2D LID-DRIVEN CAVITY FLOW

Venugopal Ranganathan

University of Texas at Austin
venu22@utexas.edu
December 12, 2023

ABSTRACT

This project report is submitted towards fulfilling the requirements of the final project assessment ofASE 382Q Foundations of Computational Fluid Dynamics offered during Fall 2023. Two-dimensional lid-driven cavity flow is simulated using fractional stepping using Python.

1 Introduction

We implement a solver to compute the steady solution to the incompressible Navier-Stokes equations inside a closed square cavity with the flow driven by the top lid.

The geometry of the domain is given by:

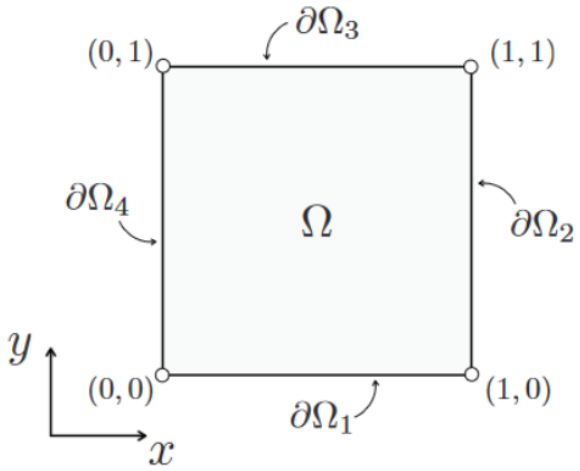

Figure 1: 2D square geometry with boundary labels, taken from [1]

As per this defined geometry, the governing equations are given by:

$$\nabla.\boldsymbol{u} = 0 \tag{1}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}.\nabla\boldsymbol{u} = -\nabla\pi + \frac{1}{Re}\nabla^2\boldsymbol{u} \tag{2}$$

where $\boldsymbol{u}$ = (u,v) is the velocity vector with x components u and v and π is the hydrodynamic pressure. The following boundary conditions are imposed:

$$\boldsymbol{u} = (0,0) \text{ on } \partial\Omega_1, \partial\Omega_2, \partial\Omega_4 \tag{3}$$

$$\boldsymbol{u} = (1, 0) \text{ on } \partial\Omega_3 \tag{4}$$

The unknowns to be solved for are the velocity (2 components - *u* and *v*) and the hydrodynamic pressure π.

2 Literature Survey

The methodology adopted is the fractional-step method as given by Kim and Moin [1985] and is given in detail in Section 17.8 of [1]. It is replicated here for reference:

The momentum equation (2) in semi-discrete form is given by

$$\frac{\partial u}{\partial t} = -G(\pi) + N(u) + \frac{1}{Re}L(u) \tag{5}$$

where the three operators are defined as follows:
The gradient operator:

$$G(\pi) = \nabla\pi \tag{6}$$

The convective (nonlinear) operator:

$$N(u) = -u \cdot \nabla u \tag{7}$$

The viscous (linear) operator:

$$L(u) = \nabla^2 u \tag{8}$$

We will also be using a Divergence operator which is defined for a vector q as

$$D(q) = \nabla.q \tag{9}$$

To implement our 2D solver, we implement 3 steps:

1. The predictor step, where momentum is advanced over an interval of size k from t_n to $t_{n+1} = t_n + k$ without the pressure term $-\nabla\pi$. We use a *semi-implicit additive Runge-Kutta method*:

$$u^* - u^n = \frac{k}{2}[3N(u^n) - N(u^{n-1})] + \frac{k}{2}\frac{1}{Re}L(u^* + u^n) \tag{10}$$

Using the linearity of the viscous operator, (10) can be rewritten as

$$u^* - \frac{k}{2}\frac{1}{Re}L(u^*) = \frac{k}{2}[3N(u^n) - N(u^{n-1})] + u^n + \frac{k}{2}\frac{1}{Re}L(u^n) \tag{11}$$

2. The projection (or correction) step, where u^* is corrected to obtain u^{n+1}

$$u^{n+1} - u^* = -kG(\phi^{n+1}) \tag{12}$$

3. We require u^{n+1} to be solenoidal i.e.,

$$D(u^{n+1}) = 0 \tag{13}$$

Equations (12) and (13) can be combined to get

$$kD(G(\phi^{n+1})) = D(u^*) \tag{14}$$

which is a Poisson problem that is solved for ϕ^{n+1}.

Section 17.5.1 from [1] can be referred for explanation of ϕ and it can be shown that $G(\pi^{n+1})$ an $G(\phi^{n+1})$ differ by a term $\mathcal{O}(\frac{k}{Re})$.

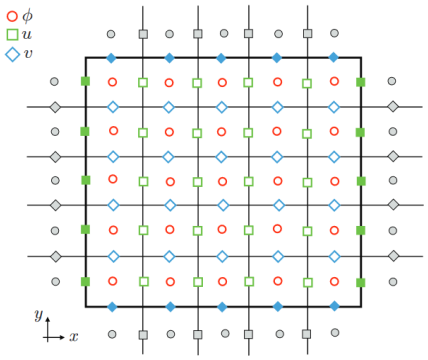

Figure 2: Arrangement of variables in in the computational domain, taken from [1]

3 Implementation of a 2D solver

The solver implementation details are given in section 17.6 of [1] and are given as follows:

1. We apply the scheme as given in section 2 on a square domain of length 1.We use a staggered mesh approach as shown in figure 2.

For this, we define two meshes -

Primary mesh:

$$x_i = (i-1)h, y_j = (j-1)h \tag{15}$$

with $i = j = 0, ...N$ and $h = 1/N$. This primary mesh spans the entire computational domain, so that $(x, y) \in [0, 1]$x$[0, 1]$. We assume we are solving over a square mesh.
Secondary mesh:

$$\hat{x}_i = \frac{x_i + x_{i+1}}{2}, \quad \hat{y}_j = \frac{y_j + y_{j+1}}{2} \tag{16}$$

where $i = j = 1, ..N$

As discussed at the section 1, there are 3 unknowns - the two components of velocity and pressure.

In the staggered mesh arrangement, these solution variables are denoted by
1. $\phi = \phi(\hat{x}_i, \hat{y}_j), (i = j = 1, .., N)$
2. $u_{i,j} = u(x_i, \hat{y}_j), (i = 1, ...N, j = 1, ..., N + 1)$
3. $v_{i,j} = v(\hat{x}_i, y_j), (i = 1, ...N + 1, j = 1, ..., N)$

Doing the discretization for the variables and operators we defined in the 3 steps in section 2,

1. We define velocities at the pressure grid as:

$$\hat{u}_{i,j} = \frac{u_{i,j} + u_{i+1,j}}{2} \tag{17}$$

$$\hat{v}_{i,j} = \frac{v_{i,j} + u_{i,j+1}}{2} \tag{18}$$

2. Discrete approximation to the convective operators used in the predictor step for the x and y-component of velocity:

$$2hN^u_{i,j}(u, \hat{v}) = -\frac{u_{i,j}(u_{i+1,j} - u_{i-1,j})}{2} - \frac{\hat{v}_{i,j} + \hat{v}_{i-1,j}}{2}(u_{i,j+1} - u_{i,j-1}). \tag{19}$$

$$2hN^v_{i,j}(\hat{u}, v) = -\frac{\hat{u}_{i,j} + \hat{u}_{i,j-1}}{2}(v_{i+1,j} - v_{i-1,j}) - v_{i,j}\frac{v_{i,j+1} - v_{i,j-1}}{2}. \tag{20}$$

3. Discrete approximation to the viscous operators used in the predictor step for the x and y-component of velocity:

$$h^2 L^u_{i,j} = u_{i+1,j} + u_{i,j+1} - 4u_{i,j} + u_{i-1,j} + u_{i,j-1}. \tag{21}$$

$$h^2 L^v_{i,j} = v_{i+1,j} + v_{i,j+1} - 4v_{i,j} + v_{i-1,j} + v_{i,j-1}. \tag{22}$$

4. Using the above discretizations from 1. - 3. we get the following discretization for the predictor step

$$u^* - \frac{k}{2}\frac{1}{Re}L^u(u^*) = \frac{k}{2}[3N^u(u^n, \hat{v}^n) - N^u(u^{n-1}, \hat{v}^{n-1})] + u^n + \frac{k}{2}\frac{1}{Re}L^u(u^n). \tag{23}$$

$$v^* - \frac{k}{2}\frac{1}{Re}L^v(v^*) = \frac{k}{2}[3N^v(\hat{u}^n, v^n) - N^v(\hat{u}^{n-1}, v^{n-1})] + v^n + \frac{k}{2}\frac{1}{Re}L^v(v^n). \tag{24}$$

(23) and (24) are used to solve for u^* and v^* as shown in Figure 3 and 4.

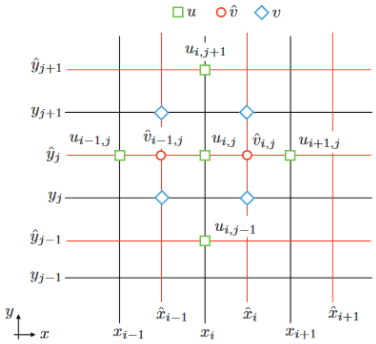

Figure 3: Stencil for u^*, taken from [1]

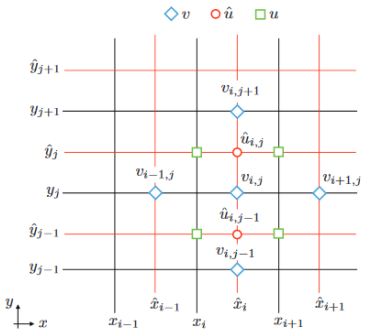

Figure 4: Stencil for v^*, taken from [1]

5. The discrete divergence operator is used to calculate the divergence at $(\hat{x}_i, \hat{y}_j)$ as

$$hD^{uv}{}_{i,j}(u, v) = u_{i+1,j} - u_{i,j} + v_{i,j+1} - v_{i,j} \tag{25}$$

6. The Poisson equation for ϕ is written as

$$kL^{\phi}(\phi^{n+1}) = D^{uv}(u^*, v^*) \tag{26}$$

using the stencil arrangement for ϕ as shown in figure 5. This equation is used to solve for ϕ^{n+1}

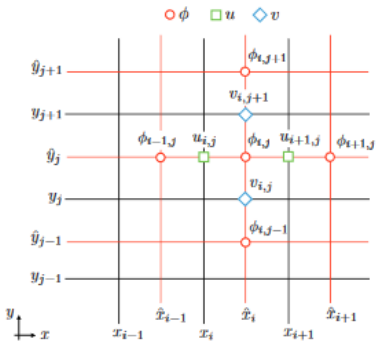

Figure 5: Arrangement of variables in in the computational domain, taken from [1]

7. Once u^*, v^*, and ϕ^{n+1} have been computed, they can be used to update the x and y component of velocities as

$$u^{n+1} = u^* - kG^x(\phi^{n+1}) \tag{27}$$

$$v^{n+1} = v^* - kG^y(\phi^{n+1}) \tag{28}$$

4 Boundary conditions and ghost nodes

For our specific case of lid-driven flow, we set the following boundary conditions:

Top: u = 1, v = 0
Bottom: u = 0, v = 0
Left: u = 0, v = 0
Right: u = 0, v = 0

The boundary conditions for u on the left and right boundary are on account of no-penetration and on the bottom is to maintain no-slip. Similarly for v.

To use the Laplacian operators in the viscous term (see equation 21, 22), a 5 point stencil is required. In order to use this discretization near the boundary nodes, ghost nodes are used.

The introduction of ghost nodes means we have to incorporate additional equations into our system for these ghost nodes. Thee conditions imposed for u^* and v^* for the predictor step are given for a sample $2x2$ grid in figure 6 and 7

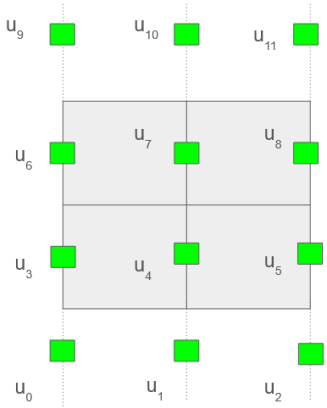

Figure 6: Sample stencil for u^* for 2 x 2 case

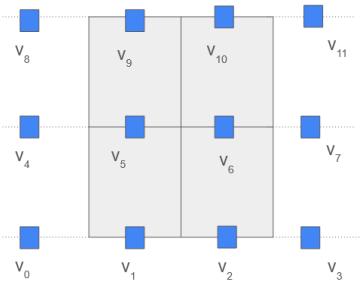

Figure 7: Sample stencil for v^* for 2 x 2 case

For u^*, the boundary conditions for the case in figure 6 are:
$u_0^* = -u_3^*, \quad u_1^* = -u_4^*, \quad u_2^* = -u_5^*$

$u_3^* = 0, \quad u_5^* = 0, \quad u_6^* = 0, \quad u_8^* = 0$

$u_9^* + u_6^* = 2, \quad u_{10}^* + u_7^* = 2, \quad u_{11}^* + u_8^* = 2$

The rhs is populated as given by (23) for u_4^* and u_7^*

Similarly for v^* the boundary conditions for the case in figure 7 are:
$v_0^* = -v_1^*, \quad v_3^* = -v_2^*, \quad v_4^* = -v_5^*, \quad v_7^* = -v_6^*, \quad v_8^* = -v_9^*, \quad v_{11}^* = -v_{10}^*$

$v_1^* = 0, v_2^* = 0, v_9^* = 0, v_{10}^* = 0$

The rhs is populated as given by (24) for v_5^* and v_6^*

As for velocity, the Laplacian is also required for pressure (ϕ) in equation (26). Hence, we introduce ghost nodes for ϕ as well. The formulation is shown for a sample 2 x 2 plot in figure 8.

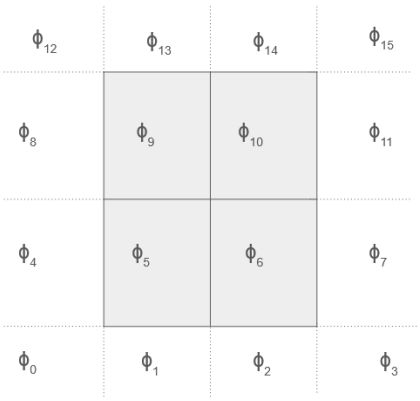

Figure 8: Sample stencil for ϕ for 2 x 2 case

$\phi_0^* = 0, \phi_3^* = 0, \phi_{12}^* = 0, \phi_{15}^* = 0,$

$\phi_1^* = \phi_5^*, \quad \phi_2^* = \phi_6^*, \quad \phi_4^* = \phi_5^*, \quad \phi_7^* = \phi_6^*, \quad \phi_8^* = \phi_9^*, \quad \phi_{11}^* = \phi_{10}^*, \quad \phi_{13}^* = \phi_9^*, \quad \phi_{14}^* = \phi_{10}^*,$

The Laplacian is populated using the stencil [1 1 -4 1 1] for the points $\phi_5, \phi_6, \phi_9, \phi_{10}$

The details of the prescribed equations and why they are chosen to be the value they are for the ghost nodes is given in more detail in section (17.7.1) in [1].

5 Code structure

Our code is structured into 2 files - main.py and utils.py. main contains initialization of the varibles and the loop while all the functions are declared and defined in the utils.py function.
For our code we use the following functions to compute our pressure and velocity values:

```
create_laplacian_for_pressure(n,h,k)
Duv_maker(u,v,n,h)
G_update_u_v(p,h)
N_u_maker(u,v,n,h)
N_v_maker(u,v,n,h)
u_star_solver_lhs_operator(n,k,Re,h)
u_star_solver_rhs(n,k,Re,h,u_n,v_n,u_n_1,v_n_1)
v_star_solver_lhs_operator(n,k,Re,h)
v_star_solver_rhs(n,k,Re,h,u_n,v_n,u_n_1,v_n_1)
```

Listing 1: Functions used

In main.py we first initialize the variables that we will be using

```
n = 49 #sqaure mesh, no of points in pressure grid along each dimension
nx,ny = n,n #keeping it square for now. [0,1] x [0,1]

dx = 1/(nx-1)

#Specify time-step size and number of time iterations, vary to achieve convergence
    .
# dt = T/(nt-1) #time

dt = 0.01
T =  100
nt = 1 + int(T/(dt))

Re = 5000 # Reynolds Number
```

```
tol = 1e-3
# tolerance for early termination, i.e. if difference x-comp
# of velocity from one t step to another is less than tol

phi = np.zeros(int( (n+2)**2),dtype=np.float64)
u_n = np.zeros(((ny+2)*(nx+1)),dtype=np.float64)
v_n = np.zeros(((ny+1)*(nx+2)),dtype=np.float64)

G_x = np.zeros(len(u_n),dtype=np.float64)
G_y = np.zeros(len(v_n),dtype=np.float64)

N_u_n = np.zeros(len(u_n),dtype=np.float64)
N_v_n = np.zeros(len(v_n),dtype=np.float64)

#to start the loop
u_n_1 = u_n
v_n_1 = v_n

#operators that need to be made only once
lhs_u_star = u_star_solver_lhs_operator(n,dt,Re,dx)
lhs_v_star = v_star_solver_lhs_operator(n,dt,Re,dx)
laplacian_pressure = create_laplacian_for_pressure(n,dx,dt)
```

Listing 2: Initialization of variables

Our main loop looks as follows. A significant amount of time is spent in the spsolve commands from scipy. This could be optimized in future by using callbacks in Conjuagte Gradient as the null vector for the rank deficient system of pressure can be computed.

```
for t in range(nt):

    u_star_rhs = u_star_solver_rhs(n, dt, Re, dx, u_n, v_n, u_n_1, v_n_1)
    v_star_rhs = v_star_solver_rhs(n, dt, Re, dx, u_n, v_n, u_n_1, v_n_1)

    u_star = scipy.sparse.linalg.spsolve(lhs_u_star,u_star_rhs)
    v_star = scipy.sparse.linalg.spsolve(lhs_v_star,v_star_rhs)

    Duv = Duv_maker(u_star,v_star,n,dx)

    phi = scipy.sparse.linalg.spsolve(laplacian_pressure,Duv)

    Gx,Gy = G_update_u_v(phi,dx)

    u_n_1,v_n_1 = u_n,v_n

    u_n,v_n = u_star - dt*Gx, v_star - dt*Gy
```

Listing 3: Main loop of code

Finally, we print out the value of the norm of the Divergence matrix and make the quiver and streamplots.

```
u_n,v_n = u_n.reshape(ny+2,nx+1),v_n.reshape(nx+1,ny+2)

exp_val = (u_n[1:-1,1:] - u_n[1:-1,0:-1])/dx + (v_n[1:,1:-1] - v_n[0:-1,1:-1])/dx
print(np.linalg.norm(exp_val))

x = np.linspace(0,1,nx+1)
y = np.linspace(0,1,ny+1)
xcc = (x[0:-1] + x[1:])/2
ycc = (y[0:-1] + y[1:])/2
X,Y = np.meshgrid(xcc,ycc)

phi_pl1 = np.reshape(phi,(ny+2,nx+2))
phi_pl1 = phi_pl1[1:-1,1:-1]

ucc = (u_n[1:-1,0:-1] + u_n[1:-1,1:])/2
```

```
vcc = (v_n[0:-1,1:-1] + v_n[1:,1:-1])/2



plt.figure()
plt.contourf(X, Y, phi_pl1)
plt.colorbar()
plt.quiver(X, Y, ucc, vcc, color="black")
plt.streamplot(X, Y, ucc, vcc, color="black")
plt.title(f"N = {n+1} points along each dimension and Re = {Re}")
plt.xlim(0,1)
plt.ylim(0,1)
savepath = f"/home/venu/cfd_proj_f23/draft_3/plots/Streamplots/{n+1}_{Re}.png"
plt.savefig(savepath)
plt.show()
```

Listing 4: Main loop of code

6 Numerical Results

Quiver plots and streamplots for velocity plotted on contour plots of pressure are as shown. For all cases time-step size is set to 0.5 seconds and the solver is run for 100 seconds.

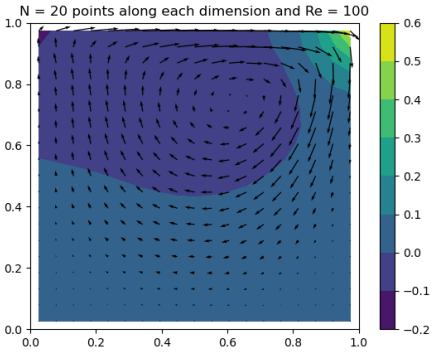

Figure 9: Quiver plot for Re = 100

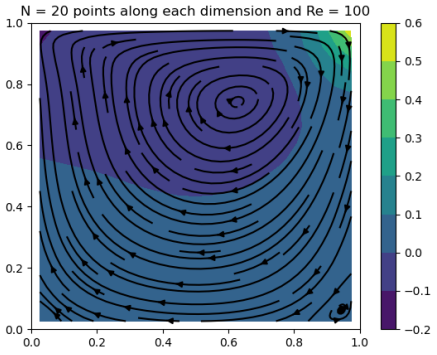

Figure 10: Stream plot for Re = 100

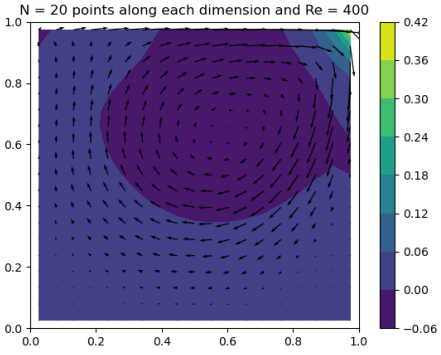

Figure 11: Quiver plot for Re = 400

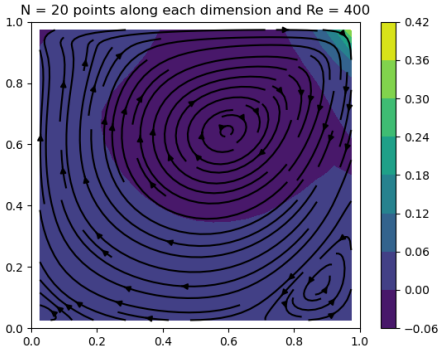

Figure 12: Stream plot for Re = 400

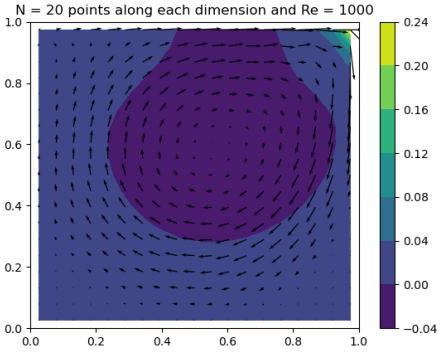

Figure 13: Quiver plot for Re = 1000

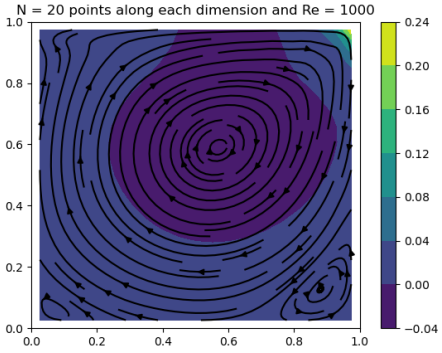

Figure 14: Stream plot for Re = 1000

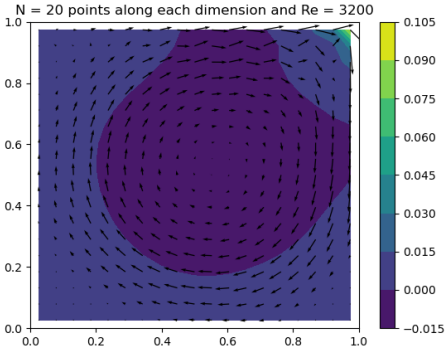

Figure 15: Quiver plot for Re = 3200

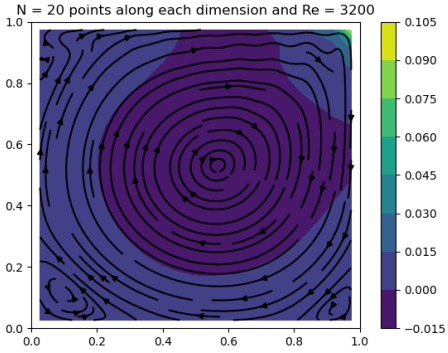

Figure 16: Stream plot for Re = 3200

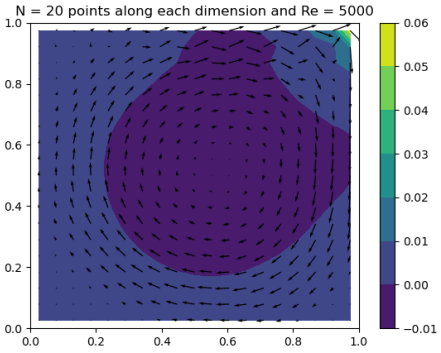

Figure 17: Quiver plot for Re = 5000

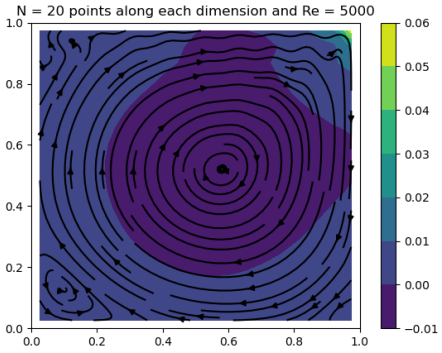

Figure 18: Stream plot for Re = 5000

7 Validation

1. We formulate our Laplacian that we use in computing our ϕ values (pressure). The order of convergence is compared by manufacturing a known vector: $T(x, y) = sin(2\pi x)cos(\pi y)$. The expected rhs when this vector is multiplied by the Laplacian is $\nabla^2 T(x, y) = -5\pi^2 sin(2\pi x)cos(\pi y)$ We compare the error of our numerical solution against this analytical solution.

The expected order of convergence of 2 is obtained. Code for this particular validation is given in Github in the file validate-laplacian.py

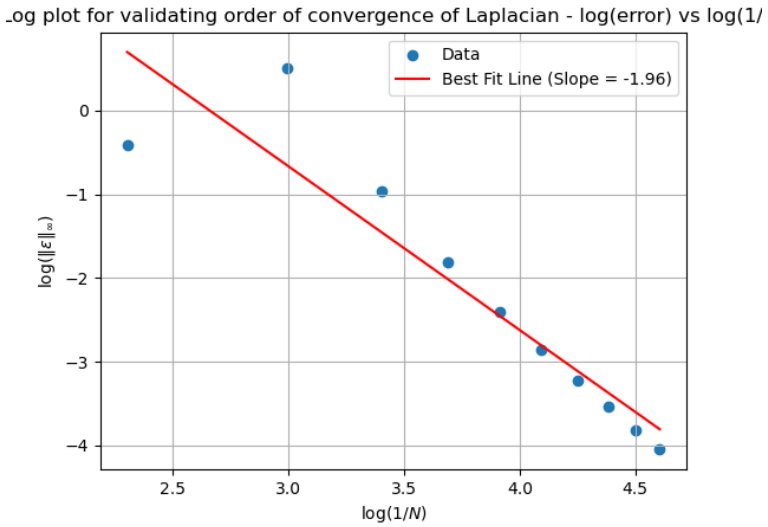

Figure 19: 2nd order convergence of laplacian seen for a manufactured solution.

2.For the above 6 cases tested, we calculate the value of the norm of the divergence of the velocity values at each grid point. Since, analytically, we strive for a solenoidal velocity, this value should be zero.

This is obtained in the code as follows:

```
exp_val = (u_n[1:-1,1:] - u_n[1:-1,0:-1])/dx + (v_n[1:,1:-1] - v_n[0:-1,1:-1])/dx
print(np.linalg.norm(exp_val))
```

Listing 5: Calculating norm of the divergence

The values are as follows:

Divergence values	
Reynolds Number (n = 20, dt = 0.05, T = 100s)	$\|\|D(u,v)\|\|_2$
100	7.915258205778096e-15
400	6.08499149540492e-15
1000	3.7167380572527076e-15
3200	3.12331795211701e-15
5000	3.0445246437689395e-15

8 Code

All code associated with the project is up on Github and accessible at this link Github.

References

1. Bisetti, Introduction to Computational Fluid Dynamics, 2022
2. Link: Singular Linear System - Pressure Poisson Equation
3. Link:Numerically solving a Poisson equation with Neumann boundary conditions.