. Discrete divergence, gradient, Laplacian, and nonlinear advection operators are then developed, and their outputs are verified by performing a spatial convergence check. A solver is then written that applies these operators to the discretized forms of the governing equations. After a spatial and temporal convergence check of the solver is performed, the direct forcing immersed boundary method is integrated into the solver to simulate a body within the cavity.

***K*eywords** Incompressible Flow Solver · Immersed Boundary Method

1 Introduction

Incompressible flow solvers are useful in studying a variety of fluid systems, such as biological flows, environmental flows, and flows around vehicles. Using Computational Fluid Dynamics (CFD) to solve for different flow properties is valuable for researching systems of interest and calculating forces acting on a body. For this reason, it is advantageous to write an independent CFD solver to customize systems of interest and further an understanding of fluid mechanics and numerical methods. Incompressible CFD solvers calculate flow parameters by numerically solving the governing momentum conservation, or Navier-Stokes (N-S), equations,

$$\rho\Big[\frac{\partial \mathbf{u}}{\partial t} + (\nabla \cdot \mathbf{u})\mathbf{u}\Big] = -\nabla p + \mu\nabla^2\mathbf{u} + \rho\mathbf{g}, \tag{1}$$

and the mass conservation, or continuity, equation,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho\mathbf{u}) = 0 \tag{2}$$

These equations are solved using finite differencing and time-stepping techniques, which vary in accuracy and stability properties. Some methods exhibit dispersive errors, where the solutions become erratic and more unstable with each time step, while other methods exhibit diffusive errors, where the solutions progressively smooth with each time step. Similar to those presented in Professor Taira's book [1], examples of dispersive and diffusive errors are observed in Fig. 1. The following section introduces and explores various time-stepping methods and compares methods based on accuracy and computational costs.

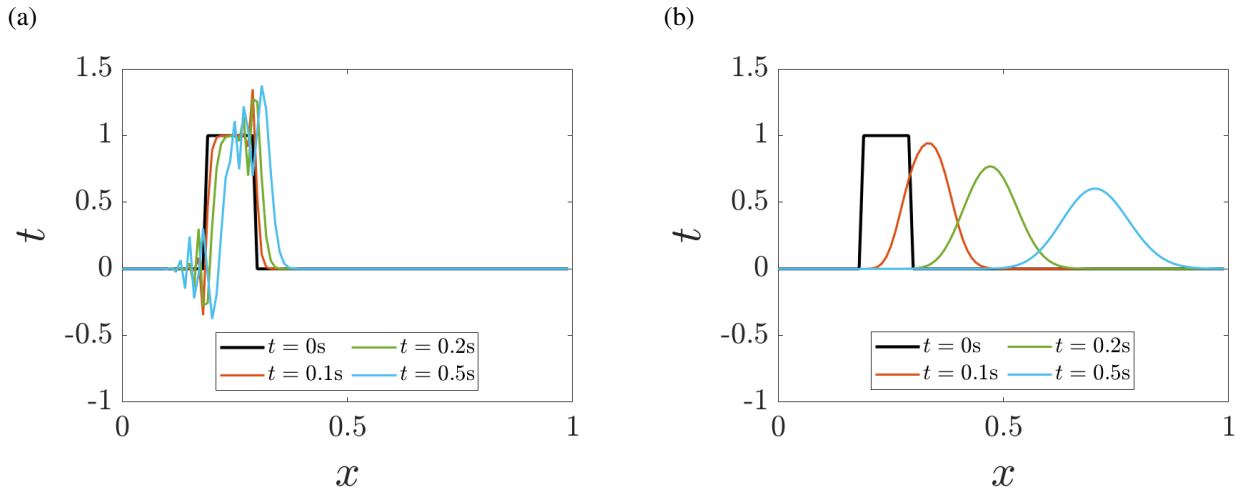

Figure 1: Types of error arising from different finite differencing and time-stepping methods. **(a)** Dispersive error due to central finite differencing. **(b)** Diffusive error due to upwind finite differencing.

2 Finite Difference and Time-stepping Methods

As mentioned, different numerical methods exhibit different accuracy and stability properties. All the following methods will be discussed as they are applied to the advection equation,

$$\frac{\partial f}{\partial t} + c\frac{\partial f}{\partial x} = 0. \tag{3}$$

This equation can be explored using three finite-differencing schemes: downwind differencing, central differencing, and upwind differencing. Downwind differencing, expressed as,

$$\left.\frac{\partial f}{\partial x}\right|_j = \frac{f_{j+1} - f_j}{\Delta x},$$

is generally not used in fluid flow simulations because the error exhibits negative diffusivity. In other words, with each time step, the solution becomes sharper, resulting in the solution blowing up at late times. Central differencing,

$$\left.\frac{\partial f}{\partial x}\right|_j = \frac{f_{j+1} - f_{j-1}}{2\Delta x},$$

is sometimes used in fluid flow simulations, but the error from this scheme behaves dispersively. This means that with each time step, the solution becomes increasingly more erratic, resulting in the solution blowing up at late times. This can be mitigated by decreasing the step size, but the manner of the error cannot be escaped. Upwind differencing,

$$\left.\frac{\partial f}{\partial x}\right|_j = \frac{f_j - f_{j-1}}{\Delta x},$$

is also sometimes used in fluid flow simulations because of its error exhibits a positive diffusivity. This means that the solution becomes smoother with each time step rather than blowing up. In the following subsections, five different time-stepping methods will be explored using central and upwind finite differencing schemes. The properties of each of the five time-stepping methods will be compared, and a verification of each method's accuracy will be presented. Additionally, a spatial convergence check was performed for the different central and upwind differencing schemes, as seen in Fig. 2. As expected, central differencing scheme produced an error slopes of approximately two, while the upwind differencing scheme produced an error slopes of approximately one.

Note that all five methods are explicit methods, meaning that the solution at the next time step is calculated using information from the current time step. For this reason, the accuracy of all the following methods can be improved with smaller spatial and temporal step sizes. Additionally, all these methods are constrained by the Courant-Friedrichs-Lewy (CFL) condition,

$$\text{CFL} = \frac{c\Delta t}{\Delta x} \leq 1. \tag{4}$$

This condition indicates that information traveling at a speed c over the time step Δt must not translate faster than the step size Δx. If the temporal step size becomes too large and the CFL number exceeds one, the method becomes unstable. This is observed as dips in Fig. 3(a) and 3(b), where the dip indicates the temporal step size at which the CFL number becomes 1. Implicit methods, in which the solution at the next step is calculated using information from current and future time steps, also exist, but will not be explored in this report.

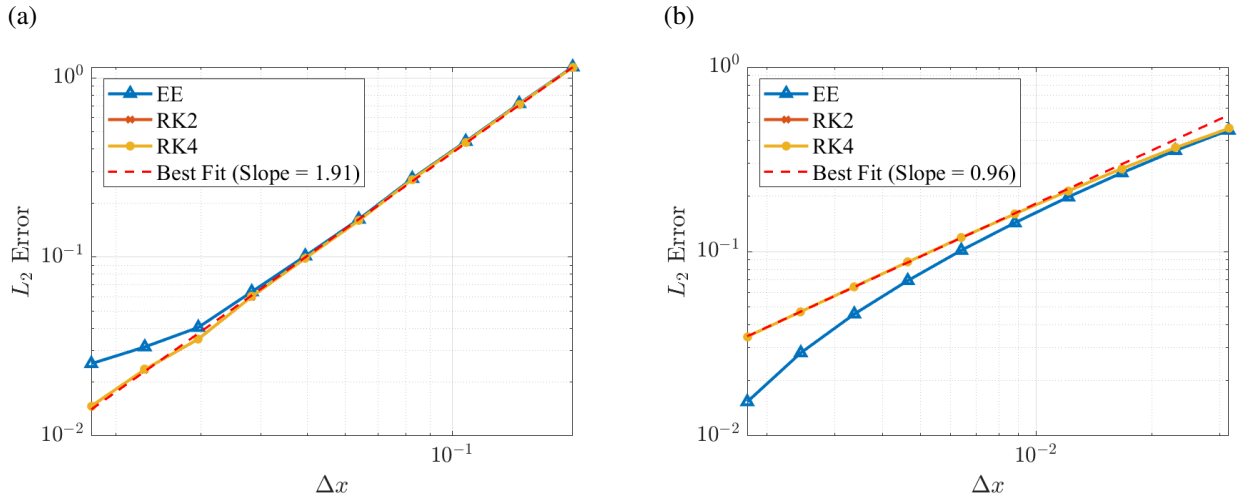

Figure 2: Local spatial errors using a **(a)** central differencing scheme, and **(b)** upwind differencing scheme.

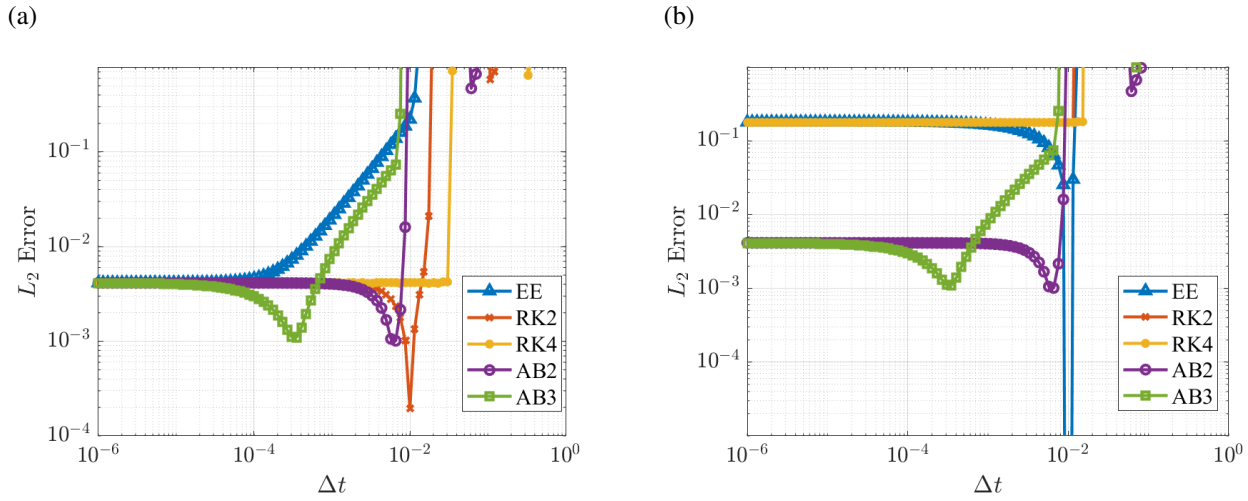

Figure 3: Global errors using a **(a)** $\Delta x = 0.01$ central differencing scheme and a **(b)** $\Delta x = 0.01$ upwind differencing scheme.

2.1 Explicit Euler Method

The simplest method we will explore is the is the explicit Euler method. The explicit Euler method is described by the equation,

$$f^{n+1} = f^n + \Delta t g^n, \tag{5}$$

where g^n is a chosen finite-differencing scheme slope of Eqn. (3). This method is first-order accurate in time, meaning that the error increases linearly with the temporal step size. This is verified when local truncation error is plotted as a function of temporal step size in Fig. 4(a) and 4(b). In this figure, a slope of two is observed, indicating that this method is first-order accurate in time. Due to this linear relationship with step size, the accuracy of this method is highly dependent upon temporal step sizes. This is verified in Fig. 3(a) and 3(b), where the error grows significantly as a function of temporal step size.

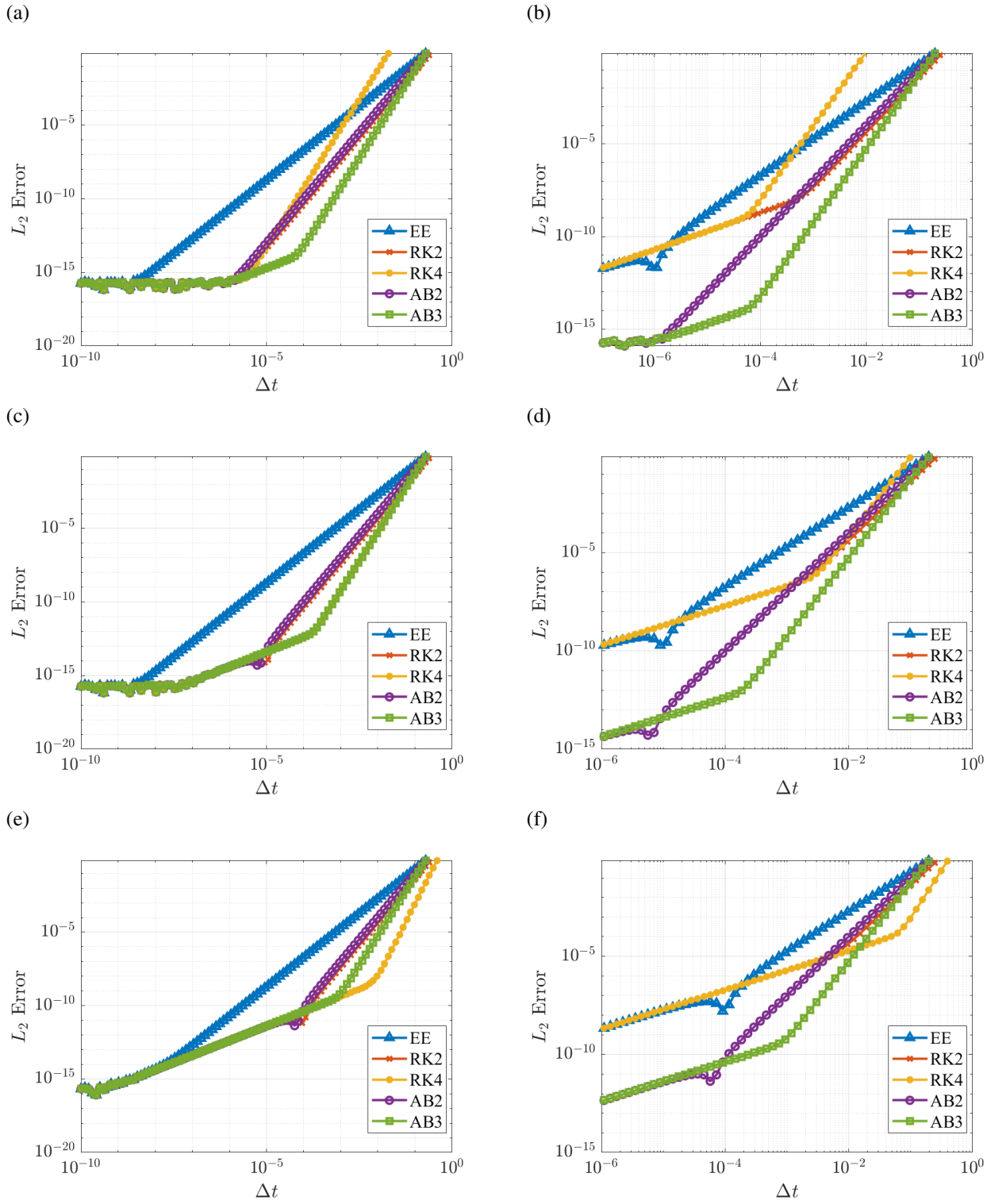

Figure 4: Local temporal errors using a **(a)** $\Delta x = 1 \times 10^{-6}$ central differencing scheme, **(b)** $\Delta x = 1 \times 10^{-6}$ upwind differencing scheme, **(c)** $\Delta x = 1 \times 10^{-5}$ central differencing scheme, **(d)** $\Delta x = 1 \times 10^{-5}$ upwind differencing scheme, **(e)** $\Delta x = 1 \times 10^{-4}$ central differencing scheme, and **(f)** $\Delta x = 1 \times 10^{-4}$ upwind differencing scheme.

2.2 2nd Order Runge-Kutta Method

Next, the two-step, or second-order, Runge-Kutta method will be explored. This method incorporates multiple predictions at a given time step and is described by two steps,

$$\begin{aligned} f^{(1)} &= f^n + \frac{\Delta t}{2} g^n \\ f^{n+1} &= f^{(2)} + \Delta t g^n, \end{aligned} \tag{6}$$

where the first step is essentially a half-step that is used to better inform the full-step estimate. Per its name, the second-order Runge-Kutta method is second-order accurate in time, meaning that the error is proportional to the square of the time step size. This is verified in Fig. 4(a)-(f), where the local truncation error has a slope of three, indicating second-order accuracy in time. Additionally, though it does not require as small of a time step size as the explicit Euler method, this method still requires small step sizes to achieve high accuracy. This is observed in Fig. 3(a) and 3(b), where the error grows as the time step size increases. If second-order accuracy is required, this method provides an easy way of achieving it.

2.3 4th Order Runge-Kutta Method

Similar to the second-order Runge-Kutta method is the four-step, or fourth-order, Runge-Kutta method. This method is the same as the second-order Runge-Kutta method but with more intermediate steps to obtain a better full-step estimate. The fourth-order Runge-Kutta method is described by the four steps,

$$\begin{aligned} f^{(1)} &= f^n + \frac{\Delta t}{2} g^n \\ f^{(2)} &= f^n + \frac{\Delta t}{2} g^{(1)} \\ f^{(3)} &= f^n + \Delta t g^{(2)} \\ f^{n+1} &= f^n + \Delta t \frac{g^n + 2g^{(1)} + 2g^{(2)} + g^{(3)}}{6}, \end{aligned} \tag{7}$$

where, similar to its second-order sibling, each intermediate function is an estimate at a smaller step size to better inform the full-step estimate. This method is fourth-order accurate in time, as verified by Fig. 4(a)-(f) where the slope of the local truncation error is five. For this reason, this method can achieve higher accuracy at larger step sizes when compared to the second-order Runge-Kutta and explicit Euler methods. This is verified in Fig. 3(a) and 3(b), where the error explodes at a larger temporal step size. This method provides an accurate way of time-stepping, but it is not necessary if the problem does not require fourth-order accuracy. The second-order Runge-Kutta method provides a similar estimate at a lower computation cost, but also a lower accuracy.

2.4 2nd Order Adams-Bashforth Method

Adams-Bashforth methods are a group of multi-step methods, where current and past states $(g^n, g^{n-1}, g^{n-2}, \dots)$ are considered in an explicit formulation. When Taylor series expanding f^{n-1} about f^n, and truncating at the third term, the second-order Adams-Bashforth is produced,

$$f^{n+1} = f^n + \Delta t \frac{3g^n - g^{n-1}}{2}. \tag{8}$$

Similar to the second-order Runge-Kutta method, this method is second-order accurate in time. This is verified in Fig. 4(a)-(f), where the slope of the local truncation error is observed to be three. As expected from a second-order method, this Adams-Bashforth method still requires smaller step sizes to achieve accurate results. This is verified in Fig. 3(a) and 3(b), where error remains lower than the first-order explicit Euler method, but significantly increases at higher temporal step sizes. As mentioned, Adams-Bashforth methods are multi-step methods, meaning that information from previous steps is stored to predict future steps. This property presents the potential for this method to become very computationally intensive. For this reason, the second-order Runge-Kutta method may be more appealing if second-order accuracy is required.

2.5 3rd Order Adams-Bashforth Method

Lastly, the third-order Adams-Bashforth method is essentially the second-order method with an additional past state considered. When the Taylor series expansion of f^{n-1} about f^n is truncated at the fourth term, the third-order

Adams-Bashforth method is derived,

$$f^{n+1} = f^n + \Delta t \frac{23g^n - 16g^{n-1} + 5g^{n-2}}{12}. \tag{9}$$

As its name suggests, this method is third-order accurate in time, which is verified by Fig. 4(a)-(f). The local truncation error slope for this method is observed to be four, indicating third-order accuracy. Due to its third-order accuracy, this method should be more accurate than its second-order sibling. However, when observed in in Fig. 3(a) and 3(b), this third-order method appears to perform worse at higher step sizes. This is because the stability region for the third-order Adams-Bashforth method is smaller than that of the second-order method. Again, Adams-Bashforth methods are multi-step methods. Because this method is third-order accurate, an additional past state is required, making this method even more computationally intensive compared to the second-order method. Unless third-order accuracy is required, better methods could likely be found.

3 Staggered Grid Generation

In order to simulate fluid flows, a domain and a grid format to compute over are required. For this incompressible flow solver, a staggered grid is chosen, shown in Fig. 5. Note that the domain and number of grid cells are customizable to the user. In a staggered grid format, u-velocities are defined at the vertical cell walls, v-velocities are defined at the horizontal cell walls, pressures are defined at the cell centers, and vorticities are defined at the cell corners. By having this staggered format, staggered grids avoid the accumulation of mass conservation error and spurious pressure modes, which result in numerical instabilities and high errors. A method of flattening the flow variables was developed, effectively stacking each variable from a two-dimensional grid to a one-dimensional vector.

Using this staggered grid, flow properties are visualized by first interpolating each variable to the same point: the cell corners. In this project, boundary conditions of zero velocities are enforced at the left, right, and bottom walls. After assigning the same random sinusoidal function to all four flow variables, Fig. 6 displays each visualized variable.

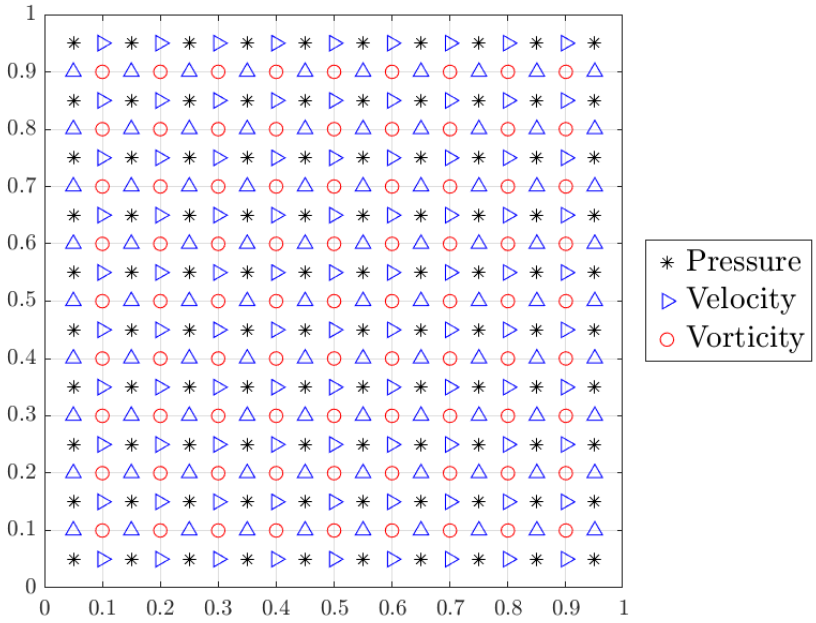

Figure 5: Staggered grid format.

4 Discrete Operators

In order to solve the governing equations to simulate incompressible flow, numerical operators must be defined. In the following section, discrete divergence, gradient, Laplacian, nonlinear advection, and conjugate gradient operators will be described, and their outputs will be verified.

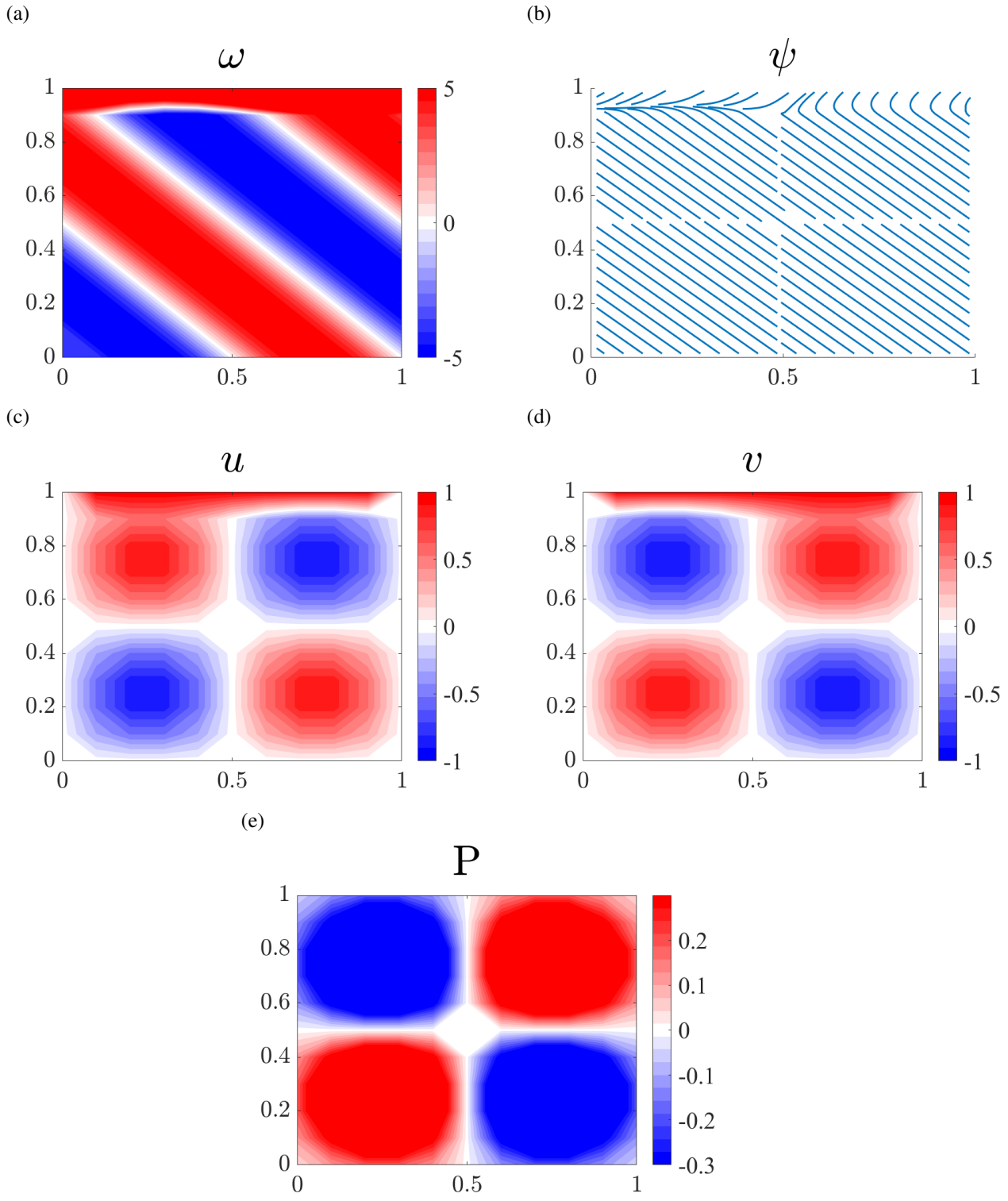

Figure 6: Visualization of flow properties with sinusoidal initial input and boundary conditions applied. **(a)** vorticity, **(b)** streamlines, **(c)** u-velocity, **(d)** v-velocity, **(e)** pressure.

4.1 Divergence Operator

The first discrete operator to be constructed is the discrete divergence operator. The discrete divergence operator is described using the equation,

$$D_{i,j} = \frac{u_{i+1,j} - u_{i,j}}{\Delta x} + \frac{v_{i,j+1} - v_{i,j}}{\Delta y}. \tag{10}$$

Note that the divergence operation decreases the rank of our input. This means that the discrete divergence operator only uses one equation to describe the divergence operation. As with all the operators described in this section, the discrete divergence operator uses a spatial central differencing scheme. This means that all of the discrete operators are first-order accurate, which is verified by the L_2 error vs. spatial step size convergence plot in Fig. 7. In the figure, a slope of two is observed for all the discrete operators, indicating first-order accuracy.

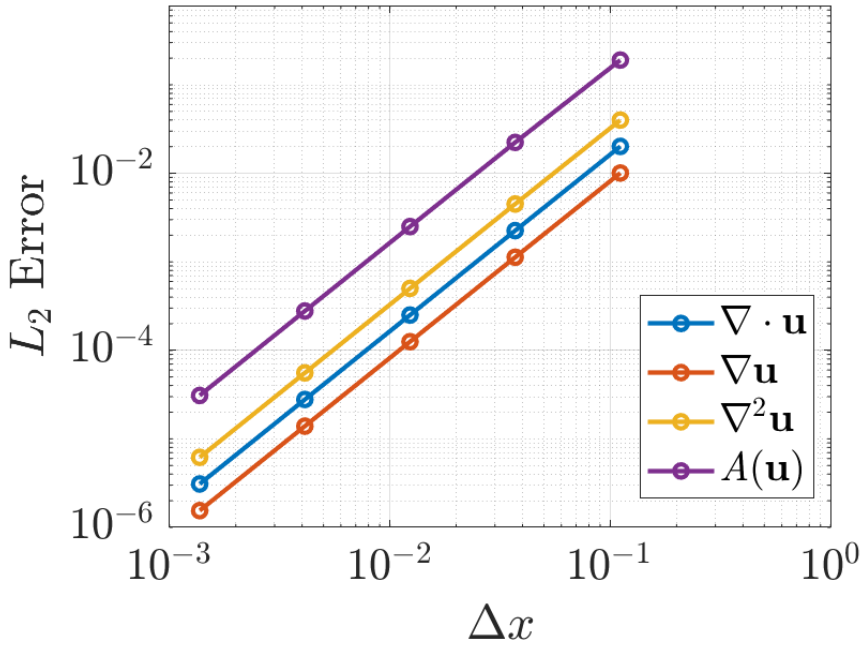

Figure 7: Convergence of the discrete operators.

4.2 Gradient Operator

The discrete gradient operator must also be constructed for our incompressible flow solver. The gradient operator is described using the two equations,

$$\begin{aligned} G^x_{i,j} &= \frac{p_{i,j} - p_{i-1,j}}{\Delta x} \\ G^y_{i,j} &= \frac{p_{i,j} - p_{i,j-1}}{\Delta y}. \end{aligned} \tag{11}$$

Unlike the divergence operator, the gradient operator does not decrease the rank of our input. Therefore, two equations are required to describe the gradient operation. As mentioned, the gradient operation is first-order accurate, as verified in Fig. 7.

4.3 Laplacian Operator

Next, the discrete Laplacian operator must be constructed. The Laplacian operator is described by the two equations,

$$\begin{aligned} L^x_{i,j} &= \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta x^2} \\ L^y_{i,j} &= \frac{v_{i,j-} - 2v_{i,j} + v_{i,j+1}}{\Delta y^2}. \end{aligned} \tag{12}$$

Similar to the gradient operation, the Laplacian operation does not decrease the rank of our input. Therefore, there are two equations in the discrete Laplacian operator. This method is verified first-order accurate in Fig. 7.

4.4 Nonlinear Advection Operator

Lastly, the discrete nonlinear advection operator is also needed. In order to calculate the nonlinear advection, the average velocity around the point of interest must first be calculated. These are calculated according to,

$$\bar{u}^y_N = \frac{u_{i,j+1} + u_{i,j}}{2} \qquad \bar{u}^y_S = \frac{u_{i,j-1} + u_{i,j}}{2} \qquad \bar{u}^x_E = \frac{u_{i+1,j-1} + u_{i+1,j}}{2} \qquad \bar{u}^x_W = \frac{u_{i,j-1} + u_{i,j}}{2},$$
$$\bar{v}^x_N = \frac{v_{i-1,j+1} + v_{i,j+1}}{2} \qquad \bar{v}^x_S = \frac{v_{i-1,j} + v_{i,j}}{2},$$

where superscripts x and y indicate the axis along which a given velocity was averaged, and subscripts N, S, E, and W represent the north, south, east, and west location of a given average velocity relative to the point of interest. Using these average velocities, the nonlinear advection term in the x-direction is calculated using,

$$A^x_{i,j} = \frac{\left(\bar{u}^x_E\right)^2 - \left(\bar{u}^x_W\right)^2}{\Delta x} + \frac{\bar{u}^y_N \bar{v}^x_N - \bar{u}^y_S \bar{v}^x_S}{dy}. \tag{13}$$

Similarly, to calculate the nonlinear advection in the y-direction, the average velocities must first be calculated. These are calculated according to,

$$\bar{v}^y_N = \frac{v_{i,j+1} + v_{i,j}}{2} \qquad \bar{v}^y_S = \frac{v_{i,j-1} + v_{i,j}}{2} \qquad \bar{v}^x_E = \frac{v_{i+1,j} + v_{i+1,j}}{2} \qquad \bar{v}^x_W = \frac{v_{i-1,j} + v_{i,j}}{2},$$
$$\bar{u}^y_E = \frac{u_{i+1,j-1} + u_{i+1,j}}{2} \qquad \bar{u}^y_W = \frac{u_{i,j-1} + u_{i,j}}{2}.$$

Using these equations, the nonlinear advection term in the y-direction is calculated using,

$$A^y_{i,j} = \frac{\left(\bar{v}^y_N\right)^2 - \left(\bar{v}^y_S\right)^2}{\Delta y} + \frac{\bar{u}^y_E \bar{v}^x_E - \bar{u}^y_W \bar{v}^x_W}{dx}. \tag{14}$$

Once these equations are implemented and boundary conditions are met, the discrete operator must be verified. With a slope of two in Fig. 7, this method is verified to be first-order accurate.

4.5 Conjugate Gradient Solver

The conjugate gradient solver is important to have in our simulation of incompressible flows. It allows us to solve for x the equation $Ax = b$. Code for the conjugate gradient solver was written and verified using,

$$A = \begin{bmatrix} 4 & 2 & -1 \\ 2 & 5 & 3 \\ -1 & 3 & 6 \end{bmatrix}, \qquad x_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad \text{and} \quad b = \begin{bmatrix} 9 \\ 9 \\ -1 \end{bmatrix},$$

where x_0 is an initial guess. Analytically solving $A^{-1}b$ for x results in the answer,

$$x = \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}.$$

This is the same answer that the conjugate gradient solver outputs, verifying that it works.

5 Simulation

With all the discrete operators written, the lid-driven cavity flow is now simulated. This is done using the projection, or fractional-step, method proposed by Perot in 1993 [2]. In short, this method calculates the velocity at a fractional time step and uses it to calculate the velocity and pressure at a full time step. Referencing much of Professor Taira's textbook [1], the following subsection discusses this method in further detail.

5.1 Projection Method

In this method, the incompressible NS equations (1) is first written in a discretized form,

$$\frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t} = -GP^{n+1} + \frac{1}{2}(3\mathbf{A}^n - \mathbf{A}^{n-1}) + \frac{1}{2}\nu L(\mathbf{u}^{n+1}+\mathbf{u}^n), \tag{15}$$

where G, $\mathbf{A}$, and L are the discrete operators (11) (13) (14) (12) discussed in the previous section. For simplicity, the notation,

$$R = I - \frac{\Delta t}{2}\nu L, \qquad S = I + \frac{\Delta t}{2}\nu L,$$

is adopted where I is the identity matrix. Using this new notation, the discretized NS equations are rewritten as,

$$R\mathbf{u}^{n+1} + \Delta t G P^{n+1} = S\mathbf{u}^n + \frac{\Delta t}{2}(3\mathbf{A}^n - \mathbf{A}^{n-1}).$$

When these equations are written in a matrix format and the LU decomposition is taken,

$$\begin{bmatrix} R & \Delta t G \\ D & 0 \end{bmatrix} = \begin{bmatrix} R & 0 \\ D & -\Delta t D R^{-1} G \end{bmatrix} \begin{bmatrix} I & \Delta t R^{-1} G \\ 0 & I \end{bmatrix},$$

we arrive at the projection method with velocity boundary conditions applied,

$$\begin{aligned} R\mathbf{u}^F &= S\mathbf{u}^n + \frac{\Delta t}{2}(3\mathbf{A}^n - \mathbf{A}^{n-1}) + \frac{\Delta t \nu}{2}(\mathrm{bc}_L^{n+1} + \mathrm{bc}_L^n), \\ DR^{-1}GP^{n+1} &= \frac{1}{\Delta t} D\mathbf{u}^F + \frac{1}{\Delta t}\mathrm{bc}_D^{n+1}, \\ \mathbf{u}^{n+1} &= \mathbf{u}^F - \Delta t R^{-1} G P^{n+1}. \end{aligned} \tag{16}$$

These equations are then implemented in code, a simulation with a grid size of 129×129 at $Re = 400$ and $Re = 1000$ with CFL $= 0.5$ are run and the results displayed in Fig. 8 and Fig. 9 were obtained.

5.2 Verification of Results

Spatial and temporal convergence checks were performed for the solver to ensure its accuracy. Fig. **??** demonstrates convergence of our solver, with error slopes of about two. This indicates that our solver is first-order accurate, which is what we expect.

Simulations at the same Reynolds number and grid size were run by Ghia $et\ al.$ in 1982 [3] and serve as a reference to compare our results. Velocity profiles at the cavity's geometric center and the primary vortex's center are presented in the paper. Fig. 11 and Fig. 12 displays the velocity profiles obtained from our simulation as they compare to Ghia $et\ al.$'s results.

Additionally, Ghia $et\ al.$ also present vorticity contours and streamlines inside their cavity. Fig. 13 and Fig. 14 display Ghia $et\ al.$'s contours as they compare to our simulation results.

6 Immersed Boundary Method

Now with a working incompressible lid-driven cavity flow solver, an immersed boundary is implemented to investigate flows around 2D structures. In this project, only a cylinder is implemented, but various shapes such as airfoils and flat plates can also be explored. The object must first be defined and discretized in its own Lagrangian, or immersed boundary, grid. Then, a discrete delta function is used to enable communication between the Eulerian, or Cartesian, and Lagrangian grids. For this project, we use the three-grid cell delta function,

$$d_3(x) = \begin{cases} \frac{1}{3}\left[\sqrt{-3(\frac{x}{\Delta s})^2 + 1}\right] & \text{for } |x| \le 0.5\Delta s, \\ \frac{1}{6}\left[5 - 3\frac{|x|}{\Delta s}\sqrt{-3(1-\frac{|x|}{\Delta x})^2 + 1}\right] & \text{for } 0.5\Delta s \le |x| < 1.5\Delta s, \\ 0 & \text{otherwise,} \end{cases} \tag{17}$$

where x is the difference between the Eulerian and Lagrangian grid points, Δx is the spatial step size between Eulerian grid points, and Δs is the spatial step size of the Lagrangian grid.

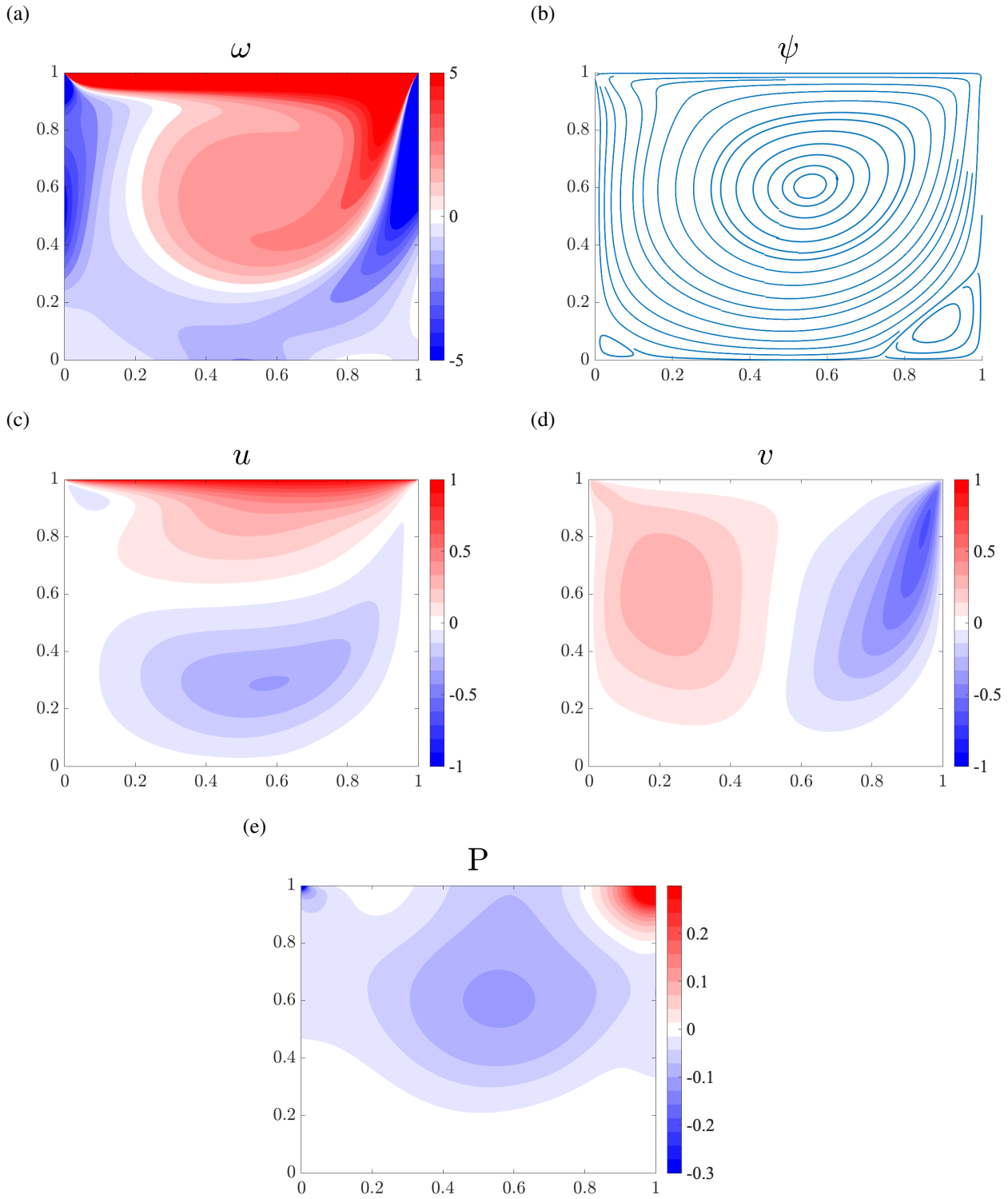

Figure 8: Simulation results at CFL $= 0.5$ and $Re = 400$ with grid size 129×129. **(a)** vorticity, **(b)** streamlines, **(c)** u-velocity, **(d)** v-velocity, **(e)** pressure.

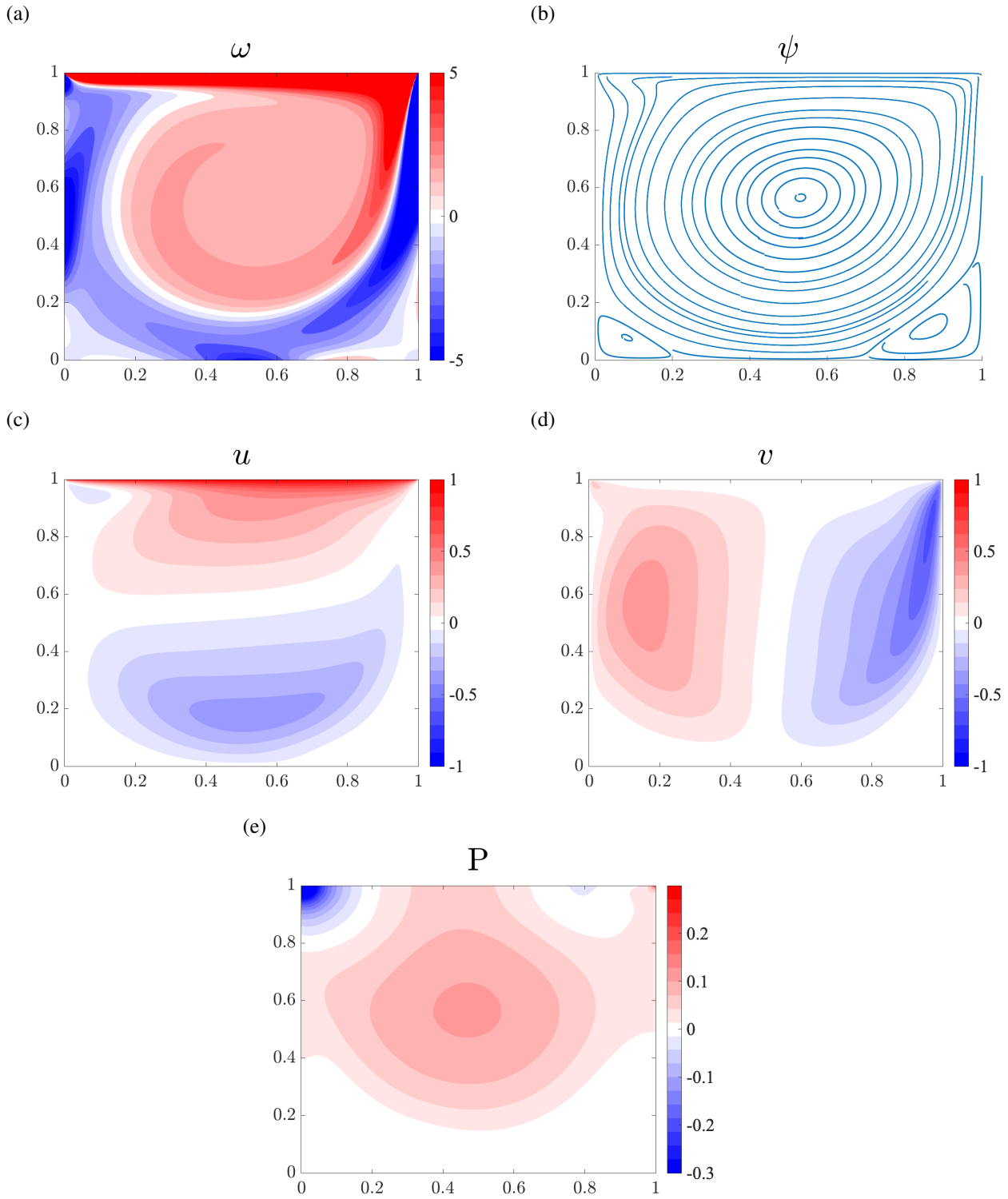

Figure 9: Simulation results at CFL $= 0.5$ and $Re = 1000$ with grid size 129×129. **(a)** vorticity, **(b)** streamlines, **(c)** u-velocity, **(d)** v-velocity, **(e)** pressure.

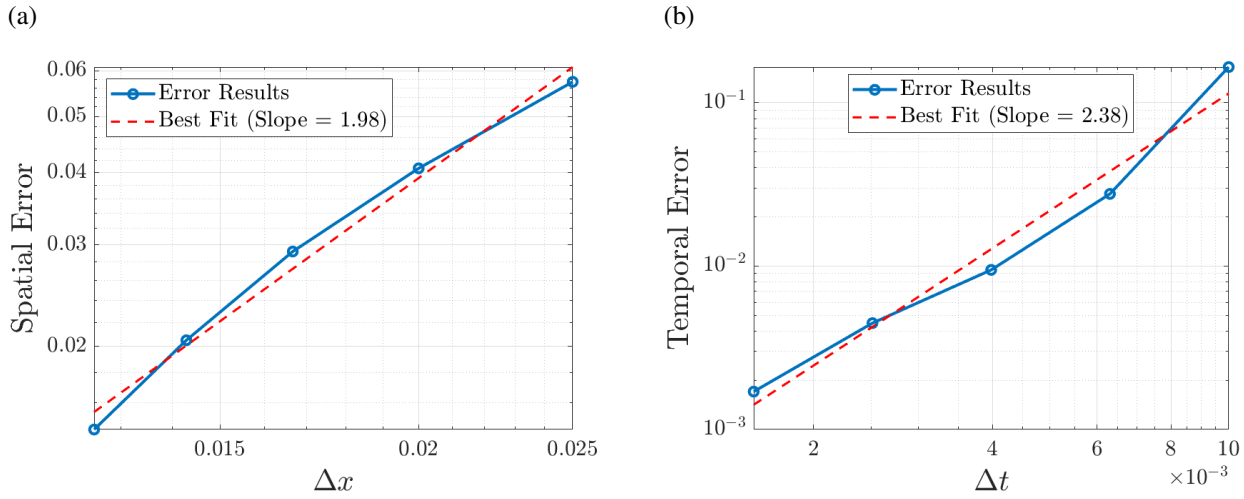

Figure 10: **(a)** Spatial convergence of CFD solver and **(b)** temporal convergence of CFD solver.

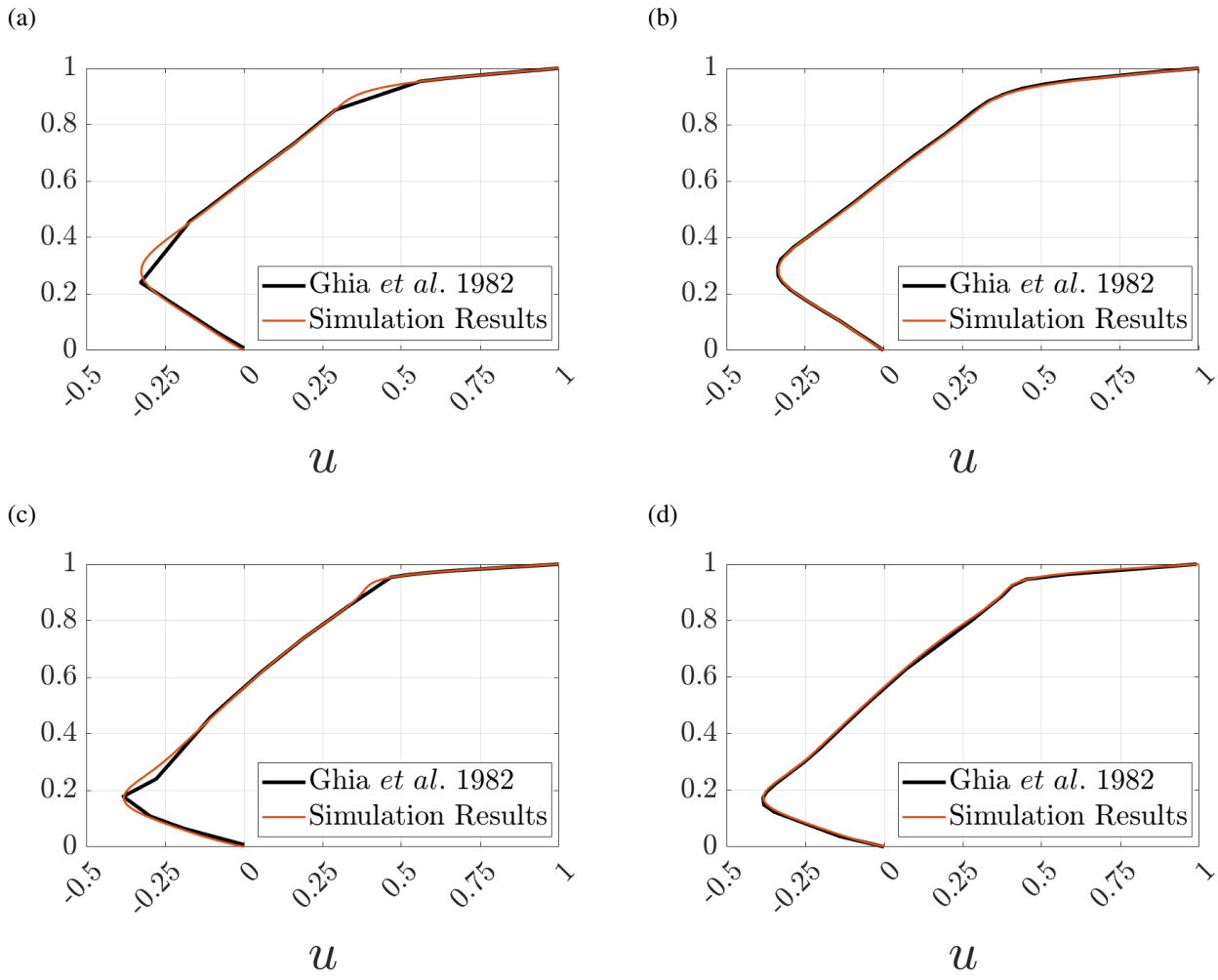

Figure 11: u-Velocity contour slice with a grid size of 129×129 at **(a)** $Re = 400$ through the geometric center of the cavity, **(b)** $Re = 400$ through the primary vortex center, **(c)** $Re = 1000$ through the geometric center of the cavity, and **(d)** $Re = 1000$ through the primary vortex center.

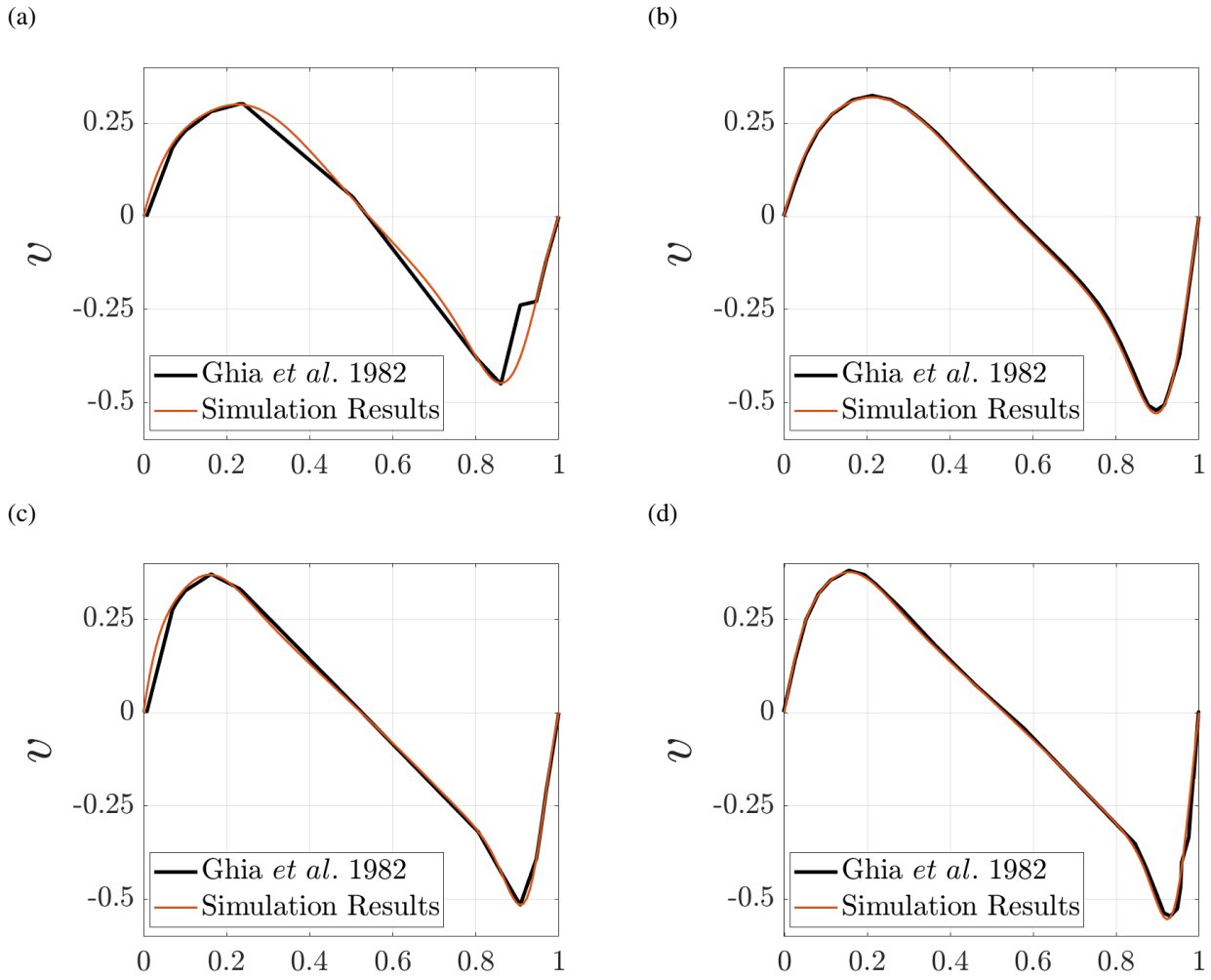

Figure 12: v-Velocity contour slice with a grid size of 129×129 at **(a)** $Re = 400$ through the geometric center of the cavity, **(b)** $Re = 400$ through the primary vortex center, **(c)** $Re = 1000$ through the geometric center of the cavity, and **(d)** $Re = 1000$ through the primary vortex center.

6.1 Interpolation Operators

To implement an immersed boundary in the solver, two new operators must be defined to communicate between the Eulerian and Lagrangian grids. The first operator is defined as,

$$E_{ki} = \alpha d(x_i - \xi_k) d(y_i - \eta_k), \tag{18}$$

where α is equal to equal to $\Delta x \Delta y$, d is the discrete delta function, and ξ_k and η_k are the $x-$ and $y-$ coordinates of immersed boundary points in the Lagrangian grid. This operator uses the fractional-step velocity to calculate the force required at the immersed boundary to satisfy the no-slip condition at the boundary surface. The other operator, which is equal to the transpose of the E operator, is defined as,

$$H_{ik} = \beta d(\xi_k - x_i) d(\eta_k - y_i), \tag{19}$$

where β is equal to $\Delta x \Delta y$. This operator takes the force of the boundary and calculates another fractional velocity after the immersed boundary force is applied.

6.2 Direct Forcing Method

The immersed boundary is implemented using the direct forcing method. This method was chosen due to its easy integration into the existing solver architecture. In between the first and second equations in the projection method, Eqn.

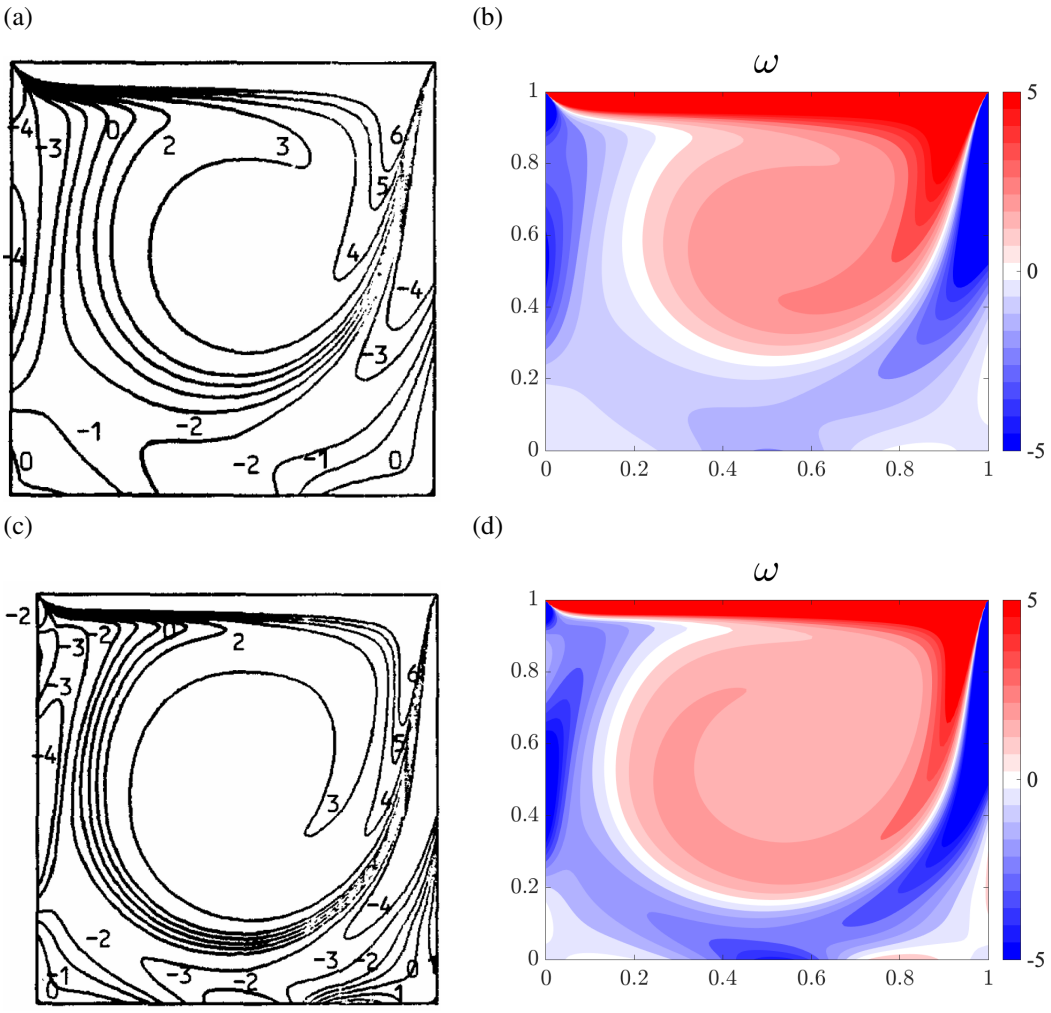

Figure 13: Comparison of vorticity contours **(a)** Ghia *et al.*'s results at $Re = 400$, **(b)** Our results at $Re = 400$, **(c)** Ghia *et al.*'s results at $Re = 1000$, **(d)** Our results at $Re = 1000$.

16, an additional step is implemented. In this step, the force required to satisfy no-slip at the immersed boundary is calculated using,

$$\boldsymbol{F}^n = \frac{\boldsymbol{u}_b^{n+1} - E\boldsymbol{u}^n}{\Delta t} - E(\text{RHS}^n), \tag{20}$$

where RHS is the right-hand side of the first equation in the project method, Eqn. 16. A new fractional velocity is calculated after this force has been applied according to,

$$\boldsymbol{u}^{n+1} = \Delta t(\text{RHS}^n + H\boldsymbol{F}^n) + \boldsymbol{u}^n. \tag{21}$$

This fractional velocity is then used in the last two equations of the projection method. These equations are implemented in the solver, and the results for a grid size of 129×129 at $Re = 400$ are shown in Fig. 15.

6.3 Verification of Results

A spatial convergence check is implemented to check the spatial accuracy of the immersed boundary method solver. Fig. 16 displays an error slope of about two, indicating first-order spatial accuracy, which is expected.

References

[1] Takeo Kajishima and Kunihiko Taira. *Computational Fluid Dynamics: Incompressible Turbulent Flows*. Springer, 2016.

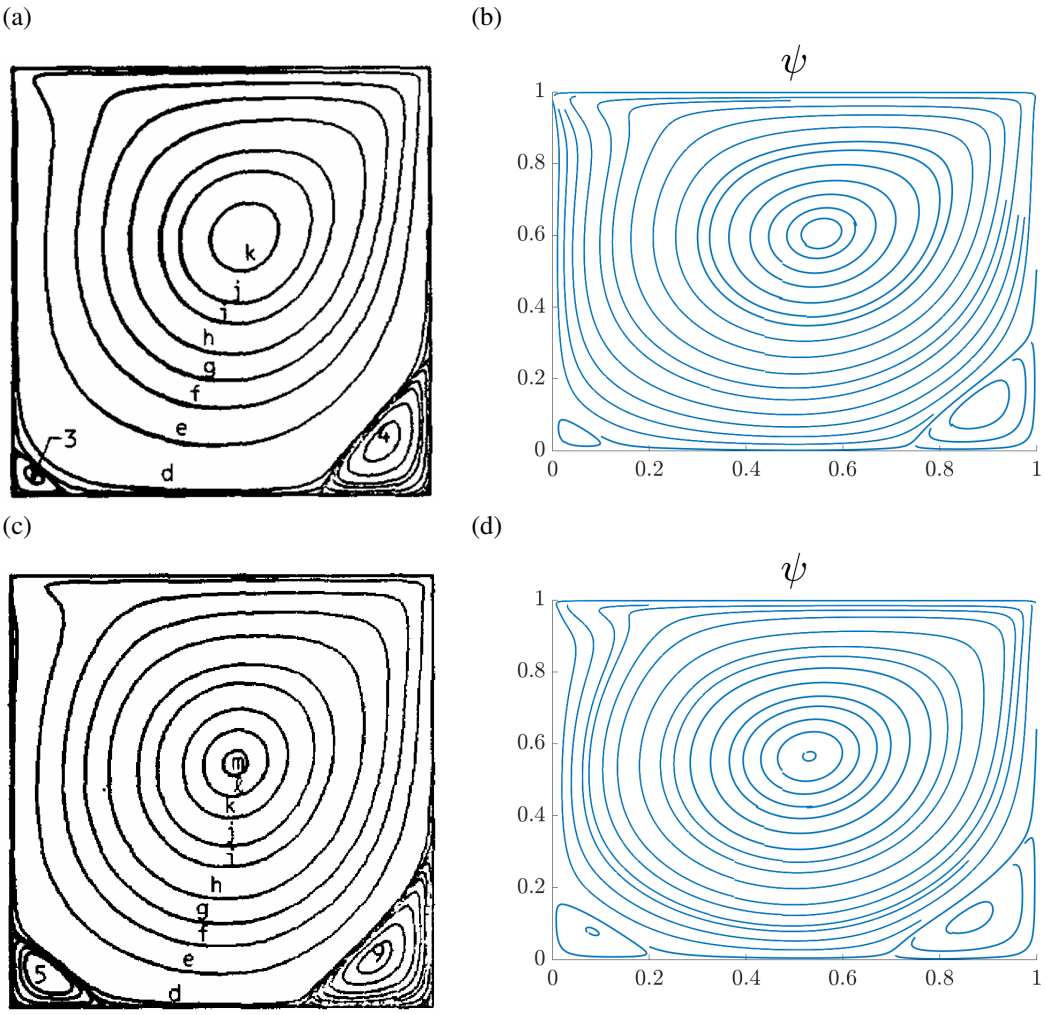

Figure 14: Comparison of streamlines **(a)** Ghia $et\ al.$'s results at $Re = 400$, **(b)** Our results at $Re = 400$, **(c)** Ghia $et\ al.$'s results at $Re = 1000$, **(d)** Our results at $Re = 1000$.

[2] J.Blair Perot. An analysis of the fractional step method. *Journal of Computational Physics*, 108(1):51–58, 1993.

[3] U Ghia, K.N Ghia, and C.T Shin. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of Computational Physics*, 48(3):387–411, 1982.

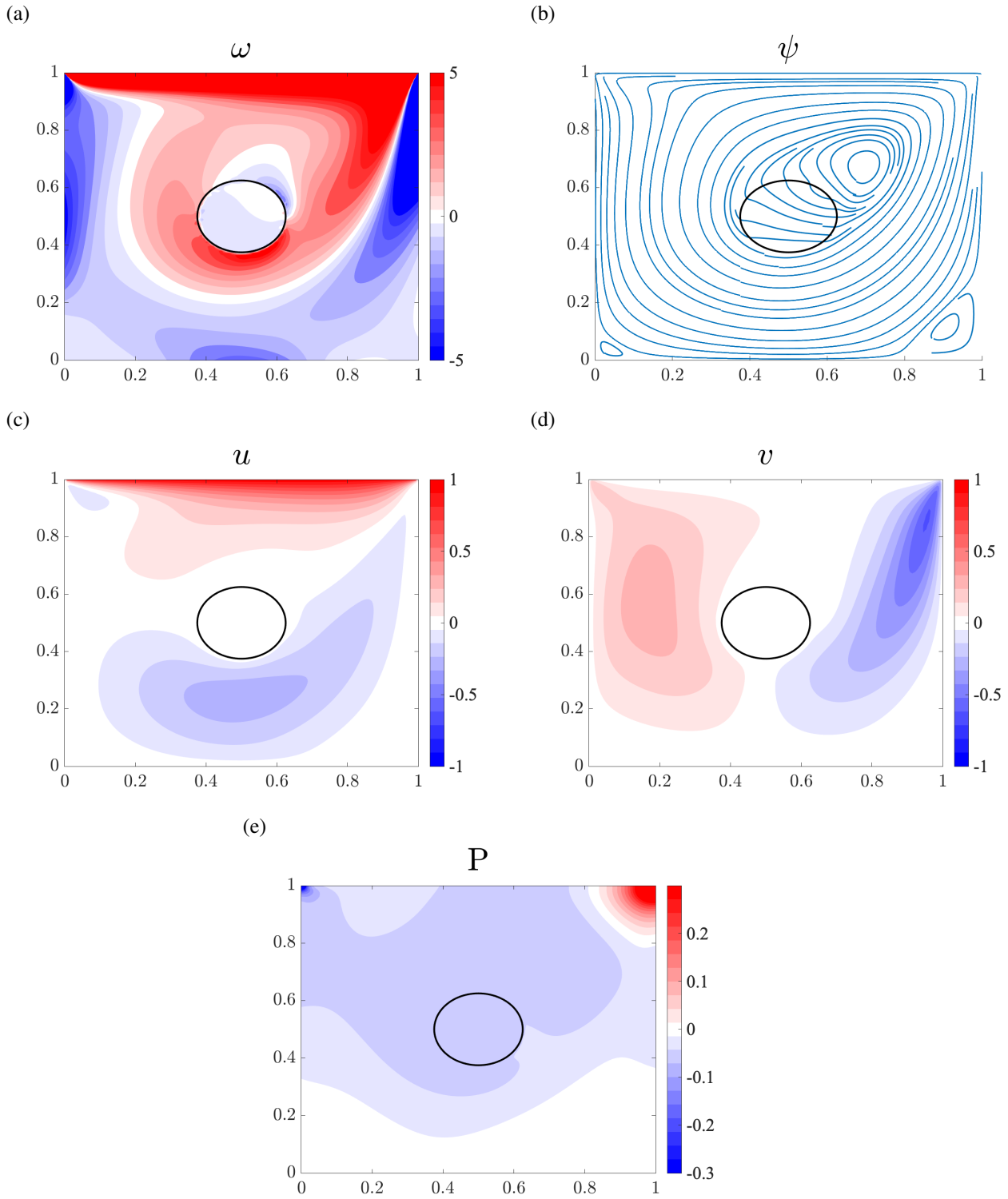

Figure 15: Immersed Boundary simulation results at CFL $= 0.5$ and $Re = 400$ with grid size 129×129. **(a)** vorticity, **(b)** streamlines, **(c)** u-velocity, **(d)** v-velocity, **(e)** pressure.

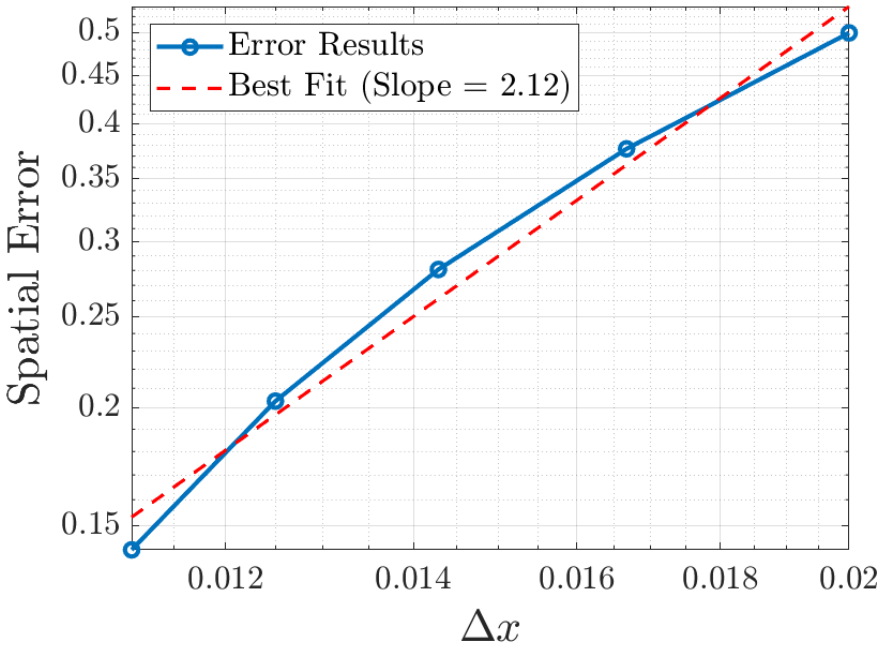

Figure 16: Spatial convergence of the immersed boundary incompressible solver.