University of Groningen

Advanced Topics in Security and Privacy

Final Assignment

LID-DRIVEN CAVITY FLOW USING FINITE DIFFERENCES

Authors

Swastik Satyanarayan Nayak S4151968

November 12, 2020

Contents

1 Introduction **3**
- 1.1 Problem statement . . . 4
- 1.2 Computational methods and/or strategies . . . 6
  - 1.2.1 Central differential approximation . . . 6
  - 1.2.2 Forward differential approximation . . . 6
  - 1.2.3 Strategy to check stability . . . 7
  - 1.2.4 Error tolerance and Max iterations . . . 7
- 1.3 Reference models . . . 8

2 Implementation **12**
- 2.1 Framework . . . 12
  - 2.1.1 Adding a new scenario . . . 12
  - 2.1.2 Automated plotting and controls . . . 12
  - 2.1.3 Automated storing and loading with compression . . . 13
  - 2.1.4 Lid Cavity class structure . . . 13
- 2.2 Stream function . . . 13
  - 2.2.1 Pseudo-code . . . 14
- 2.3 Vorticity . . . 14
  - 2.3.1 Boundary conditions . . . 15
  - 2.3.2 Pseudo-code . . . 15
- 2.4 Momentum equations . . . 16
  - 2.4.1 Pseudo-code . . . 16
- 2.5 Pressure . . . 17
  - 2.5.1 Pseudo-code . . . 17
- 2.6 Limitations . . . 17

3 Result and Discussion **18**
- 3.1 Moving top wall Re 10, N = 100, dt = 0.0001 . . . 19
- 3.2 Moving top wall Re 100, N = 200, dt = 0.00001 . . . 20
- 3.3 Moving top and bottom wall Re 10, N = 100, dt = 0.0001 . . . 21
- 3.4 Moving top and bottom wall Re 100, N = 200, dt = 0.00001 . . . 23
- 3.5 Moving Top and Bottom wall, same directional spin . . . 25
- 3.6 Comparing the vortex centers against reference . . . 28
- 3.7 Moving wall Top and Right, same directional spin . . . 31
- 3.8 Influence of Relaxation factor (Beta) . . . 33
- 3.9 Dual convergence scenario . . . 35
- 3.10 Complete divergence scenarios . . . 37
- 3.11 Complete convergence scenario . . . 40
- 3.12 Influence of Re on momentum . . . 41

4 Conclusion **41**

A Code snippets **44**
- A.1 Lid cavity class structure . . . 44
- A.2 Stream function . . . 45
- A.3 Vorticity function . . . 46

A.4 Momentum function 46
A.5 Pressure function 47
A.6 Smart plot function 47
A.7 Captured Metrics 50
A.8 GitHub 50

1 Introduction

In the world of computational fluid dynamics the lid driven cavity flow is a popular bench-marking case study for algorithms that describe the flow of an in-compressible viscous fluid. In layman's terms, a square cavity is chosen such that its walls can either allow the enter the cavity or completely block the inward flow. The simulation of the liquid flow is observed and the credibility of the algorithm is gauged.

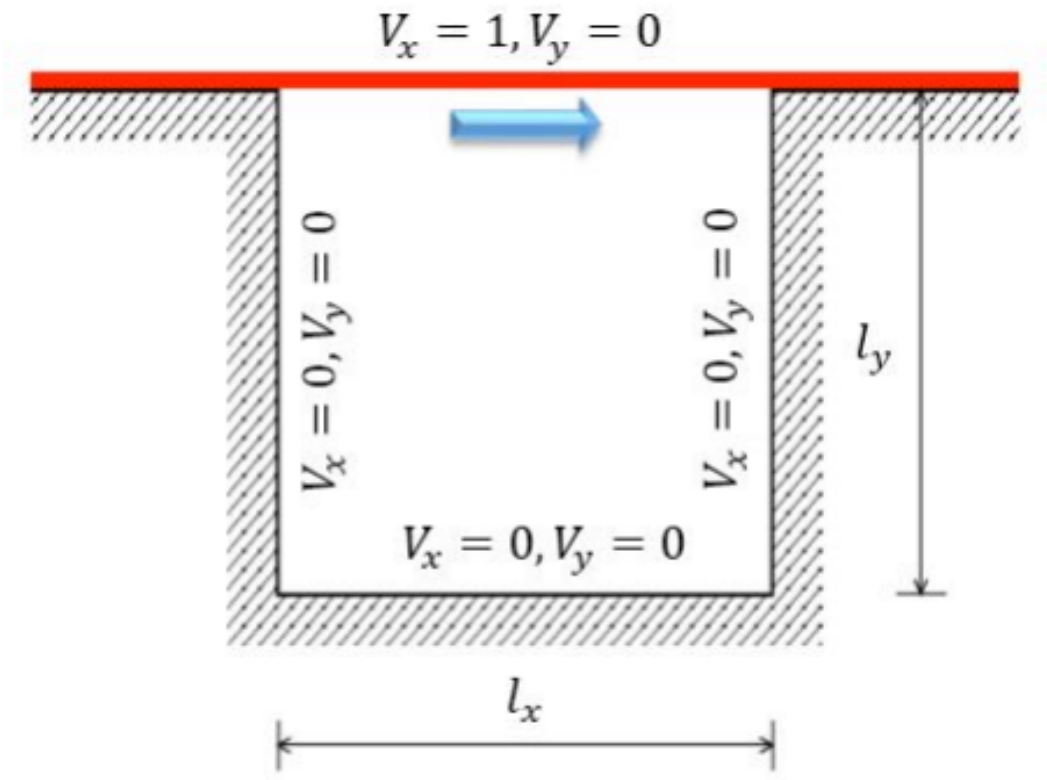

Figure 1: Illustrates the Lid driven cavity flow problem.

From figure 1, we can observe that the top wall is marked in **red** whereas the other walls are shaded in gray. This indicates that the top wall can be moved and the rest of the walls are rigid bodies that cannot be altered. A viscous fluid is then let inside the cavity by slowing moving the top wall in the horizontal direction with a velocity of of V_x. It is also assumed that the fluid flow is incompressible and the rigid walls exhibit the no-slip condition. The V_x and V_y represent the velocity of the fluid in horizontal (x-axis) and vertical (y-axis) directions, it will also be termed as the u-momentum and v-momentum. The l_x and l_y indicate the width and length of the cavity.

- **Incompressible flow**: It refers to a flow in which the fluid's material density is constant with respect to time and space. There are no real fluid that exhibit a total lack of compressibility in real world, and this assumption is enforced for mathematical simplicity [11].
- **No-slip condition**: The assumption states that viscous fluids at the solid boundaries have zero velocity. The physical justification to this assumption is that a particle close to the surface do not move along with the flow when the adhesion is stronger than cohesion. It is simpler

to assume that the particles are adhered to the wall than to introduce more mathematical computation that increases the complexity of the system [12].

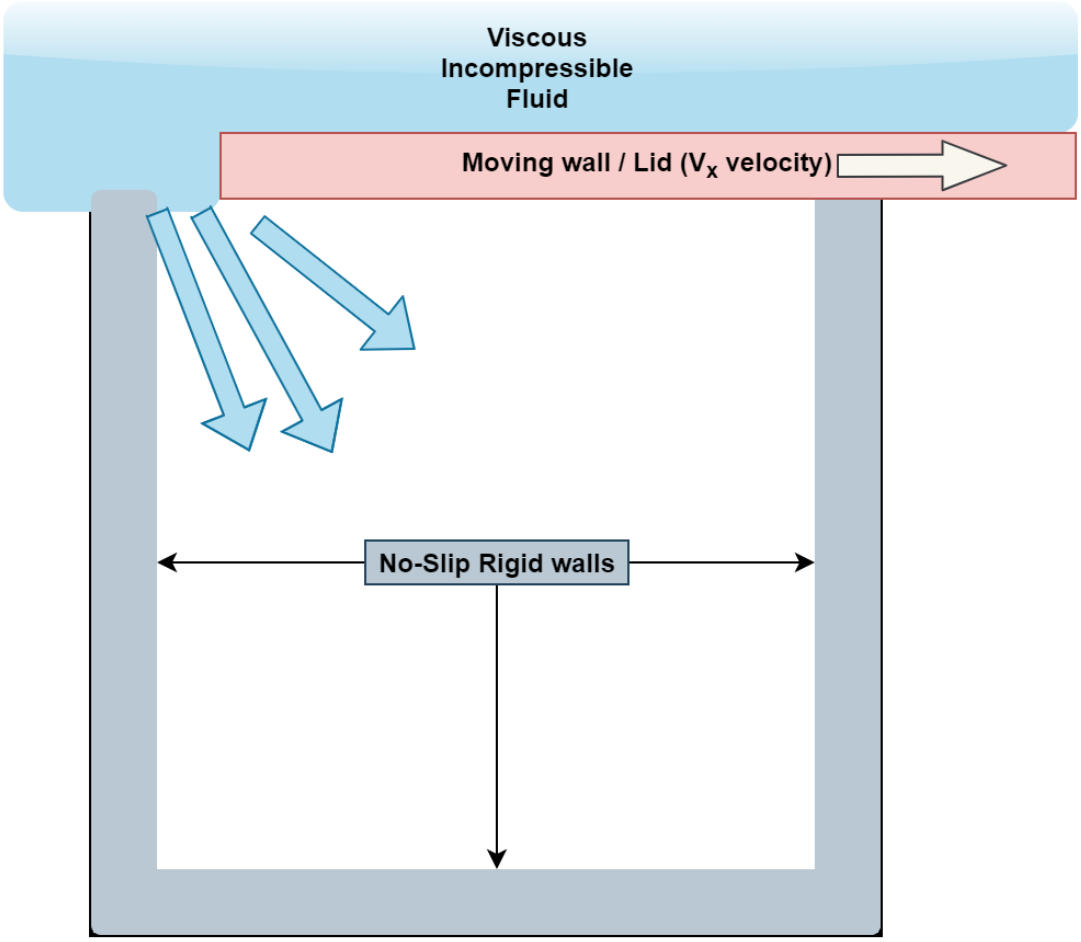

Figure 2: Illustrates the conceptual entities of the Lid driven cavity flow problem.

1.1 Problem statement

The aim of this assignment is to find the velocity and pressure profiles in this rectangular and/or square (if $l_x == l_y$) domain. To determine these quantities, we will utilize the stream function and vorticity field from the vorticity form of the Navier-strokes equations.

- **Stream Function**: The flow velocity component of a fluid can be expressed as the derivatives of the scalar stream function. Streamlines can be plotted using the stream function that can be used to represent the trajectories of particles in a steady flow [13].
- **Vorticity**: The vorticity is the pseudo-vector field that describes the local spinning motion of a continuum near some point, as would be seen by any observer located at the point and travelling along with the flow. It is used to explain the complex flow phenomena of the fluids such as formation and motion of vortex rings [16].
- **Navier-strokes equations**: Are a set of partial differential equations (PDE) that describe the motion of viscous fluid substances.

The below section describes the various Navier-Strokes equations and their discretized equivalents. The non-dimensionalized vorticity form of Navier-Stokes equations are,

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -w \tag{1}$$

$$\frac{\partial w}{\partial t} = -\frac{\partial u}{\partial y}\frac{\partial w}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2}\right) \tag{2}$$

The Stream function is denoted by **u** and the vorticity is denoted by **w**, and **Re** is the "*Reynolds number*", it describes the relative contribution of viscous effects versus inertial effects in the fluid flow.

The equations can be used to derive the stream functions, vorticity (pseudo-vector field that describes the local spinning motion of a continuum near some point), pressure distribution, horizontal and vertical velocity of the fluid.

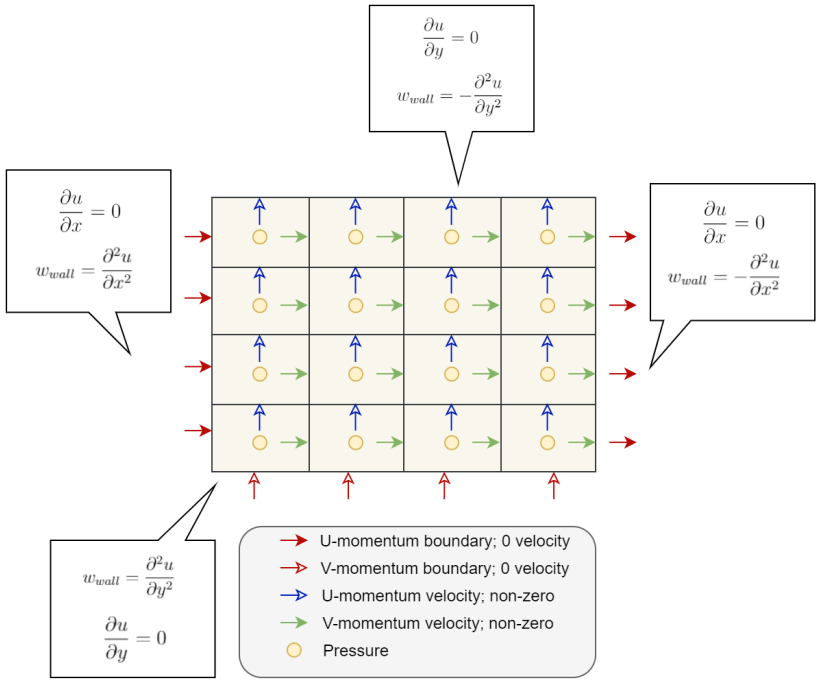

Figure 3: Illustrates the Velocity and Pressure components of the Navier-stokes equation.

The figure 3 shows how the velocities and pressure components are arranged/calculated. The u-velocity and v-velocities are independent of each other and act on their respective x and y planes. The pressure resides at the center of the box, as it is primarily an average of its corner point values. The red arrows indicate that the velocity around the border is 0, since no-slip condition defines that the liquid around the boundaries stick to the surface and exhibit no motion. The top wall is a moving wall, and hence the velocity over it is non-zero. The figure also shows the summarized version of the boundary conditions of the vorticity component.

1.2 Computational methods and/or strategies

1.2.1 Central differential approximation

For a given function $f(x)$ if the finite difference will take $f(x+h)$ and $f(x-h)$ values in its slope to compute the $f^{'(x)}$ then the numeric operation is a central differential approximation (CDA). Figure 4 shows the concept of CDA where the $f'(4)$ is computed using its neighboring values. One key observation here is that the slope tangential to $f(4)$ is parallel to the slope drawn from $f(2)$ and $f(5)$ indicating that they are reciprocals of each other. Equation 3 shows the generalize formulation of the central differential approximation.

$$f^{'}(x) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{3}$$

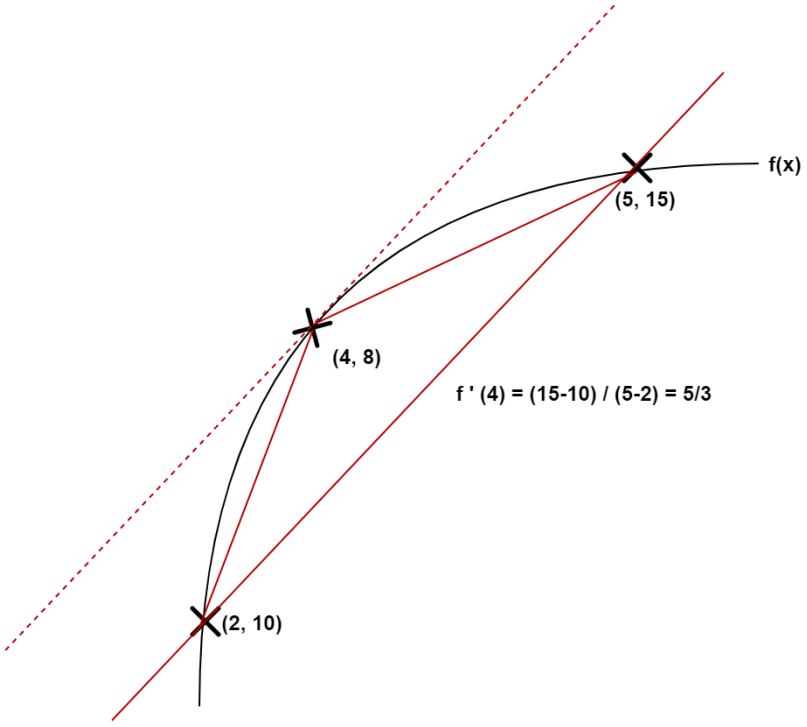

Figure 4: Illustrates the Central differential approximation.

1.2.2 Forward differential approximation

The forward differential approximation is similar to the earlier discussed central differential approximation. But in this approximation the current data value $f(x)$ and future data value $f(x+h)$ is selected to determine the $f^{'}(x)$ value. Equation 4 shows the general form of the forward differential equation and figure 5[1] shows the geometrical representation .

$$f^{'}(x) \approx \frac{f(x+h) - f(x)}{h} \tag{4}$$

[1]https://source.ggy.bris.ac.uk/wiki/NumMethodsPDEs

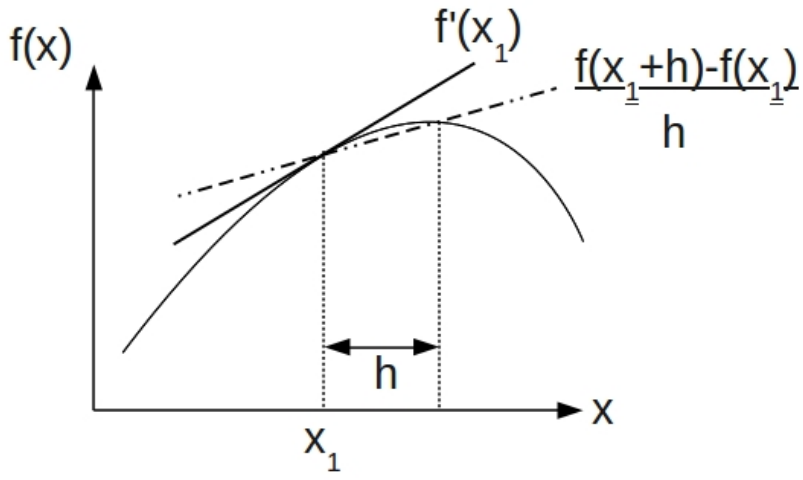

Figure 5: Illustrates the Forward differential approximation

1.2.3 Strategy to check stability

The stability of the system is calculated using the vorticity matrix. Some decisions and assumptions related to this are as follows,

- The exact center of the cavity at time t=0 and t=1 would be the same since the viscous fluid has not expressed its influence over this section of the cavity at the said time. So a decision was not made to check only the center of the cavity for stability.
- The vorticity matrix that illustrates the local spinning motion of a continuum, and if unchanged after a set number of iterations would indicate that a global stability has been achieved by the system. This is performed by subtracting the previous iteration vorticity with the current, and determining the maximum absolute error value inside the matrix.

$$max(abs(old\_vorticity - vorticity)) = Errorordeviation$$

.

The code snippet used that corresponds to this functionality is as shown below,

```
if np.max(np.max(np.abs(np.subtract(obj.w, obj.old_w)))) < obj.err_tol:
  print(np.max(np.max(np.abs(np.subtract(obj.w, obj.old_w)))));
  obj.cont = False;
```

Figure 6: Stability of the system code snippet

1.2.4 Error tolerance and Max iterations

The simulation is run with the error tolerance of $1e-5$ with an upper limit of 10000 iterations. A stable steady state is not achieved if a simulations dose not converge (i.e its total error greater than the tolerance) and has reached the maximum iteration of 10000.

Initially the simulation was executed with an error tolerance of $1e-3$, the simulation takes 50% less time to execute, but the accuracy of the simulation was not satisfactory.

The max iteration of 10000 was chosen with an idea that any simulation should terminate after it has executed approximately for about an hour without convergence.

1.3 Reference models

The section will define the expected state of the system that the will function as the ground truth. The expected scenario is taken from the research article *"FOUR-SIDED LID-DRIVEN CAVITY FLOW USING TIME SPLITTING METHOD OF ADAMS-BASHFORTH SCHEME" by C. S. N. Azwadi, A. Rajab and A. Sofianuddin* [2] and *"A detailed study of lid-driven cavity flow at moderate Reynolds numbers using Incompressible SPH"* [6].

As a preliminary validation we will compare the output generated for Reynolds number 100 against the reference as shown in the image below, the result is not expected to be a one-to-one match as the

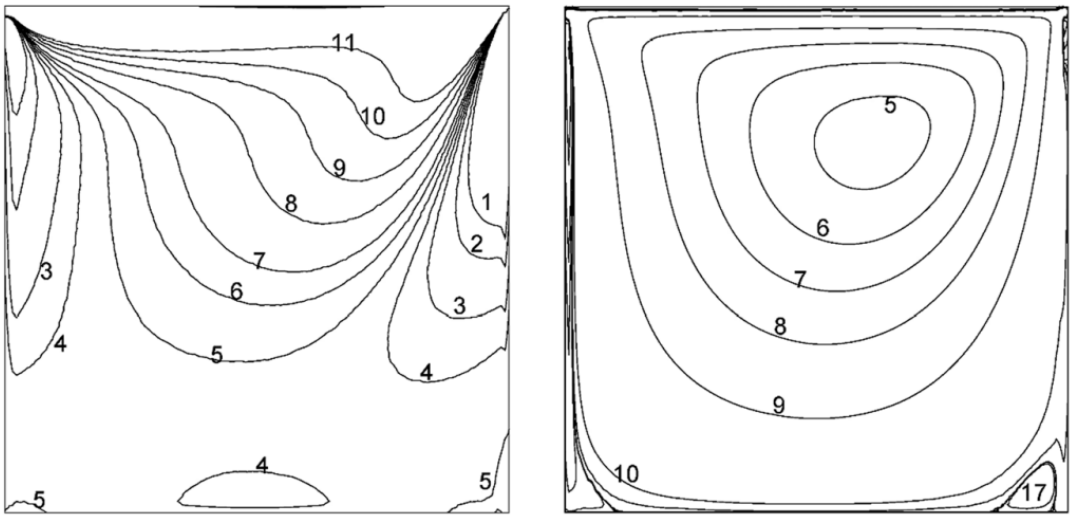

Figure 7: Vorticity and Stream functions for Re=100 [6]

This scenario from [2] was selected due to the following reasons,

- The experiment illustrates all the 4 sides of the cavity, thus allowing us to test the accuracy with respect to the 4 walls.
- The vortex centers are readily available, thus simplifying the evaluation process.
- Reynolds number 10, 100 and 127 are studied in the paper, which is inline with the Reynolds number chosen in this assignment.
- The grid size used is of a smaller scale than the one adopted in the assignment, thus we need to perform $co - ordinate/N$ to accurately match the results.
- The paper does not utilize the Navier-stokes equations, so a small variation in the vortex centers with respect to the ones generated by the assignment is expected.

Reynolds number	Left Primary Vortex (LPV)	Right Primary Vortex (RPV)	Top Primary Vortex (TPV)	Bottom Primary Vortex (BPV)
10	0.155,0.497	0.851,0.509	0.509,0.851	0.497,0.155
100	0.161,0.442	0.845,0.559	0.559,0.845	0.442,0.161
127	0.168,0.442	0.839,0.559	0.559,0.839	0.442,0.168

Figure 8: Expected vortex coordinates for Re=10, 100, and 127 [2].

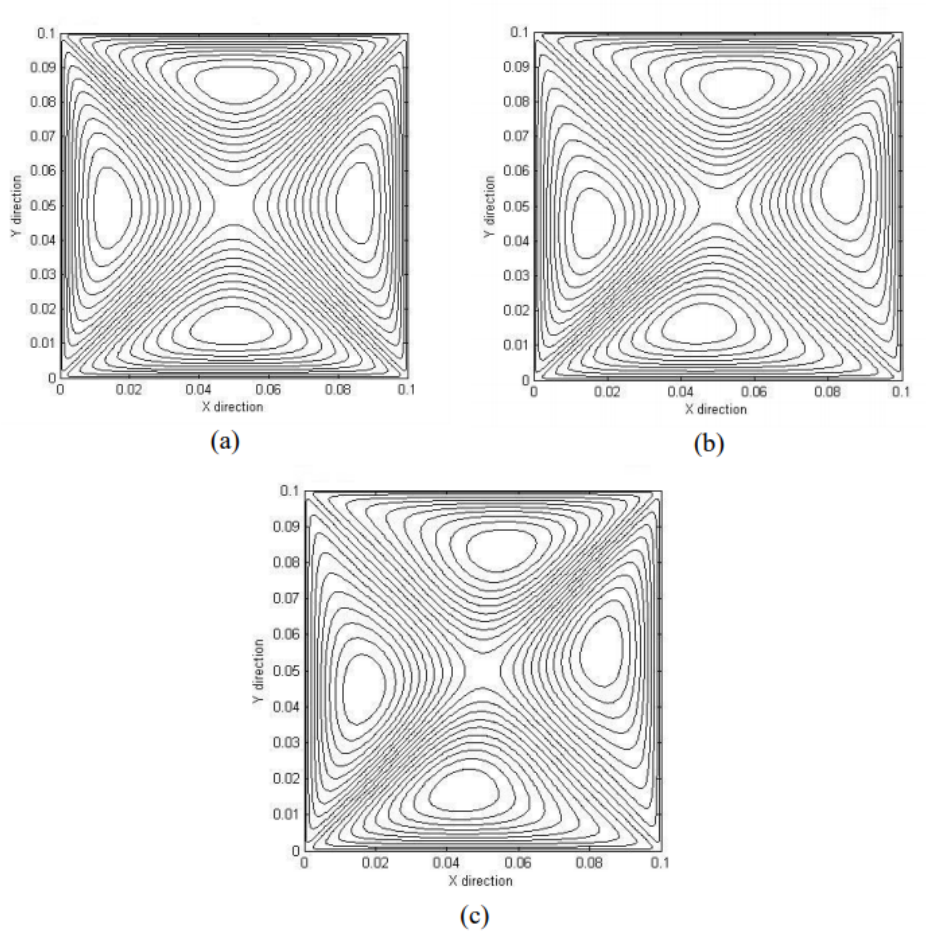

Figure 9: Streamline patterns for four-sided lid-driven cavity: (a) Re=10, (b) Re=100 and (c) Re=127 [2]

The u-momentum and v-momentum across at the domain center for the 4 vortex formation reference with respect to their perpendicular planes and/or axis is as shown in the figures below.

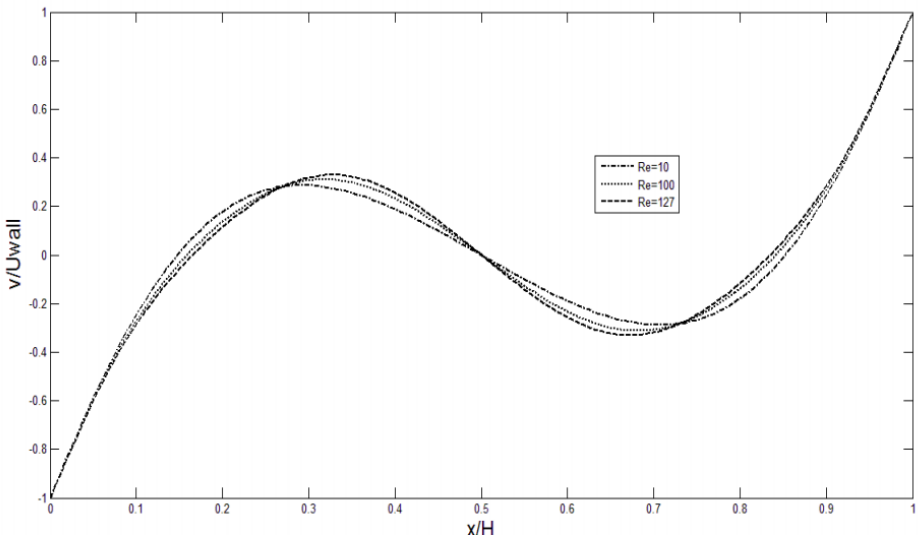

Figure 10: Graph of v-velocity profile along y = 0.5 [2]

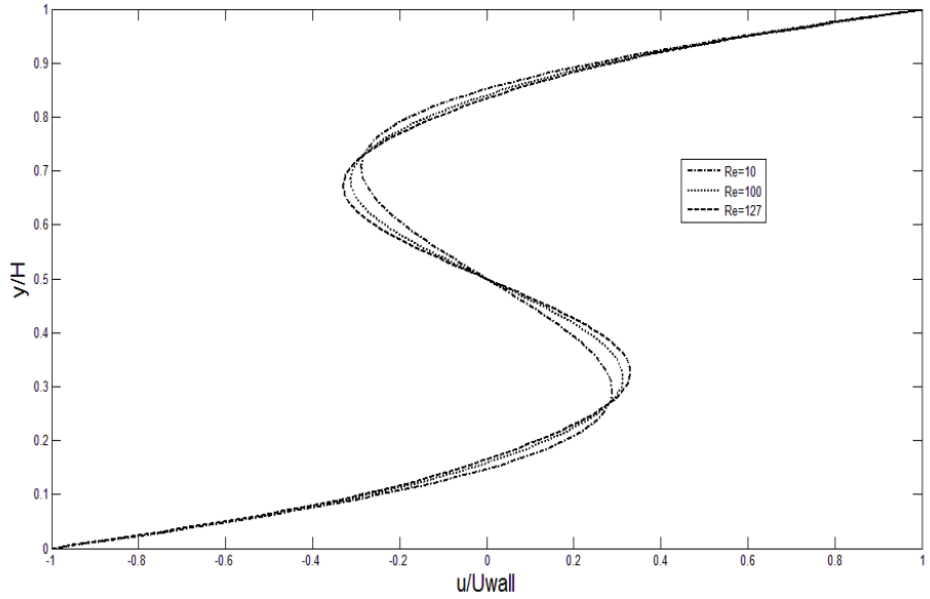

Figure 11: Graph of u-velocity profile along y = 0.5 [2]

The u-profile and the v-profile for Re = 10, and 100 is as shown in the figures below,

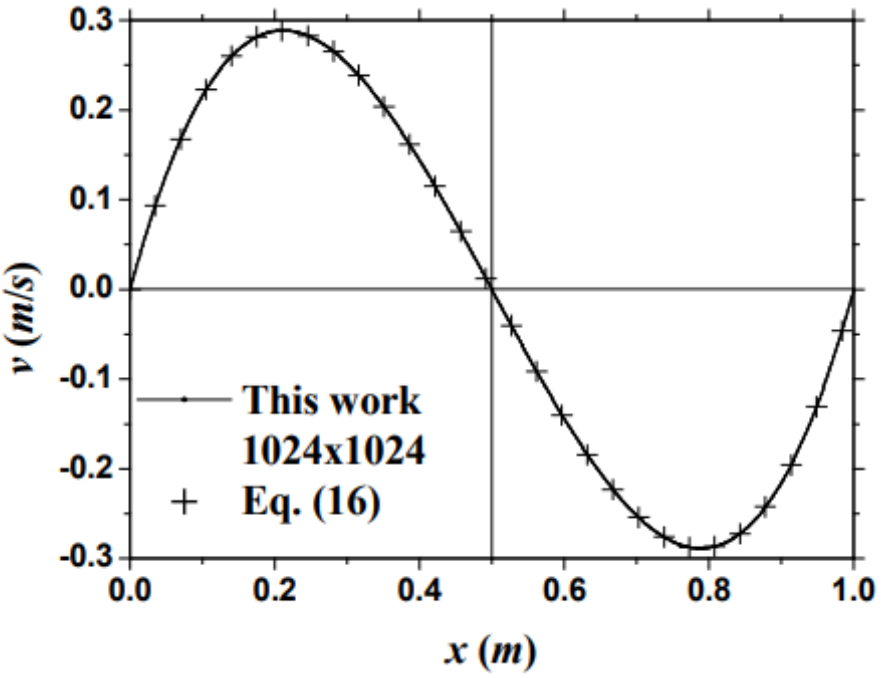

Figure 12: Graph of v-velocity profile along y = 0.5 [8]

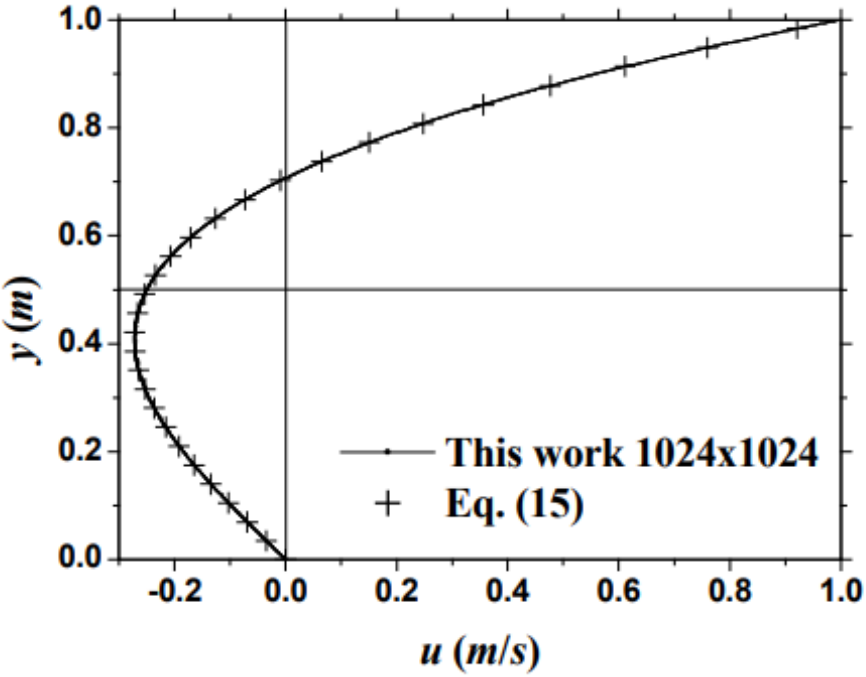

Figure 13: Graph of u-velocity profile along y = 0.5 [8]

Evaluation strategy: The top, bottom, left, and right vortex centers in figure 8 are cross-verified against the co-ordinates obtained from the assignment. The contours portrayed in the figure 9 are visually compared against our result. The momentum plots Figures 10 and 11 are also compared.

Vortex's in a fluid flow has the following properties, that the analysis would attempt to confirm

- Two or more vortices that are approximately parallel and circulating in the same direction will attract and eventually merge to form a single vortex, whose circulation will equal the sum of the circulations of the constituent vortices [15].
- On the contrary, vortices that spin in the opposite direction should not merged into a single large vortex [15].
- The pressure around the core/center of the vortex is severely less in-comparison to the pressure away from the core of the vortex [15]. In real world scenario the center of the vortex can also be void of fluids and can be air pockets that exhibit low pressure areas in certain conditions.

2 Implementation

The section will describe the strategy that was adopted to implement the lid driven cavity flow. The relevant equation, and their discretized form, along with the pseudo-code of the project is discussed in this section. The relevant code can be found in the Appendix section or on

2.1 Framework

The project designed is highly reusable. The subsection will discussion some of the appealing non-functional aspects of the framework that has been designed with the intention of code re-usability.

2.1.1 Adding a new scenario

To create and plot a new scenario the user simply needs to make an new scenario entry with the right set of parameters as shown in figure 15.

```
scenarios = [];
scenarios.append((LidCavity(0.01,1,  10, 1e-5,0.0001, 0.001, 1.5, 10000, 1, run=False, top=1, bottom=0, dir_suffix="_T"), [0, 2]));
scenarios.append((LidCavity(0.001, 0.2, 100, 1e-5,0.00001, 0.001, 1.5, 10000, 1, run=False, top=1, bottom=0, dir_suffix="_T", disp_separate=True), [1]));
scenarios.append((LidCavity(0.01,1,  10, 1e-5,0.0001, 0.001, 1.5, 10000, 1, run=False, top=1, bottom=1, dir_suffix="_TB"), [0]));
scenarios.append((LidCavity(0.001, 0.2, 100, 1e-5,0.00001, 0.001, 1.5, 10000, 1, run=False, top=1, bottom=1, dir_suffix="_TB", disp_separate=True), [1]));
scenarios.append((LidCavity(0.01,1,  10, 1e-5,0.0001, 0.001, 1.5, 10000, 1, run=False, top=1, bottom=-1, dir_suffix="_TMB"), [0]));
scenarios.append((LidCavity(0.001, 0.2, 100, 1e-5,0.00001, 0.001, 1.5, 10000, 1, run=False, top=1, bottom=-1, dir_suffix="_TMB", disp_separate=True), [1]));
scenarios.append((LidCavity(0.01,1,  10, 1e-5,0.0001, 0.001, 1.5, 10000, 1, run=False, top=3, bottom=1, dir_suffix="_3TB"), [3]));
```

Figure 14: Illustrates how scenarios can be setup.

2.1.2 Automated plotting and controls

The user can control the plot using the array at the end of the scenario entry. The framework adopts a grouping mechanism where, if a scenarios that are tagged with 0 will be plotted in a single subplot, and tagged as 1 will be plotted in another subplot and so on. A simulation can be a member of more than 1 type of group. There is also no limit to the amount of plot groups that should be created by the framework with a small constraint that an appropriate group message has to be entered to the dictionary as shown in the below figure below, this is to ensure that the readability of the plots is preserved.

```
group_messages = {0: 'Re = 10 and Beta = 1.5',
                  1: 'Re = 100 and Beta = 1.5',
                  2: 'Different Beta values 0.5, 1.0 and 1.5',
                  3: 'Re = 10 and (T,B,L,R) = (3,+/-1,0,0)',
                  4: 'Re = 100 and (T,B,L,R) = (3,+/-1,0,0)'}

```

Figure 15: Illustrates how group messages can be setup.

2.1.3 Automated storing and loading with compression

The framework utilizes **picker** python package to perform storage operations. The scenario's contain a **run** parameter that when set to **True** will start the simulation and store the simulation results in an appropriate directory (automated), the user can influence the directory name by suffixing it using the **dir_suffix** parameter. If the **run** parameter is set to **False** the framework will automatically fetch the stored results related to the scenario from the previous execution and load it to the environment.

The data stored by the pickler can be either compress or uncompressed, this is controlled by setting the flag **is_compressed = True or False**.

2.1.4 Lid Cavity class structure

The **LidCavity** class is setup that accepts various hyper-parameters that can be used to tweak the nature of simulation. Some variables are setup for future proof in an event of any extensions made to the code like end_time that would allow the end-user to tweak the code to iterate based on max time rather than convergence.

The parameters **top, bottom, left, and right** are the most notable variables that is used to convert any rigid wall to a moving wall. If **U** is the magnitude with which the top lid is moved, then the **top** parameters has the following effect on the code,

- If top value is equal to 0, the wall is rigid as $top * U = 0 * U = 0$.
- If top value is equal to n, the wall is a moving wall with $top * u = n * u$ magnitude. The **n** value can also be negative.

This setup eliminated the need to make any code to the wall conditions with respect to the scenarios. The complete class structure can be found in the **Appendix A.1**.

2.2 Stream function

In this section we will look at how equation 1 is used to obtain the discretized stream function. The discretization of the stream function will be performed using the central differential approximation.

Applying CDF on equation 1 we will obtain the discritized stream function as follows,

$$\frac{u^n_{i+1,j} + u^n_{i-1,j} + u^n_{i,j+1} + u^n_{i,j-1} - 4u^n_{i,j}}{h^2} = -w^n_{i,j} \tag{5}$$

Reordering the equation and solving for $u^n_{i,j}$ we get the stream function equation,

$$u^n_{i,j} = 0.25(u^n_{i+1,j} + u^n_{i-1,j} + u^n_{i,j+1} + u^n_{i,j-1} + h^2 w^n_{i,j} \tag{6}$$

Successive over relaxation is an iterative optimization that can be applied to the stream function to induce an early convergence. This is possible as the iteration processes the grid points, half of the points would have already been updated.

Stream function will successive over relaxation is as follows,

$$u^n_{i,j} = \beta 0.25(u^n_{i+1,j} + u^n_{i-1,j} + u^n_{i,j+1} + u^n_{i,j-1} + h^2 w^n_{i,j} + (1-\beta)u^n_{i,j} \tag{7}$$

2.2.1 Pseudo-code

Algorithm 1: Stream Function

```
// n => is the size of the cavity.
// stream_func_acc => Stream function collection/array.
// vorticity => vorticity collection of the viscous fluid.
// h => Cavity step size.
// Beta => Relaxation factor.
function stream_function(n, stream_func_acc, vorticity, h, Beta)
    i ← 0 to n;
      j ← 0 to n;
        // compute stream function
    ;     stream_func_acc[i, j] = Equation  7;
    return stream_func_acc

```

The complete implementation of this function is found in the **Appendix A.2**

2.3 Vorticity

In this section, we will discuss how equation 2 is used to obtain the discretized vorticity function using Forward differential approximation.

Applying CDF to equation 2, we obtain the discretized vorticity function as follows,

$$\frac{w^{n+1}_{i,j} - w^n_{i,j}}{dt} = \left(\frac{w^n_{i+1,j} - w^n_{i-1,j}}{2h}\right)\left(\frac{w^n_{i,j+1} - w^n_{i,j-1}}{2h}\right) - \left(\frac{w^n_{i,j+1} - w^n_{i,j-1}}{2h}\right)\left(\frac{w^n_{i+1,j} - w^n_{i-1,j}}{2h}\right) + \frac{1}{Re}\left(\frac{w^n_{i+1,j} + w^n_{i-1,j} + w^n_{i,j+1} + w^n_{i,j-1} - 4w^n_{i,j}}{h^2}\right) \tag{8}$$

Solving the equation 8 for $w^{n+1}_{i,j}$ we can obtain the equation to compute the vorticity at the new future time, it can be expressed as shown below,

$$w^{n+1}_{i,j} = w^n_{i,j} - dt[\left(\frac{w^n_{i,j+1} - w^n_{i,j-1}}{2h}\right)\left(\frac{w^n_{i+1,j} - w^n_{i-1,j}}{2h}\right) - \left(\frac{w^n_{i+1,j} - w^n_{i-1,j}}{2h}\right)\left(\frac{w^n_{i,j+1} - w^n_{i,j-1}}{2h}\right) + \frac{1}{Re}\left(\frac{w^n_{i+1,j} + w^n_{i-1,j} + w^n_{i,j+1} + w^n_{i,j-1} - 4w^n_{i,j}}{h^2}\right)] \tag{9}$$

2.3.1 Boundary conditions

The sections lists the discretized four boundary conditions for vorticity under the no-slip assumption is as follows,

Given vorticity for top wall,

$$w_{topwall} = (u_{i,j=1} - u_{i,j=2})\frac{2}{h^2} - V_{wall}\frac{2}{h} + O(h) \tag{10}$$

Bottom wall,

$$w_{botwall} = (u_{i,j=ny} - u_{i,j=ny-1})\frac{2}{h^2} + V_{wall}\frac{2}{h} + O(h) \tag{11}$$

Left wall,

$$w_{leftwall} = (u_{i=1,j} - u_{i=2,j})\frac{2}{h^2} + V_{wall}\frac{2}{h} + O(h) \tag{12}$$

Right wall,

$$w_{rightwall} = (u_{i=nx,j} - u_{i,j=nx-1})\frac{2}{h^2} - V_{wall}\frac{2}{h} + O(h) \tag{13}$$

Here, the V_{wall} is the velocity with which the lid or the moving wall is traversed. Also, in programming languages like python that start with "0" index collections, the i and j values will be 0 instead of 1.

2.3.2 Pseudo-code

The pseudo-code is designed with an intention to allow us to move any 4 walls if required and keep them as rigid wall otherwise.

Algorithm 2: Compute Vorticity of the fluid

```
// U => The velcoity with which the lid should be opened.
// top / bottom/ left.  right => integer value that is set to 1 if the
   respective wall should be moved.  Values greater than 1 will multiply the
   velocity of U with its magnitude, -ve values are also accepted that will
   add negative U value.
function vorticity(n, stream_func_acc, vorticity, h, h_sqr, Re, dt, U, top,
 bottom, left, right)
    i ← 0 to n;
      j ← 0 to n;
        if Not a boundary condition then
            vorticity[i, j] = Equation 9;
        else if Top wall boundary condition then
            vorticity[i=1 to n-1, n-1] = Equation 10
        else if Bottom wall boundary condition then
            vorticity[i=1 to n-1, j=1] = Equation 11
        else if Left wall boundary condition then
            vorticity[i=1, j=1 to n-1] = Equation 12
        else if Right wall boundary condition then
            vorticity[i=n-1, j=1 to n-1] = Equation 13
    return vorticity
```

The complete implementation of this function is found in the **Appendix A.3**.

2.4 Momentum equations

The velocity V is related to the stream function u and vorticity as follows,

$$V_x = \frac{\partial u}{\partial y}, V_y = \frac{\partial u}{\partial x} \tag{14}$$

$$w = \frac{\partial V_y}{\partial x} - \frac{\partial V_x}{\partial y} \tag{15}$$

From equation 14 we see that the x-momentum (horizontal) is a partial derivative on y and the vertical velocity y-momentum is a partial derivative of x. Indicating that the x and y momentum is influenced by the stream functions on the opposite axis.

Using the equation 14, the discretized momentum equations are as follows,

$$Vx_{i,j} = \frac{(u_{i,j+1} - u_{i,j-1})}{2h} \tag{16}$$

$$Vy_{i,j} = \frac{(u_{i+1,j} - u_{i-1,j})}{2h} \tag{17}$$

The boundary condition of the x-momentum across the walls is $Vxi, j = ny - 1 = 1$ and the y-momentum is $Vy_{i,j=ny-1} = 0$; and both x and y-momentum is 0 when $i = nx - 1$.

2.4.1 Pseudo-code

Algorithm 3: Momentum Function

```
// x_momentum => The velocity of the fluid flow across x-axis.
// y_momentum => The velocity of the fluid flow across y-axis.
function calculate_x_y_momentum(n, stream_func_acc, x_momentum, y_momentum, h)
    i ← 0 to n;
        j ← 0 to n;
            if not boundary condition then
                
                x_momentum[i, j] = Equation 16
                    y_momentum[i, j] = Equation 17
    else if j = n-1 then
                
                x_momentum[i, j] = 1
                    y_momentum[i, j] = 0
    else
                
                x_momentum[i, j] = 0
                    y_momentum[i, j] = 0
    return (x_momentum, y_momentum)

```

The complete implementation of this function is found in the **Appendix A.4**.

2.5 Pressure

The pressure field can be obtained by solving the following Poisson equation,

$$\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} = 2\left[\frac{\partial^2 u}{\partial x^2}\frac{\partial^2 u}{\partial y^2} - \left(\frac{\partial^2 u}{\partial x \partial y}\right)^2\right] \tag{18}$$

On solving the Poisson equation and solving for pressure at n. For readability, the equation is split into two parts equation 19 and 20.

$$rhs^n_{i,j} = \left(\frac{u^n_{i-1,j} - 2u^n_{i,j} + u^n_{i+1,j}}{h^2}\right)\left(\frac{u^n_{i,j-1} - 2u^n_{i,j} + u^n_{i,j+1}}{h^2}\right) - \left(\frac{u^n_{i+1,j+1} - u^n_{i+1,j-1} - u^n_{i-1,j+1} + u^n_{i-1,j-1}}{4h^2}\right) \tag{19}$$

$$P^n_{i,j} = \frac{P^{n-1}_{i+1,j} + P^{n-1}_{i-1,j} + P^{n-1}_{i,j+1} + P^{n-1}_{i,j-1}}{4} - \frac{(rhs^n_{i,j})h^4}{2} \tag{20}$$

From the pressure equation 20, the first part of the equation computes an average pressure across the 4 corner points with respect to the current i and j position. And the second component rhs will introduce the influence of stream function over the pressure.

2.5.1 Pseudo-code

Algorithm 4: Momentum Function

```
// x_momentum => The velocity of the fluid flow across x-axis.
// y_momentum => The velocity of the fluid flow across y-axis.
function calculate_pressure(n, pressure, pressure_old, stream_func_acc, h, rhs)
    i ← 0 to n;
      j ← 0 to n;
        // Compute the Stream function influence on pressure
        rhs[i, j] = Equation 19;
        // Compute the pressure, find the average pressure around the point
       and add the RHS component
        pressure[i, j] = Equation 20 - 0.5( ( rhs[i, j] * h^4))
    pressure_old = pressure.copy()
    return (pressure, pressure_old)

```

The complete implementation of this function is found in the **Appendix A.5**.

2.6 Limitations

The algorithm has a strong limitation where,

$$\frac{\Delta t}{Re\ h^2} <= 0.25 \tag{21}$$

If the simulation is run without satisfying the above limitation, the program will run into underflow and overflow errors. That is, the value computed during the simulation is either smaller than the minimum floating point value or the maximum floating point value respectively.

Accuracy of floating point values in python is bad, as it utilizes hardware's binary floating-point arithmetic. The arithmetic operations performed on float values can lead to some surprising outcomes, as python adopts approximations while computing floats. Using decimals can slightly elevate this issue as it is less surprising but the original issue with respect to the accuracy still remains.

In an event where convergence is not achieved even after max_iteration, it is assumed that there is no stable state for the configuration. But a stable solution might exists at a higher iteration value which is limited by the hardware and time.

It is difficult to parallelize the algorithm, as some of the equations like stream function and vorticity utilize the results from their neighbouring points. Parallel process might lead to race conditions that lead to unexpected outcomes.

3 Result and Discussion

The section will depict the various scenario and the respective observations made during this experimentation. The following are the list of scenarios experimented in the project,

Scenario	**Name**	**Re**	**dt**	**N**	**(Top, Bottom, Right, Left)**
1	Moving top wall Re 10	10	0.0001	100	(1, 0, 0, 0)
2	Moving top wall Re 100	100	0.00001	200	(1, 0, 0, 0)
3	Moving top and bottom wall Re 10	10	0.0001	100	(1, 1, 0, 0)
4	Moving top and bottom wall Re 100	100	0.00001	200	(1, 1, 0, 0)
5	Moving top and bottom same spin Re 10	10	0.0001	100	(1, -1, 0, 0)
6	Moving top and bottom same spin Re 100	100	0.00001	200	(1, -1, 0, 0)
7	Moving top and bottom wall 3:1 momentum Re 10	10	0.0001	100	(3, 1, 0, 0)
8	Moving top and bottom wall 3:1 momentum Re 100	100	0.00001	200	(3, 1, 0, 0)
9	Moving top and bottom wall 3:1 momentum same spin Re 100	10	0.0001	100	(3, -1, 0, 0)
10	Moving top and bottom wall 3:1 momentum same spin Re 100	100	0.00001	200	(3, -1, 0, 0)
11	Moving Top and Right wall Re 10	10	0.0001	100	(1, 0, 1, 0)
12	Moving Top and Right wall Re 100	100	0.00001	200	(1, 0, 1, 0)
13	Relaxation beta = 1, Re 10	10	0.0001	100	(1, 0, 0, 0)
14	Relaxation beta = 0.5, Re 10	10	0.0001	100	(1, 0, 0, 0)
15	Coupled vortex (Top-Right and Bottom-Left) Re 10	10	0.0001	100	(1, 1, 1, 1)

16	Coupled vortex 3:1 momentum (Top-Right and Bottom-Left) Re 10	10	0.0001	100	(3, 1, 1, 3)
17	4 side moving wall, opposite spin directions Re 10	10	0.0001	100	(1, -1, -1, 1)
18	4 side moving wall, opposite spin directions Re 100	100	0.00001	200	(1, -1, -1, 1)
19	4 side moving wall, opposite spin directions Re 127	127	0.00001	200	(1, -1, -1, 1)
20	4 side moving wall, total convergence, all same spin Re 10	10	0.0001	100	(1, -1, 1, -1)
21	4 side moving wall, opposite spin directions Re 10 3:1 momentum	10	0.0001	100	(3, -1, -1, 3)
22	4 side moving wall, opposite spin directions Re 100 3:1 momentum	100	0.00001	200	(3, -1, -1, 3)

The complete metrics captured of all the scenarios can be found in the Appendix A.7

3.1 Moving top wall Re 10, N = 100, dt = 0.0001

A normal scenario where on the top lid is a moving a wall, the primary expectation is to observe a primary vortex at the center and smaller secondary and possibly tertiary vortexes around the edges of the cavity. We will look at the Stream function, vorticity, pressure and momentum's of this simulation.

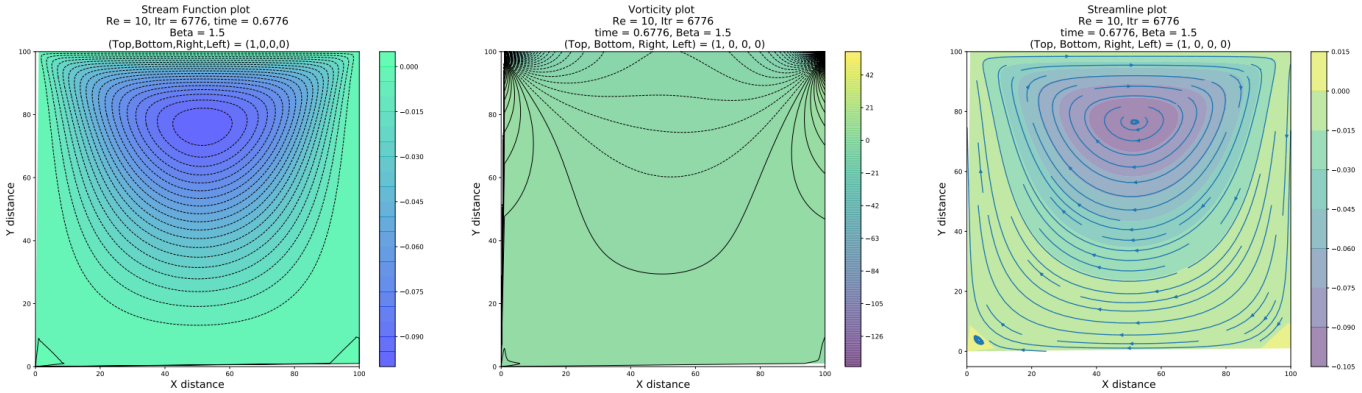

Figure 16: Scenario 1; Stream function, vorticity and, Stream line plots

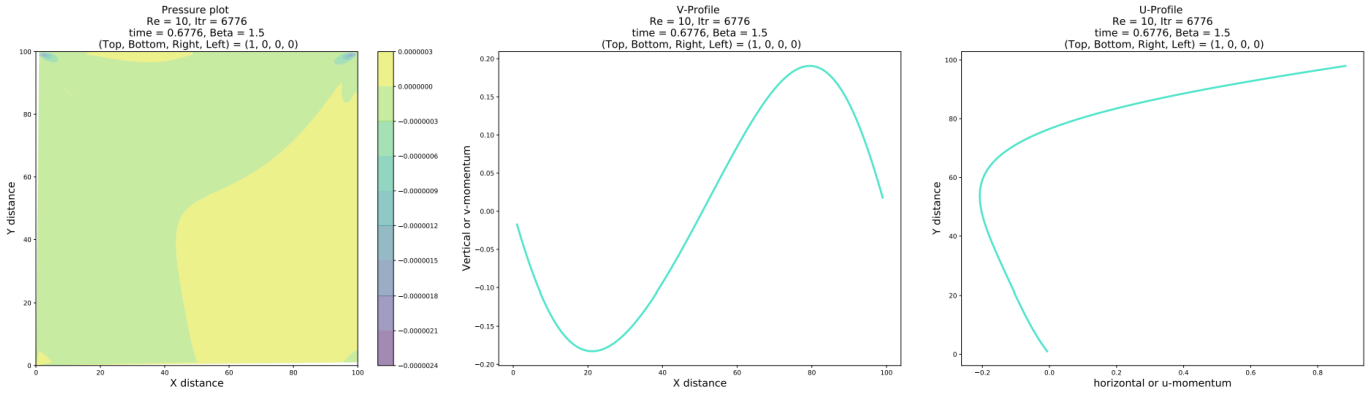

Figure 17: Scenario 1; Pressure, v-profile and, u-profile plots.

From the Stream function, vorticity and the streamline plots, we can clearly see that the primary center vortex is form as expected along with the additional secondary vortex's at the bottom left and bottom right corners of the plot. The streamline plot also shows the fluid movement that lead to the formation of the vortex's.

As mentioned earlier, the pressure at the core of the vortex is expected to be less in comparison to the pressure away from it. We can see a higher pressure around the top edges of the pressure plot in comparison to its center.

The u-profile and the v-profile of the scenario Re 10 can be observed in figure 17, the profiles match with the reference u- and v-profiles as shown in Figures 12 and 13. Similar to the reference, the velocity starts at 0.0 and ends at 1.0 with a single arc like formation.

The captured simulation metrics is as follows,

Scenario #	**Convergence**	**Iterations**	**Execution Time**	**Error <Threshold (1e-5)**
1	Yes	6776	8.9959 Minutes	9.99250e-06

3.2 Moving top wall Re 100, N = 200, dt = 0.00001

A normal scenario where the top lid is moving, the configuration of this simulation is exactly similar to that of the previous scenario. Except for the few parametric differences where, the Reynolds number Re is equal to 100, with a grid size of 200 and the time steps is valued at 0.00001.

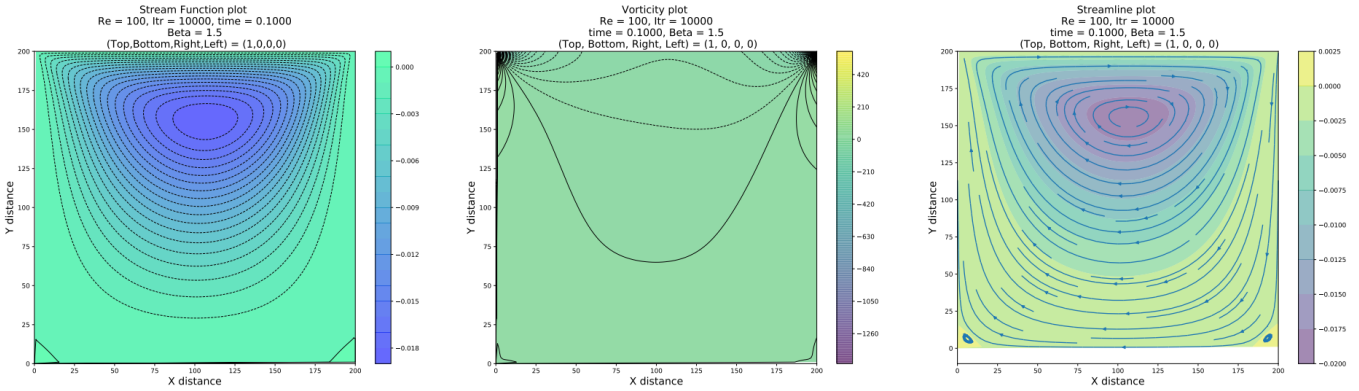

Figure 18: Scenario 2; Stream function, vorticity and, Stream line plots

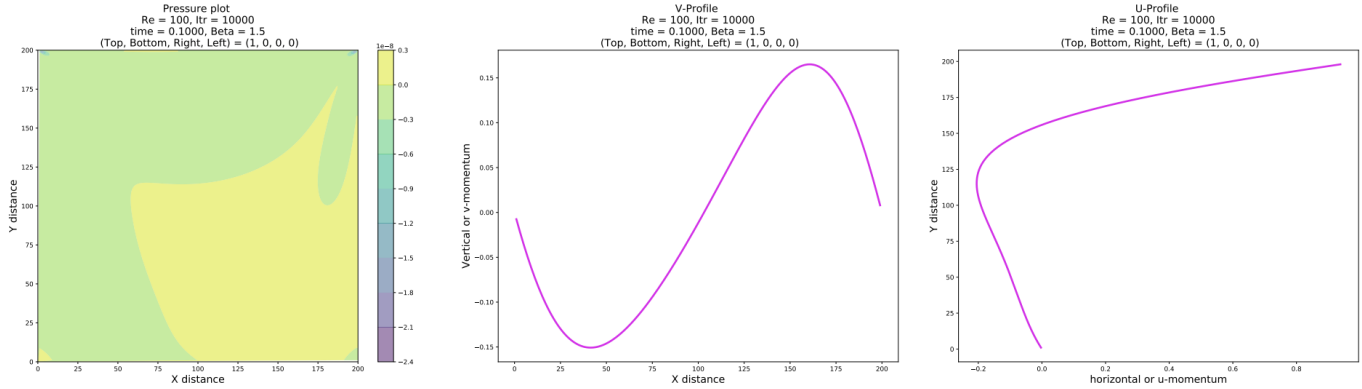

Figure 19: Scenario 2; Pressure, v-profile and, u-profile plots.

The expectation is similar to the previous scenario where a primary vortex followed by some secondary vortexes around the corners are expected. Figure 19 shows the primary and secondary vortex formations thus confirming the experimental results to the expectations.

The pressure is also similar to that of the Re =10 scenario, where the pressure is lower towards the core of the vortex and higher around the edges of the moving wall.

The u-profile and the v-profile of the scenario Re 100 can be observed in figure 17, the profiles match with the reference u- and v-profiles as shown in Figures 12 and 13. Similar to the reference, the velocity starts at 0.0 and ends at 1.0 with a single arc like formation.

Scenario #	**Convergence**	**Iterations**	**Execution Time**	**Error <Threshold (1e-5)**
2	No	10000	52.4460 Minutes	0.00040

3.3 Moving top and bottom wall Re 10, N = 100, dt = 0.0001

In this scenario, the top and the bottom walls of the cavity will move with unit momentum. The simulation is performed with Re = 10, Grid size N = 100 and time step of 0.0001.

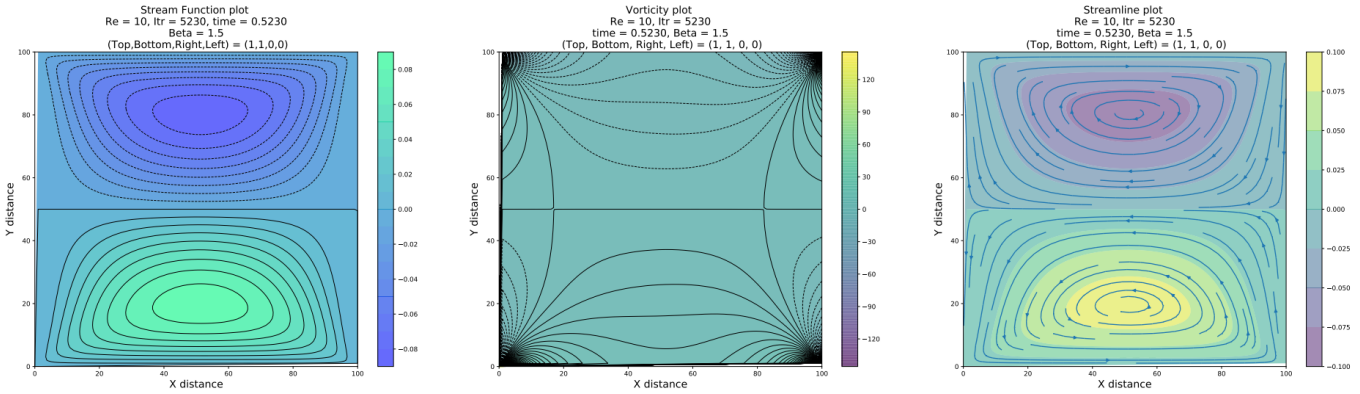

Figure 20: Scenario 3; Stream function, vorticity and, Stream line plots

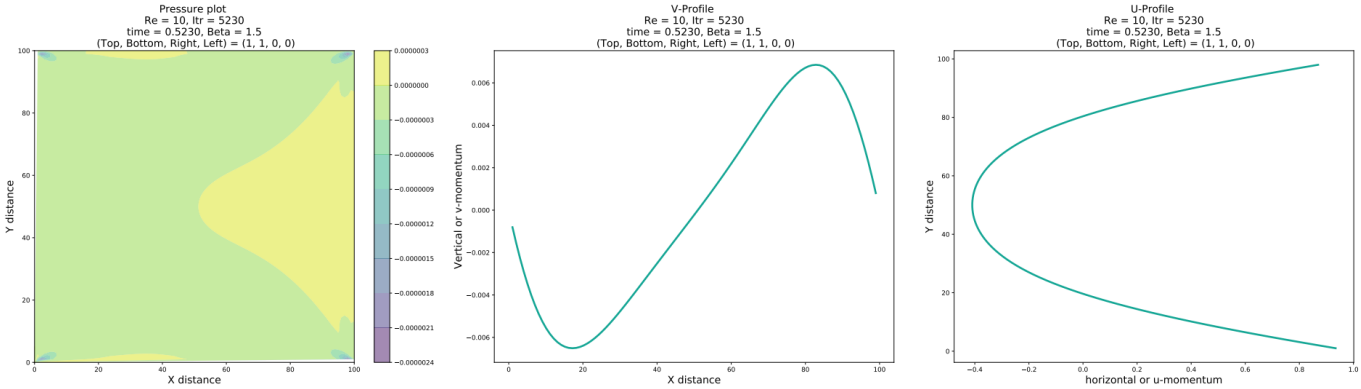

Figure 21: Scenario 3; Pressure, v-profile and, u-profile plots.

The expectation of the simulation is the formation of 2 mirror vortexes that do not merge to form a giant vortex. The vortexes should not merge since the wall vorticity conditions for Top and Bottom walls are opposite to each other as shown in Equation 10 and 11. We also earlier discussed that vortex that spin in the same direction tend to merge together and vortex's with opposite spins tend to find their own stable domain space away from the other vortex's.

From the Figure 20 we can observe that the 2 vortex's are formed that appear as the mirror images of each other. The vortex's have found their stable domain where they do not converge to form a bigger vortex. The steam-lines of the vortex's are also opposite to each other, if the image is viewed in sideways you can see that the stream-lines move away from each other and not into one another.

A variant of this simulation is to provide the momentum of the Top and Bottom wall with a 3:1 ratio. That is, the Top wall has 3 times higher influence than the bottom wall. Figure 22 and 23 shows the results of this simulation. As we can observe, the vortex formed around the top wall is larger than the bottom wall, and the voriticity is also pushed towards the bottom. Surprisingly the pressure plot appears similar to the previous scenario.

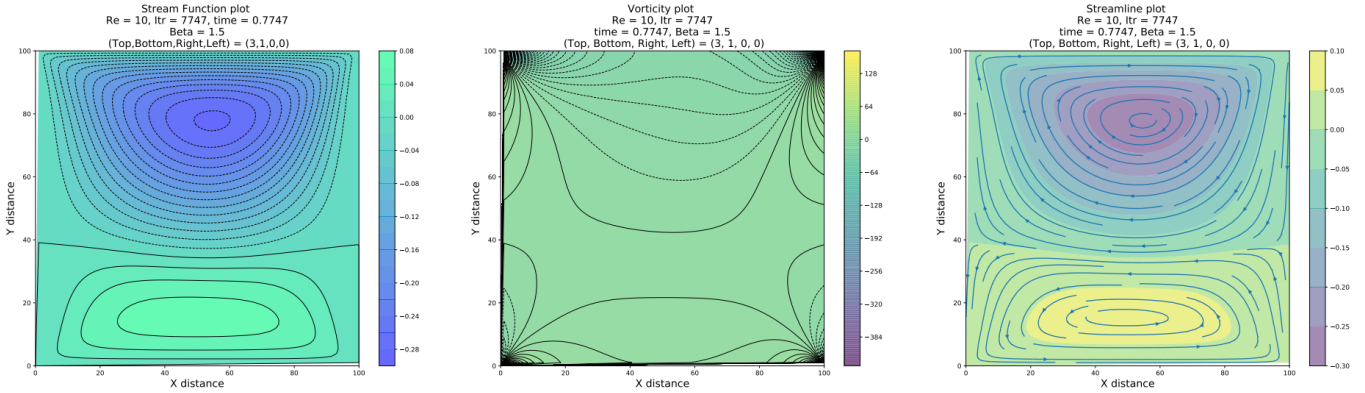

Figure 22: Scenario 7; Stream function, vorticity and, Stream line plots

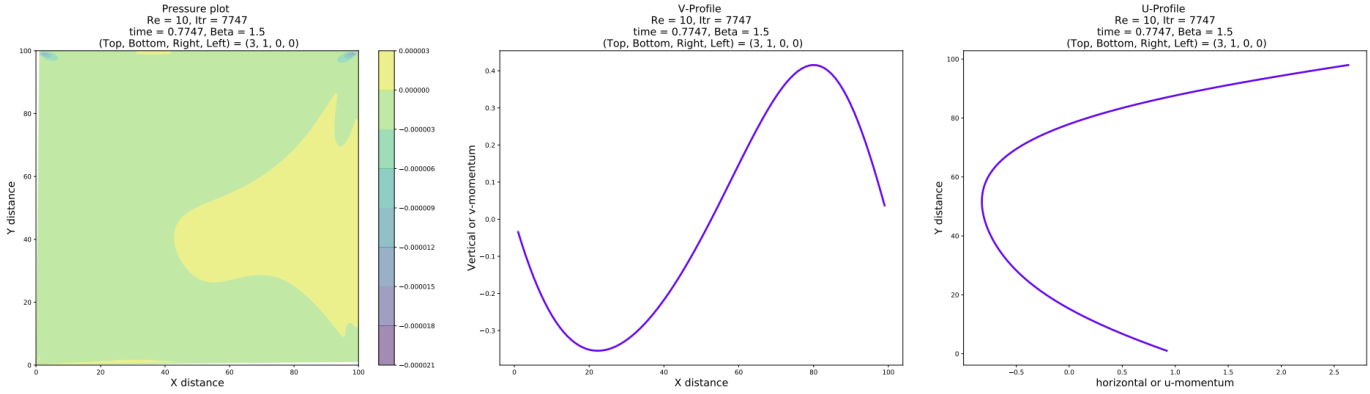

Figure 23: Scenario 7; Pressure, v-profile and, u-profile plots.

Scenario #	**Convergence**	**Iterations**	**Execution Time**	**Error <Threshold (1e-5)**
3	Yes	5230	6.7916 Minutes	9.99312e-06
7	Yes	7747	10.0208 Minutes	9.99890e-06

3.4 Moving top and bottom wall Re 100, N = 200, dt = 0.00001

A normal scenario where the top and the bottom walls are moving. The scenario is similar to the previously scenario discussed and the expectation of the simulation is the same. Where two mirror vortex's should be generated that find a stable state and not converge into a bigger vortex.

From the figure 24 we can see that the two vortex's are formed that have attain a stable state where they do not converge.

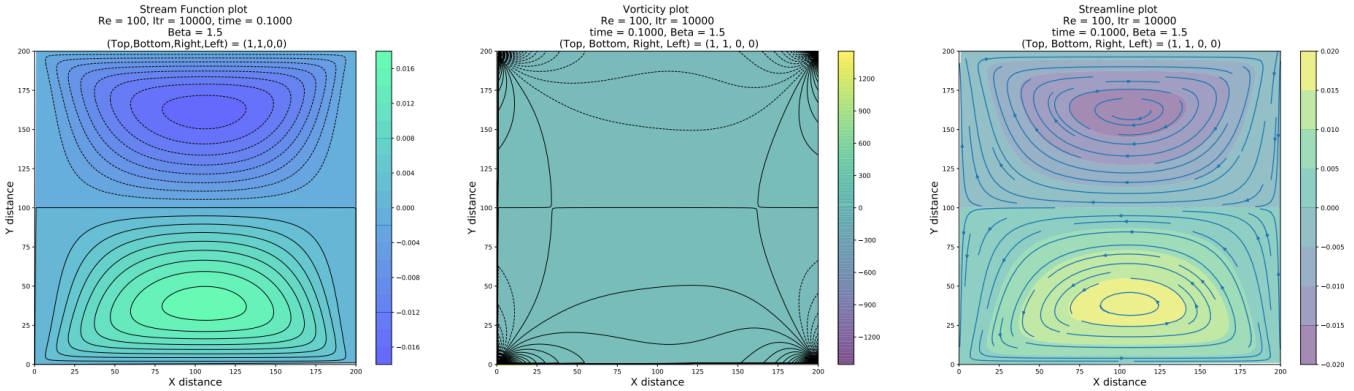

Figure 24: Scenario 4; Stream function, vorticity and, Stream line plots

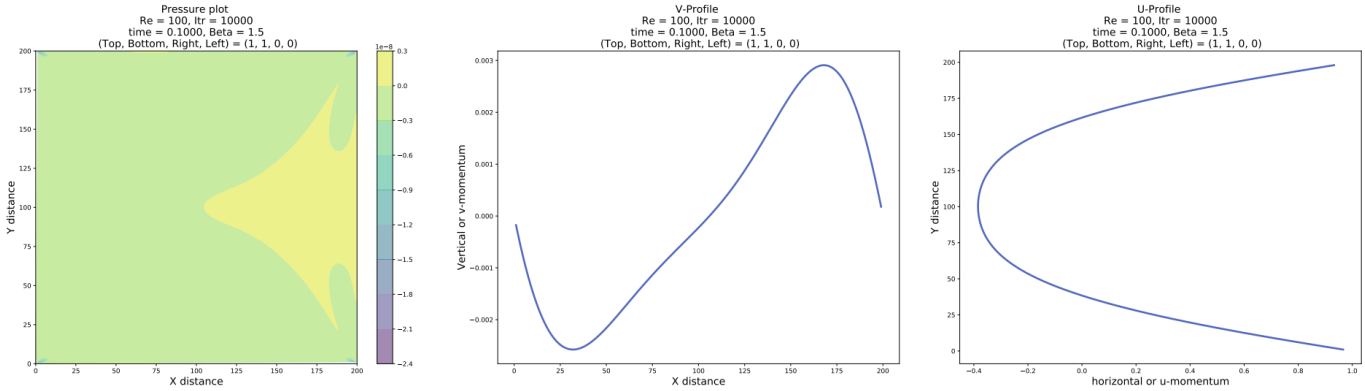

Figure 25: Scenario 4; Pressure, v-profile and, u-profile plots.

The equivalent results for the scenario where the Top and the Bottom wall influence is set to 3:1 is as follows.

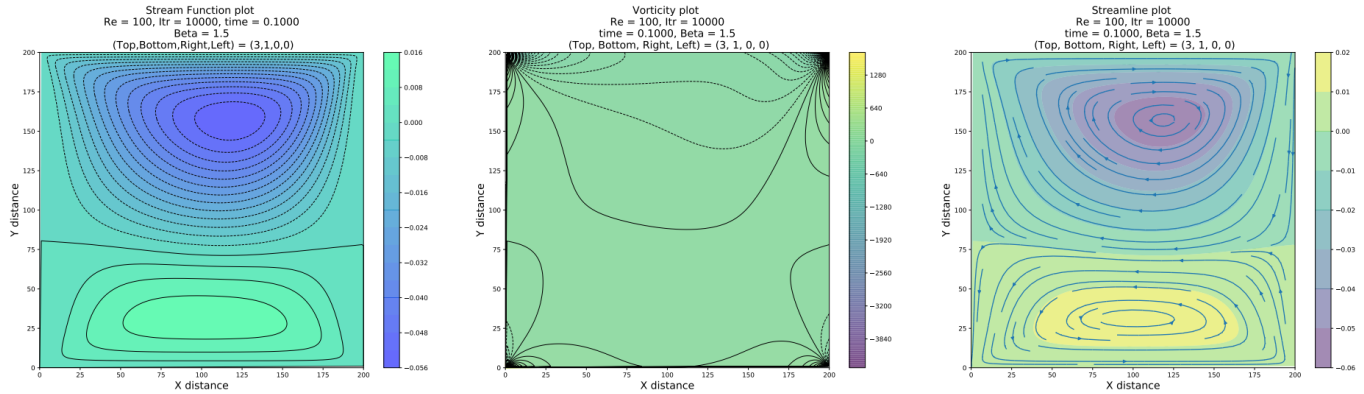

Figure 26: Scenario 8; Stream function, vorticity and, Stream line plots

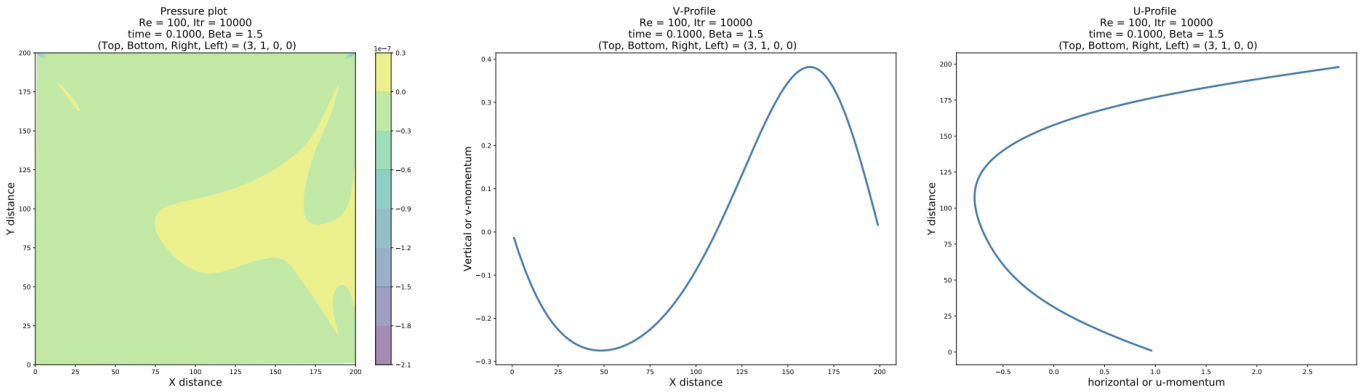

Figure 27: Scenario 8; Pressure, v-profile and, u-profile plots.

The observation is similar to the Re=10 scenario, where the top vortex is larger than the bottom vortex.

Scenario #	**Convergence**	**Iterations**	**Execution Time**	**Error <Threshold (1e-5)**
4	No	10000	52.3923 Minutes	0.00034
8	No	10000	52.9004 Minutes	0.00125

3.5 Moving Top and Bottom wall, same directional spin

In this scenario the Top and the Bottom wall are moving and have the same spin. The expectation of the scenario is that the two vortex's generated in either ends of the cavity will tend to converge into a bigger vortex.

In the Figures 29 and 29, we can see that in both the configuration setups Re =10 and Re=100, the two vortex's converge and form a bigger vortex that appears to be rather elongated in-comparison to the result obtained in scenario-1 Figure 16 and 18. The result obtained is within expectations as two equal forces influence the viscous liquid inside the cavity.

We can notable observe in the stream function and streamline plots Figures 28 and 28, that the output generated under the Re=100 configuration shows two vortex's at the center in-comparison to the Re=10 configuration. This is also observed in the pressure and momentum profile plots Figure 29 and 19, the pressure significantly appears more symmetric in the Re=100 configuration. This observation provides a hint that a higher Reynolds's number leads to better simulation accuracy. But on the downside, the simulation takes significantly higher iterations to arrive at a stable condition at Re = 100.

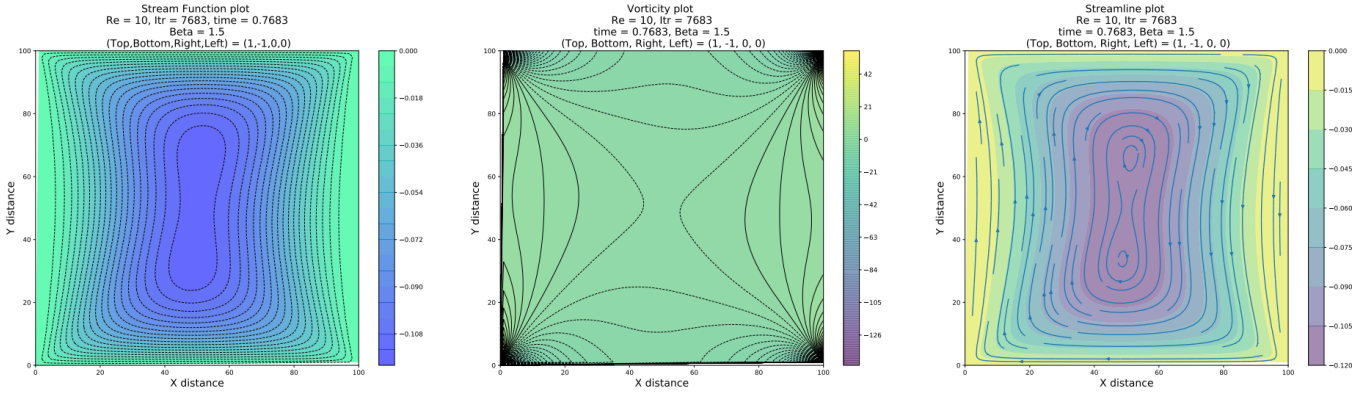

Figure 28: Scenario 5;Stream function, vorticity and, Stream line plots

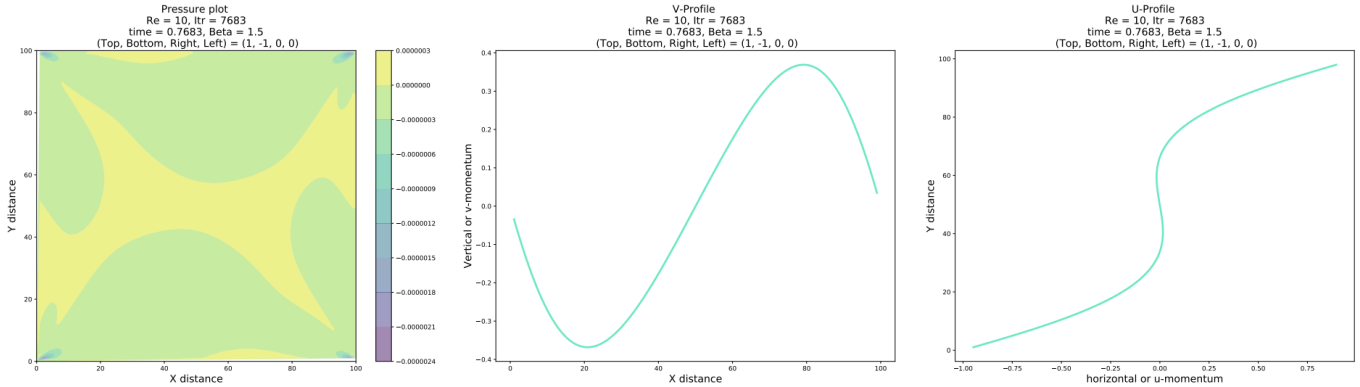

Figure 29: Scenario 5;Pressure, v-profile and, u-profile plots.

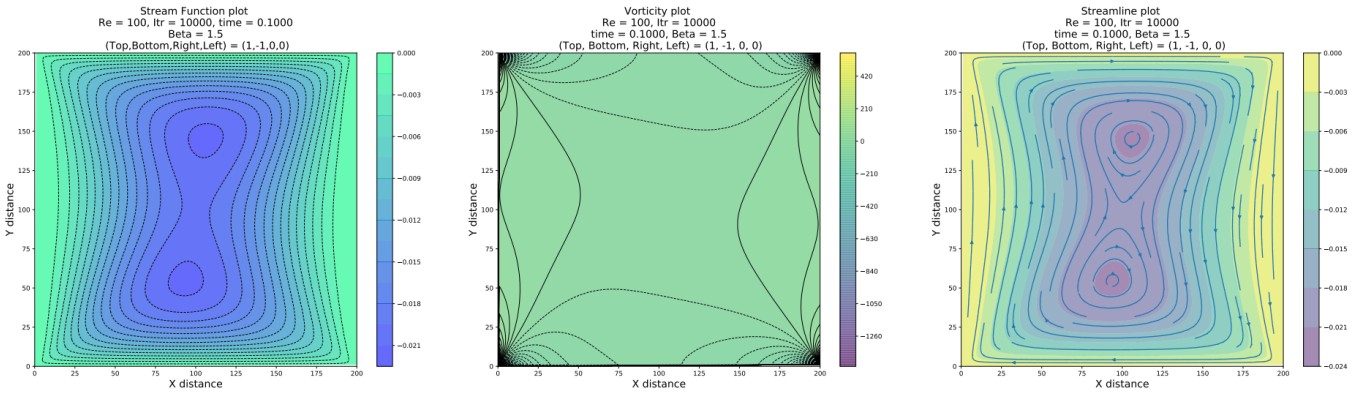

Figure 30: Scenario 6;Stream function, vorticity and, Stream line plots

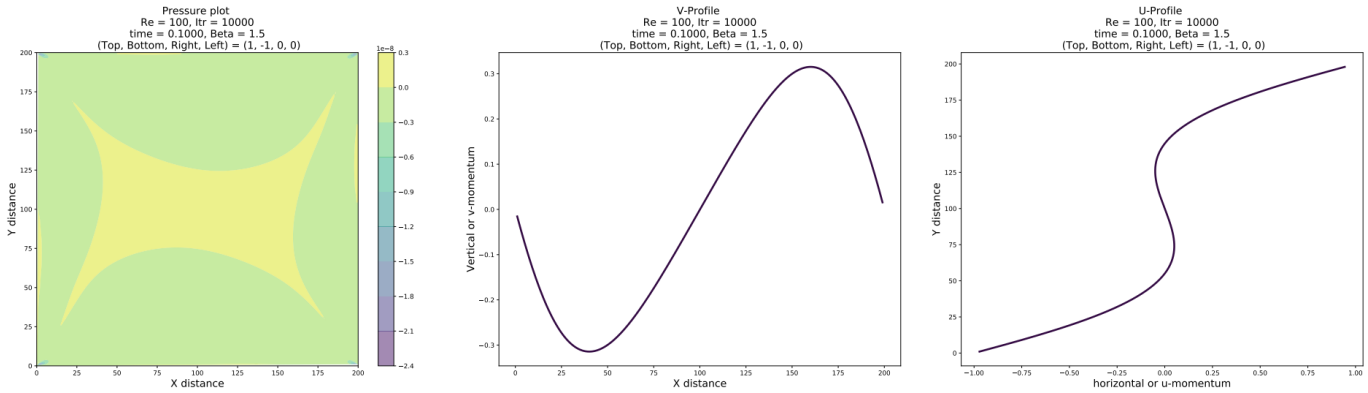

Figure 31: Scenario 6;Pressure, v-profile and, u-profile plots.

The scenarios below, are variants of the previous scenario. Where, the top lid has 3 times the magnitude in-comparison to the bottom layer. From the Figures 32 and 34, we can see that we can only find one distinctly large vortex in-comparison to the original scenario where there were 2 vortex's that were partially merged.

The pressure plots and U-profiles are also distinctly different in comparison in figures 33, 29 and figures 35, 31

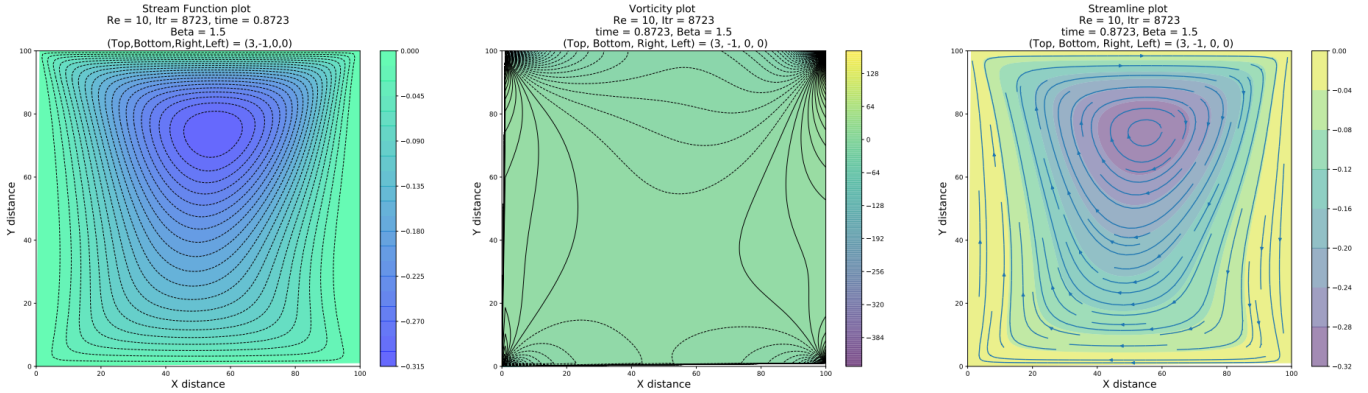

Figure 32: Scenario 9; Stream function, vorticity and, Stream line plots

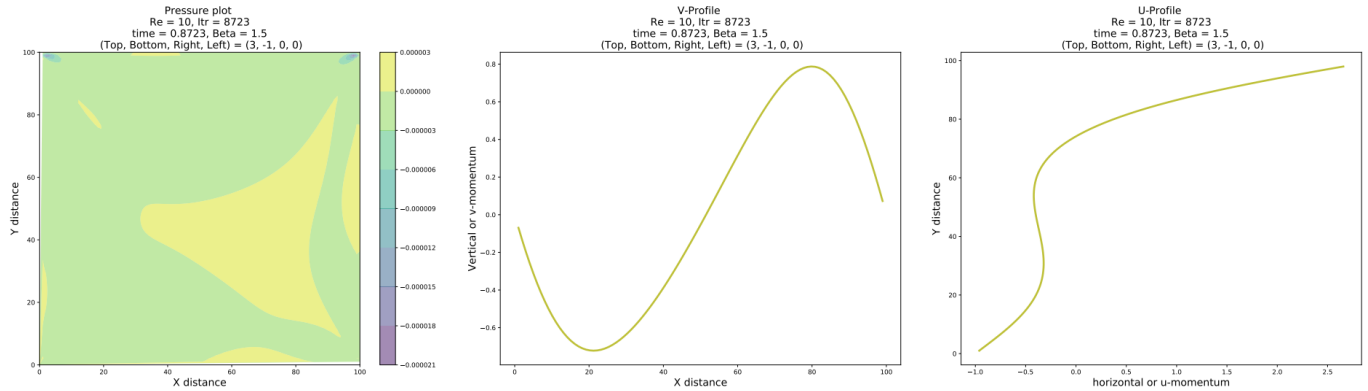

Figure 33: Scenario 9; Pressure, v-profile and, u-profile plots.

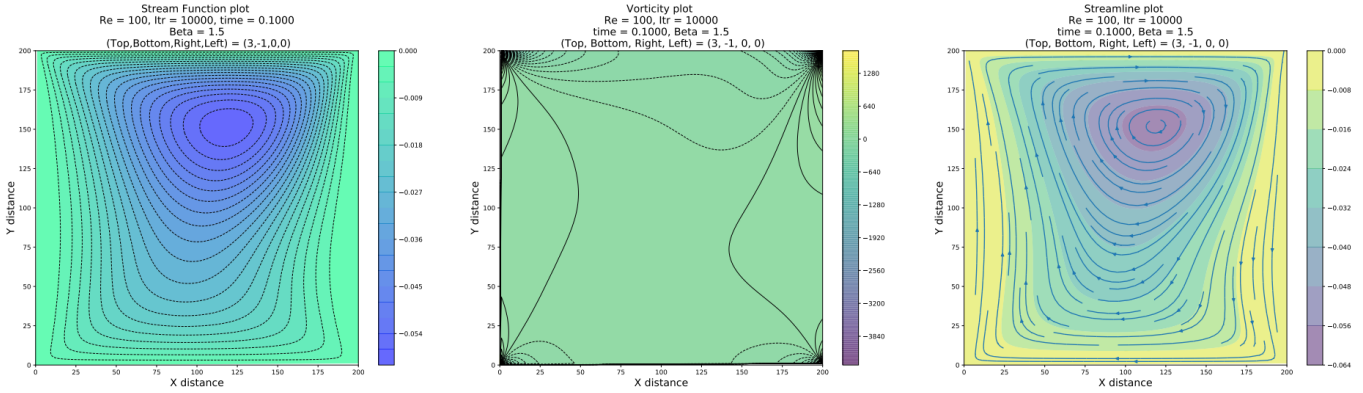

Figure 34: Scenario 10; Stream function, vorticity and, Stream line plots

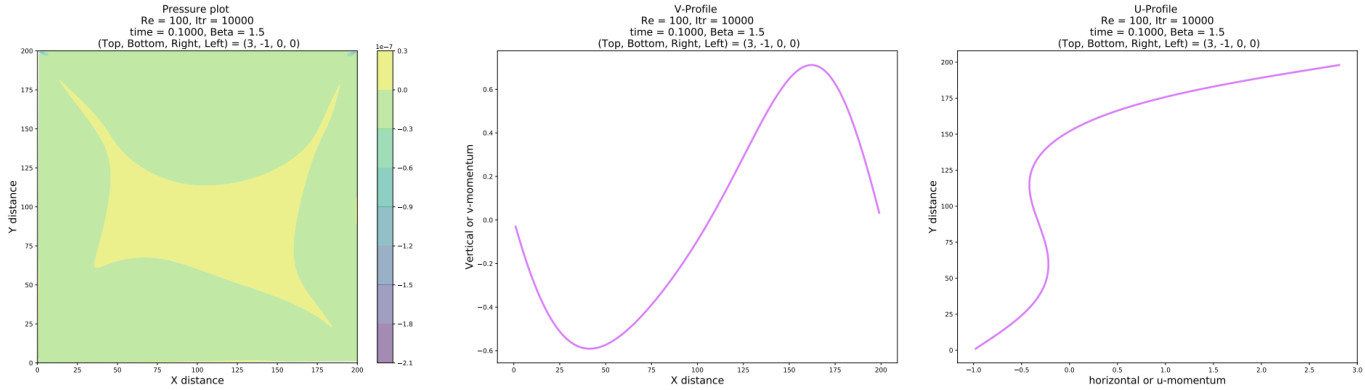

Figure 35: Scenario 10; Pressure, v-profile and, u-profile plots.

Scenario #	Convergence	Iterations	Execution Time	Error <Threshold (1e-5)
5	Yes	7683	9.9977 Minutes	9.99277e-06
6	No	10000	52.4148 Minutes	0.00057
9	Yes	8723	11.3908 Minutes	9.99435e-06
10	No	10000	52.3150 Minutes	0.00161

3.6 Comparing the vortex centers against reference

The section will compare the vortex centers obtained against the reference results that were depicted in Figure 9 and vortex co-ordinates in Table 8. The simulation is run executed in a start where Top, Bottom, Left and Right walls are moving with a magnitudes of 1, -1, 1, and -1 respectively. The magnitudes ensure that the vortexes do not spin in the same direction and merge to form a bigger vortex. As the primary goal of this experiment is to compare the 4 vortex scenario.

The Figures 36, 37, and 38 match the vortex patterns as observed in the reference scenario as shown in figure 9. We can observe that all the three scenarios Reynolds number 10, 100, and 127 shows 4 vortex's as shown in the reference. Both the experimental output and the reference do not show any fluid movements at the center of the cavity.
Experimental observations as shown in tables 2, 3, and 4 shows the deviation of the vortex centers

from the reference as shown in table 8. From the tables we can see that the experimentally obtained vortex centers deviate by a very small error margins after 1 to 2 decimal points (0.0XXX to 0.00XX), this is due to the different algorithmic approaches used to simulate the lid driven cavity experiment.

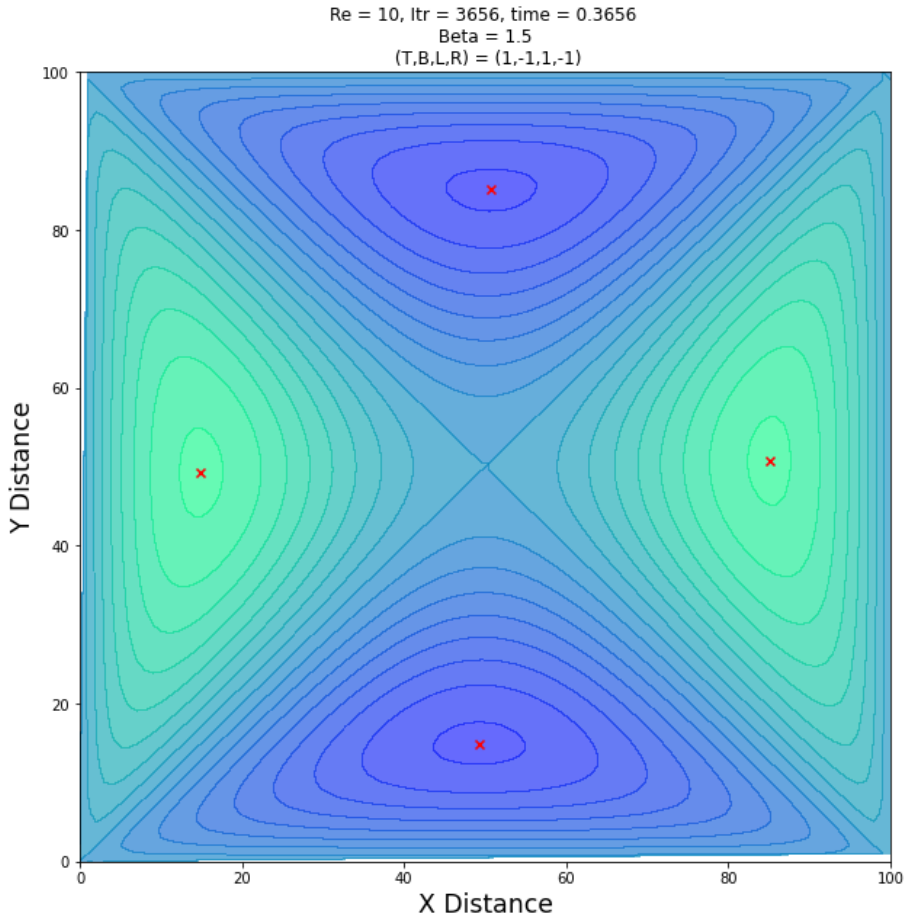

Figure 36: Vortex centers Re 10

RE=10	**Actual**		**Expected**		**Deviation**	
co-ord	**x**	**y**	**x**	**y**	**x**	**y**
Top	0.5072	0.8511	0.509	0.851	**0.0017**	**0.0001**
Bottom	0.4927	0.1488	0.497	0.155	**0.0043**	**0.0061**
Left	0.1488	0.4927	0.155	0.497	**0.0061**	**0.0043**
Right	0.8511	0.5072	0.851	0.509	**0.0001**	**0.0017**

Table 2: Vortex centers Re 10 from experimentation.

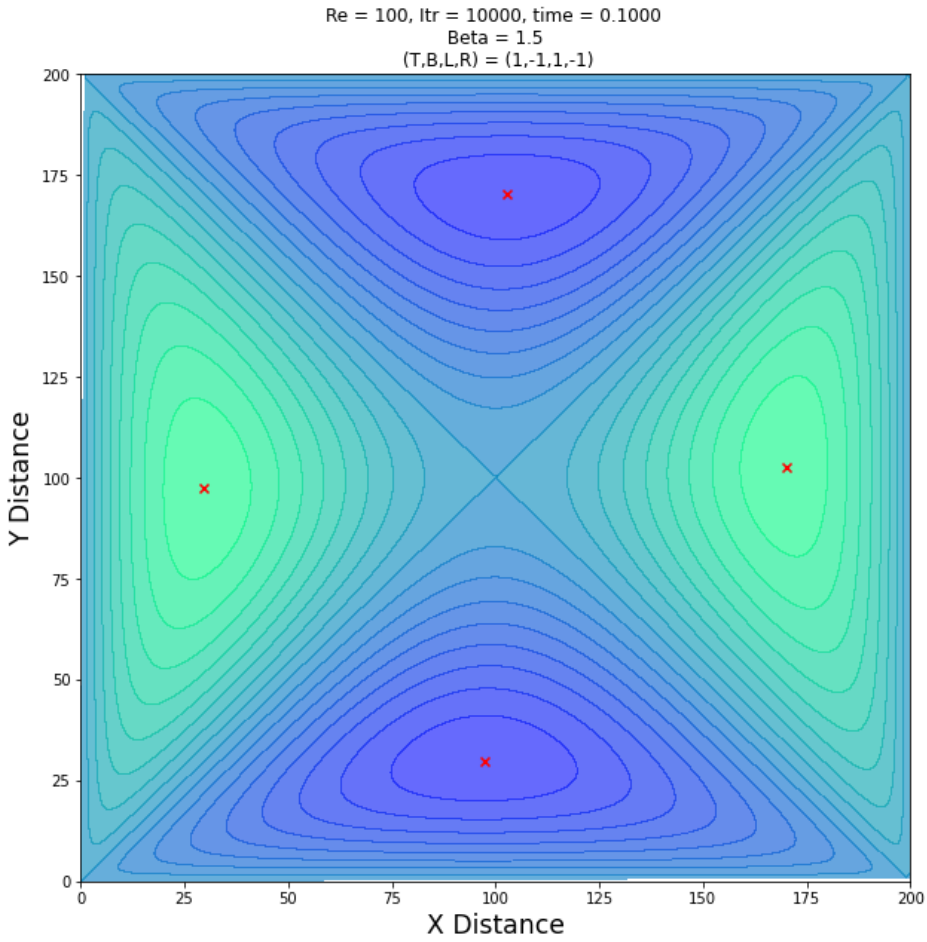

Figure 37: Vortex centers Re 100

RE=100	Actual		Expected		Deviation	
co-ord	x	y	x	y	x	y
Top	0.5135	0.8515	0.559	0.845	**0.0445**	**0.0125**
Bottom	0.4864	0.1484	0.442	0.161	**0.0455**	**0.0065**
Left	0.1484	0.4864	0.161	0.442	**0.0125**	**0.0445**
Right	0.8515	0.5135	0.845	0.559	**0.0065**	**0.0455**

Table 3: Vortex centers Re 100 from experimentation.

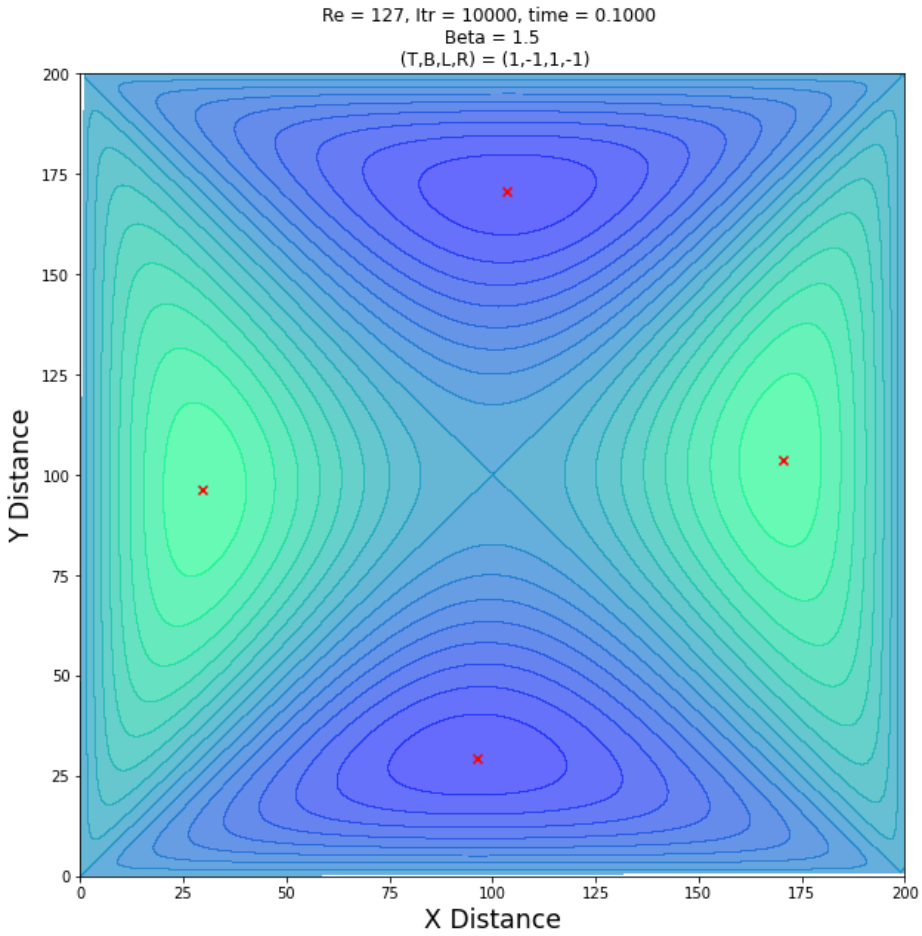

Figure 38: Vortex centers Re 127

RE=127	Actual		Expected		Deviation	
co-ord	x	y	x	y	x	y
Top	0.5178	0.8524	0.559	0.839	**0.0401**	**0.0205**
Bottom	0.4821	0.1475	0.442	0.168	**0.0411**	**0.0135**
Left	0.1475	0.4821	0.168	0.442	**0.0205**	**0.0401**
Right	0.8524	0.5178	0.839	0.559	**0.0135**	**0.0411**

Table 4: Vortex centers Re 127 from experimentation.

3.7 Moving wall Top and Right, same directional spin

We have seen the interaction of liquids when they are opposite to each other. The scenario here simulates a state where the Top and Right wall exhibit the moving wall property. The interaction of the fluid and the vortex formation is observed as shown in Figure 39 and 41.

The vortex formed is slightly oval or bloated as the two side by side forces merged to form a bigger vortex. Away from the vortex towards the bottom right portion of the cavity, we can see that the streamlines are not rigid and start to lose continuity. Indicating less powerful regions. The pressure plot in interesting as we can see that the pressure at the Top Left corner of the cavity is slightly

higher in-comparison to the other edges. As there is single big vortex, the V and U profiles does not display a sine wave like pattern.

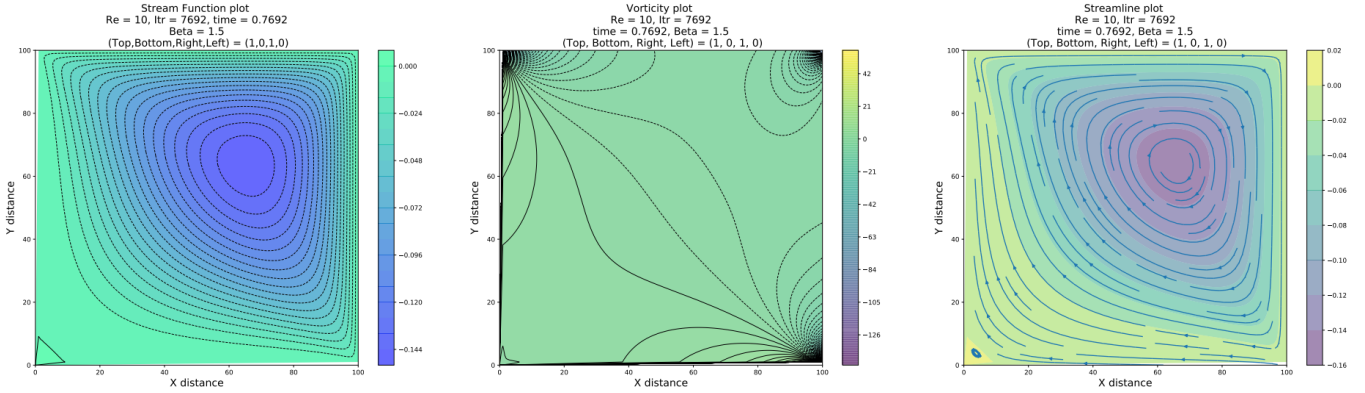

Figure 39: Scenario 11; Stream function, vorticity and, Stream line plots

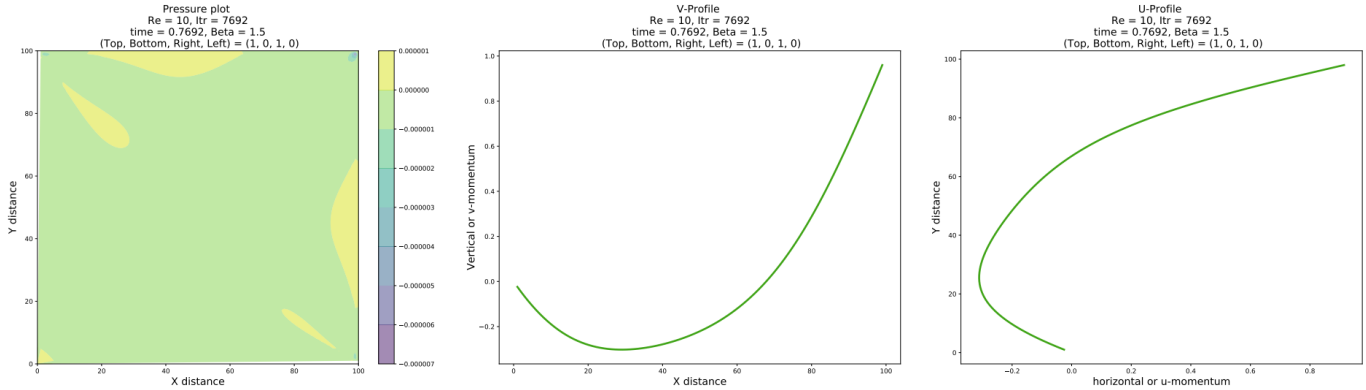

Figure 40: Scenario 11; Pressure, v-profile and, u-profile plots.

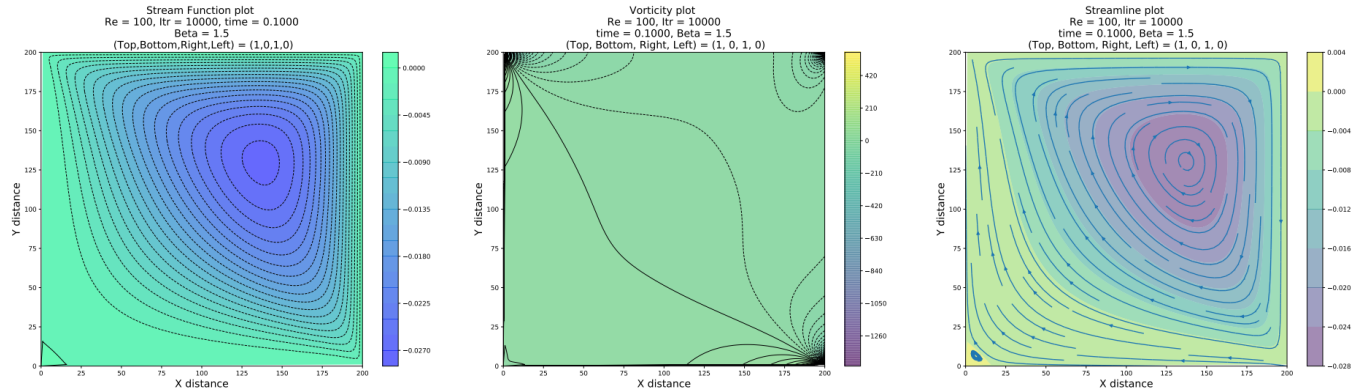

Figure 41: Scenario 12; Stream function, vorticity and, Stream line plots

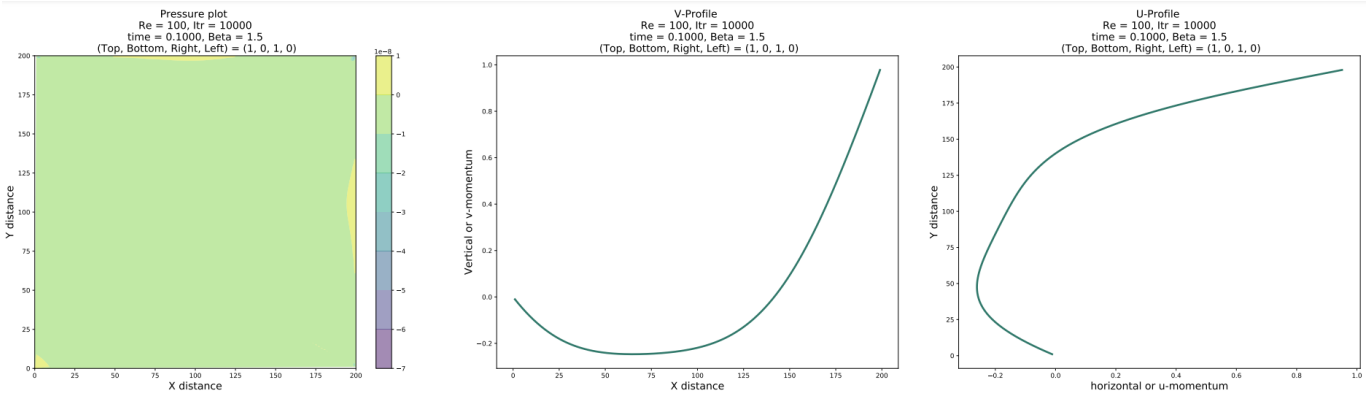

Figure 42: Scenario 12; Pressure, v-profile and, u-profile plots.

Scenario #	Convergence	Iterations	Execution Time	Error <Threshold (1e-5)
11	Yes	7692	9.8463 Minute	9.99273e-06
12	No	10000	52.6329 Minutes	0.00069

3.8 Influence of Relaxation factor (Beta)

The relaxation factor exists only to promote early convergence and thus if implemented correctly should have no effect on the underlying simulation result.

The simulation will be executed with the Re = 10 configuration for 0.5, 1.0 and 1.5 Beta values and cross-verified for expected scenario.

From the figures below, we see that there is no observable influence of the Beta value on the output of the simulation. But the simulation in Figure 43, shows that the execution terminated with *time* = 1.0000 indicating that there was **no convergence or stable state achieved** for this simulation, but convergence was attained for Beta = 1.0 and 1.5 at around time = 0.6 or Average iteration of 6500.

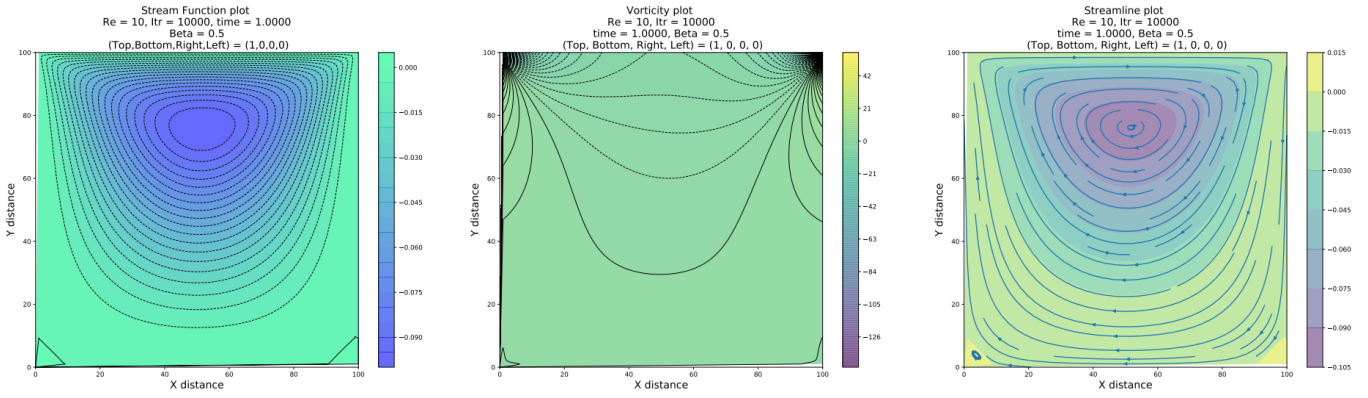

Figure 43: Scenario 14; Stream function, vorticity and, Stream line plots

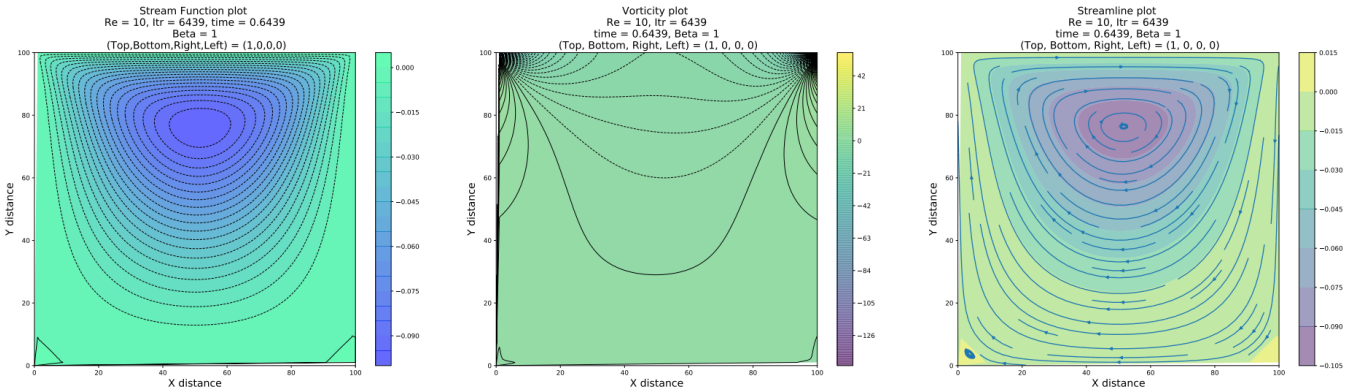

Figure 44: Scenario 13; Pressure, v-profile and, u-profile plots.

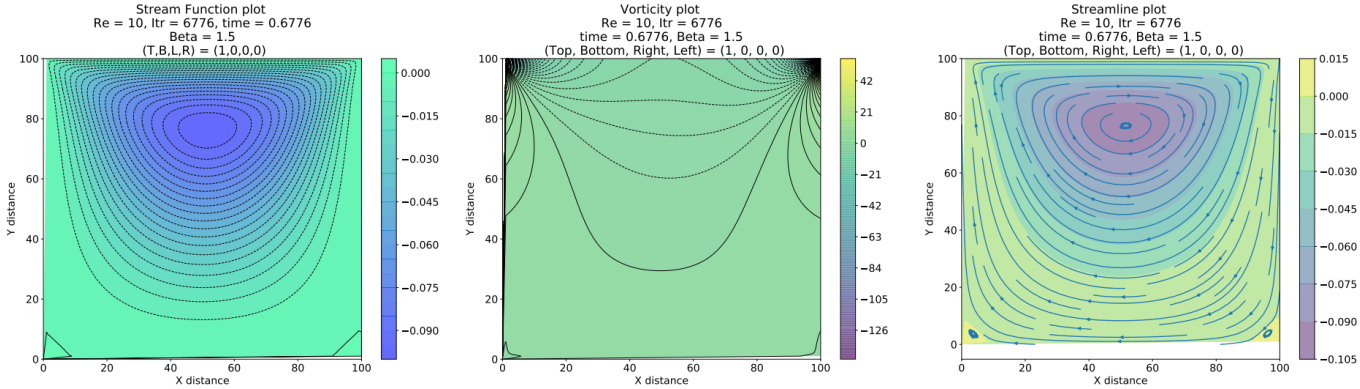

Figure 45: Scenario 1; Pressure, v-profile and, u-profile plots.

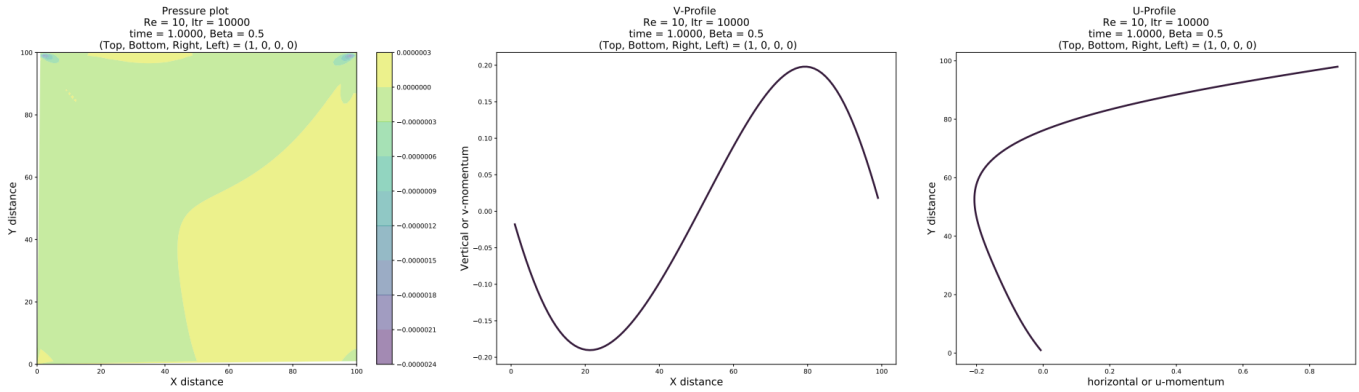

Figure 46: Scenario 14; Pressure, v-profile and, u-profile plots.

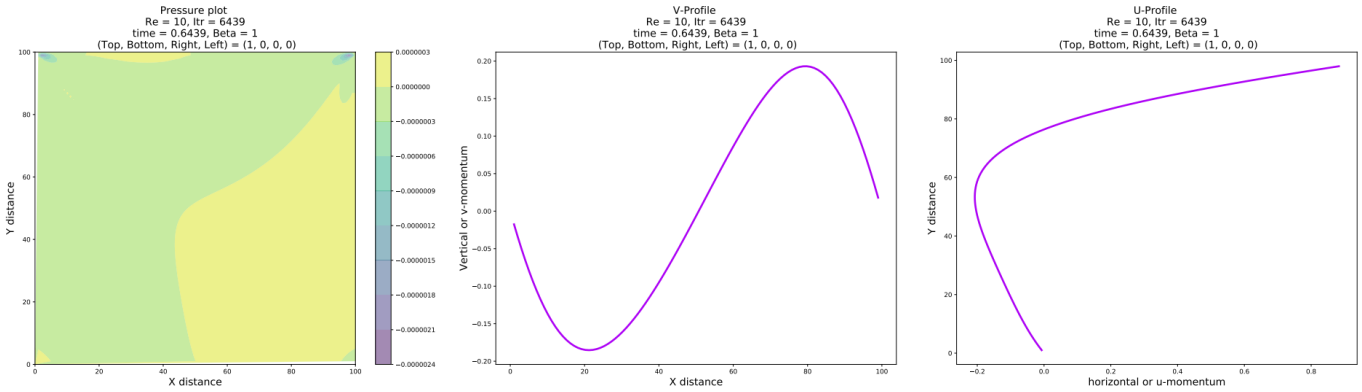

Figure 47: Scenario 13; Pressure, v-profile and, u-profile plots.

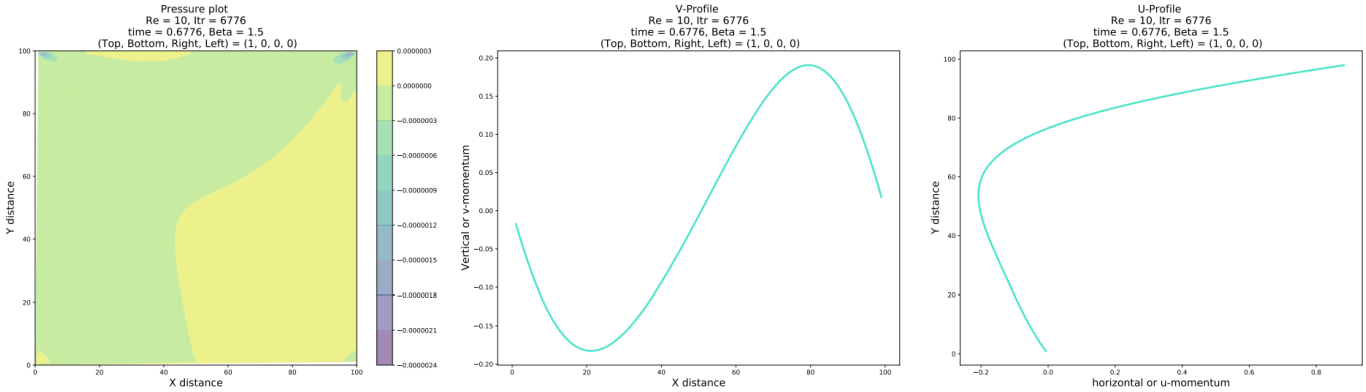

Figure 48: Scenario 1; Pressure, v-profile and, u-profile plots.

Scenario #	**Convergence**	**Iterations**	**Execution Time**	**Error <Threshold (1e-5)**
1	Yes	6776	8.9959 Minutes	9.99250e-06
13	Yes	6439	8.5565 Minutes	9.99864e-06
14	No	10000	12.7968 Minutes	3.68405e-05

3.9 Dual convergence scenario

In this scenario the Top and Left wall vortex's converge as one big vortex and the bottom vortex converges with the right. The simulation is aimed to observe how the fluid will interact in this scenario. Will it form vortex's or collapse into some unexpected state.

From the figures below we can see that, two mirror vortex's are formed where the Top and right converged to form the right vortex whereas the bottom left converted to form the left vortex. In this scenario the pressure around the diagonal should be minimal as it is a boundary where two repelling vortex's interact, from the Pressure plots we see that the diagonal has the minimal pressure recordings. The vortex generated for the Re=100 configuration is slightly offset from the center when compared to the Re=10 result.

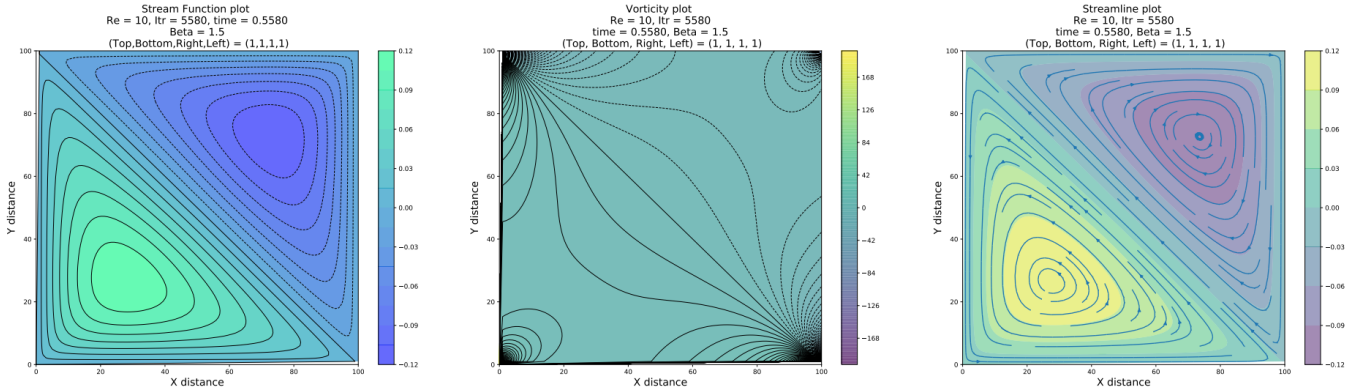

Figure 49: Scenario 15; Stream function, vorticity and, Stream line plots

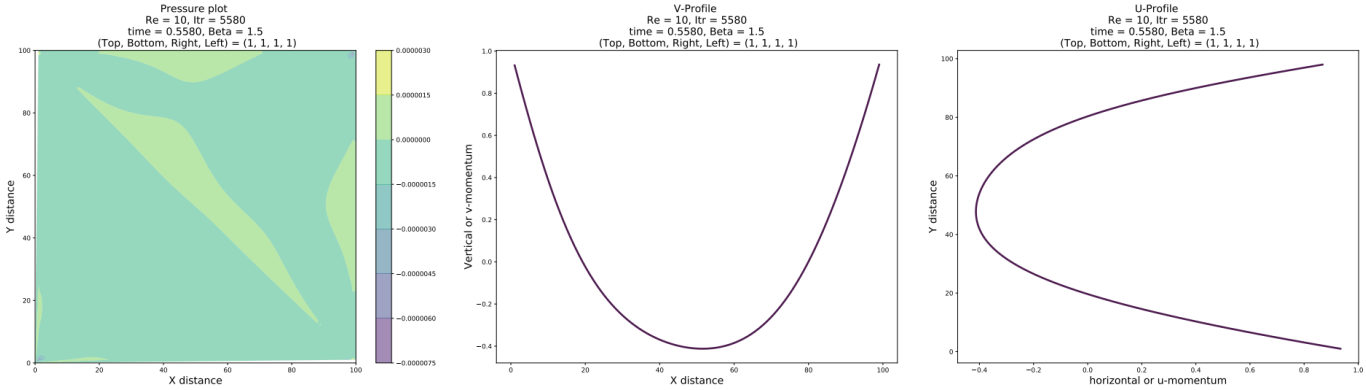

Figure 50: Scenario 15; Pressure, v-profile and, u-profile plots.

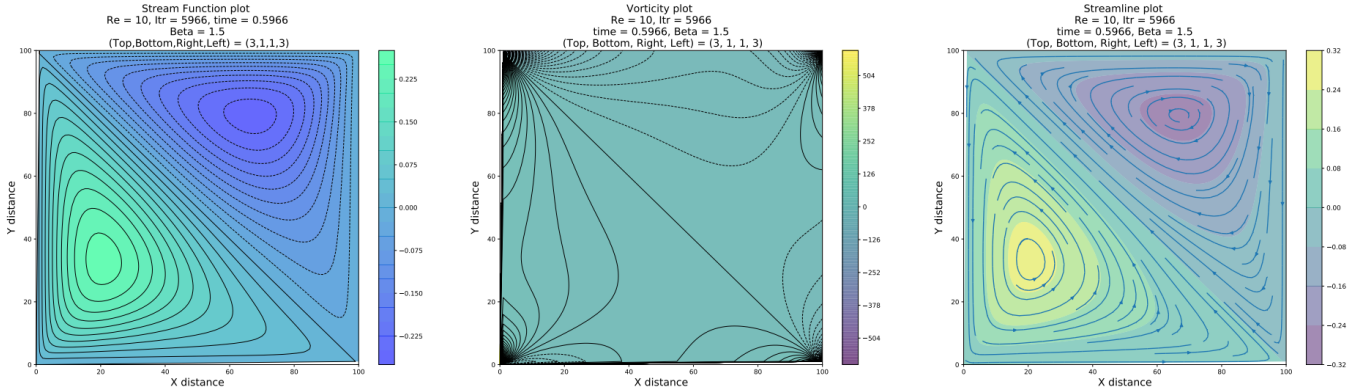

Figure 51: Scenario 16; Stream function, vorticity and, Stream line plots

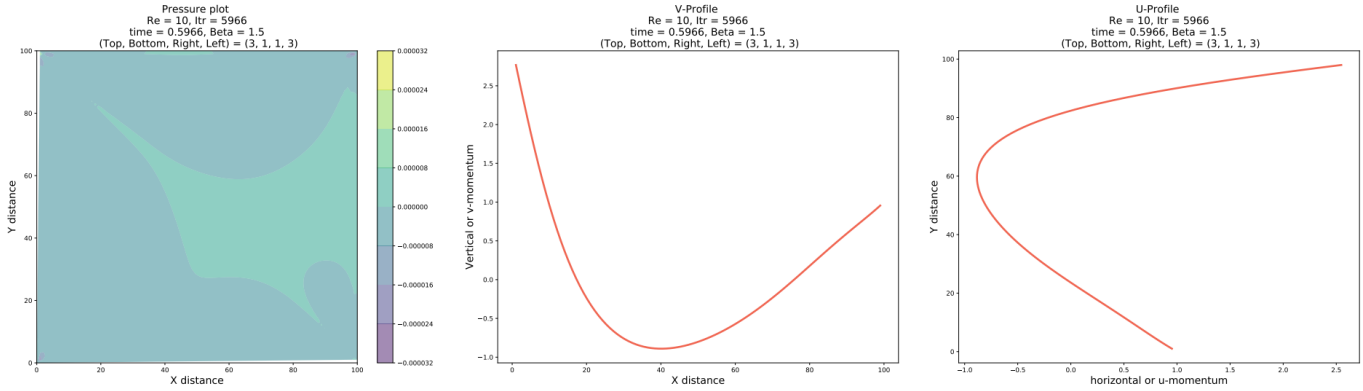

Figure 52: Scenario 16; Pressure, v-profile and, u-profile plots.

Scenario #	Convergence	Iterations	Execution Time	Error <Threshold (1e-5)
15	Yes	5580	7.2476 Minutes	9.98911e-06
16	Yes	5966	7.9394 Minutes	9.98979e-06

3.10 Complete divergence scenarios

In this scenario, the sides spin in opposite direction to each other so that there is no vortex merging events in the system. From the figure we can see that there are 4 distinct vortex's being created that repel each other but have achieved a stable convergence state without reaching the max allotted iterations.

The U-profile and the V-profile also matches the reference profiles as shown in Figure 10 and 11.

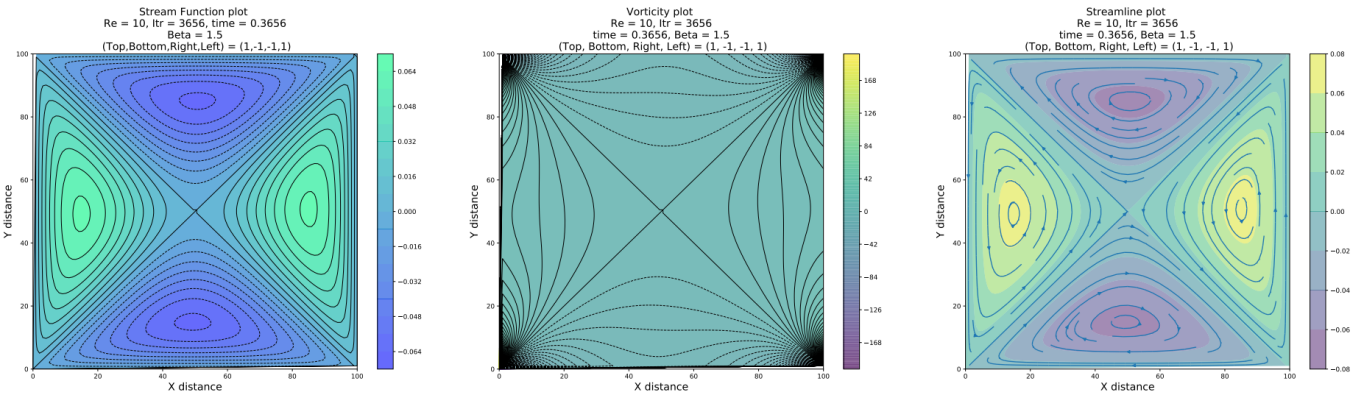

Figure 53: Scenario 17; Stream function, vorticity and, Stream line plots

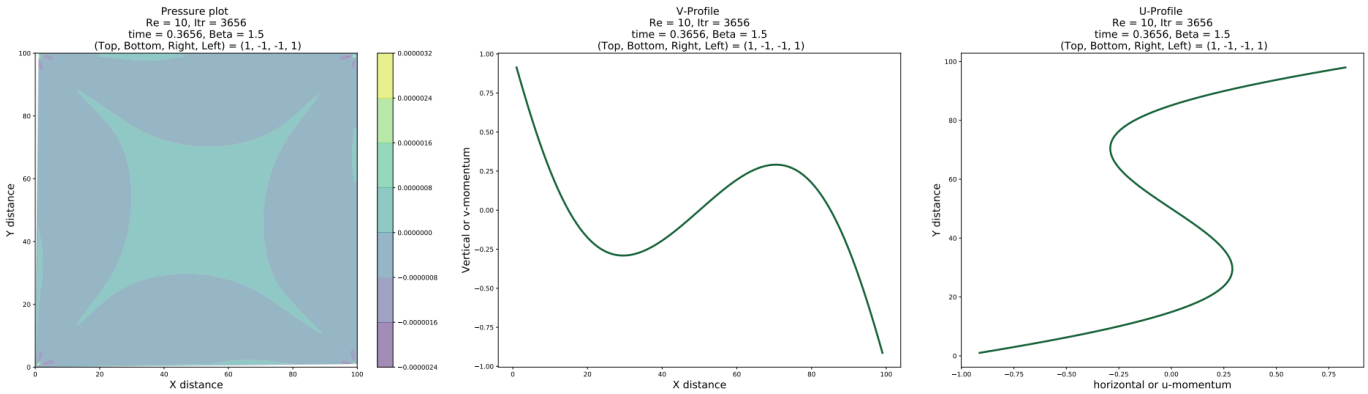

Figure 54: Scenario 17; Pressure, v-profile and, u-profile plots.

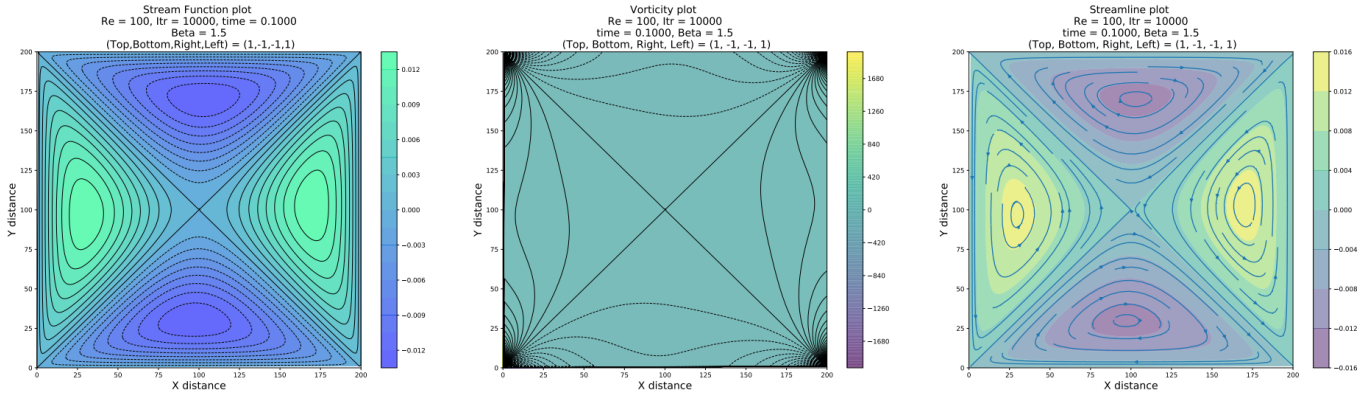

Figure 55: Scenario 18; Scenario 17; Stream function, vorticity and, Stream line plots

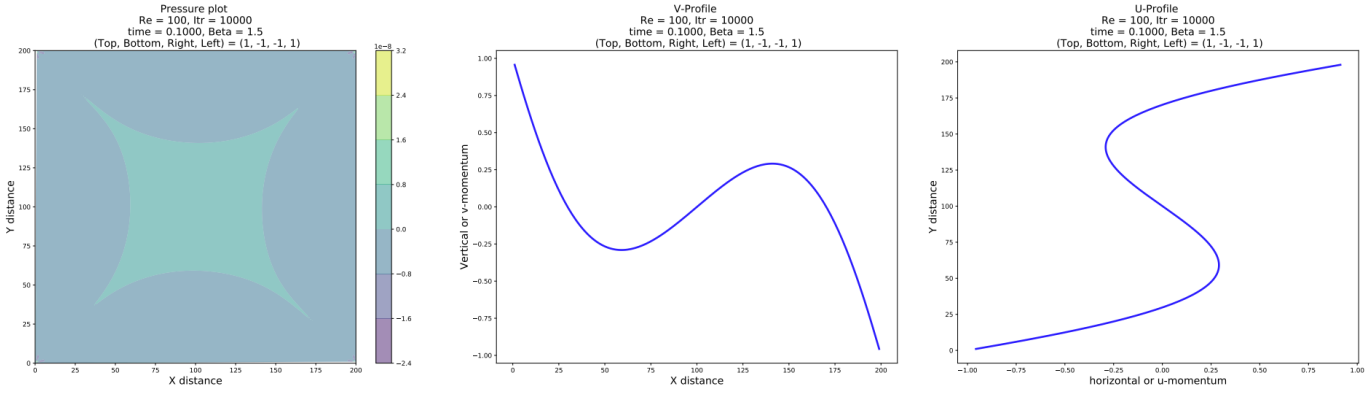

Figure 56: Scenario 18; Pressure, v-profile and, u-profile plots.

Let us see, if the stability of the system collapses if we introduce two sides with a higher momentum.

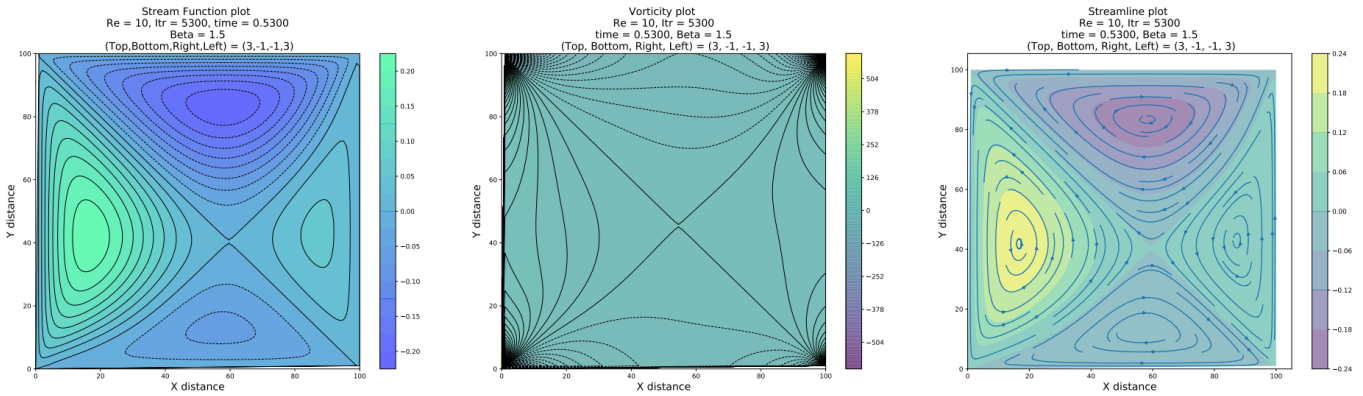

Figure 57: Scenario 21; Stream function, vorticity and, Stream line plots

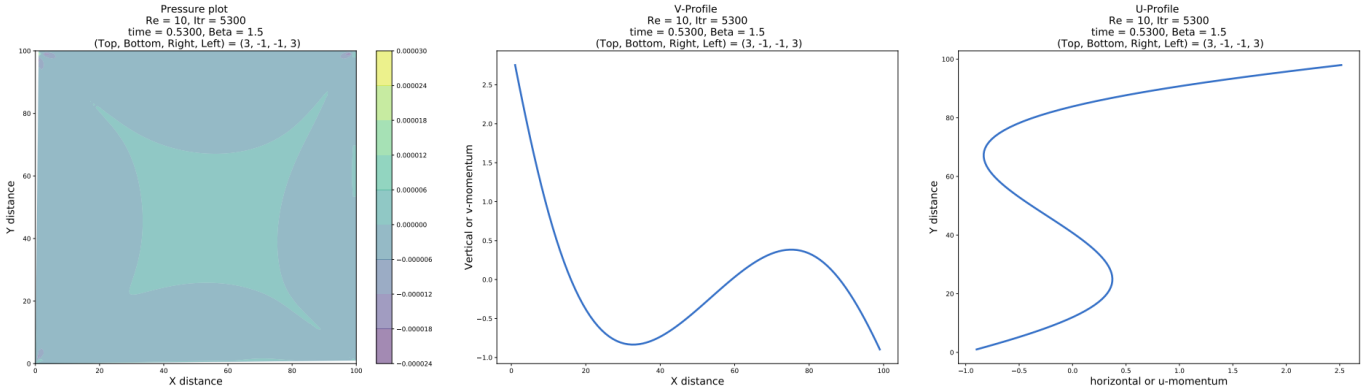

Figure 58: Scenario 21; Pressure, v-profile and, u-profile plots.

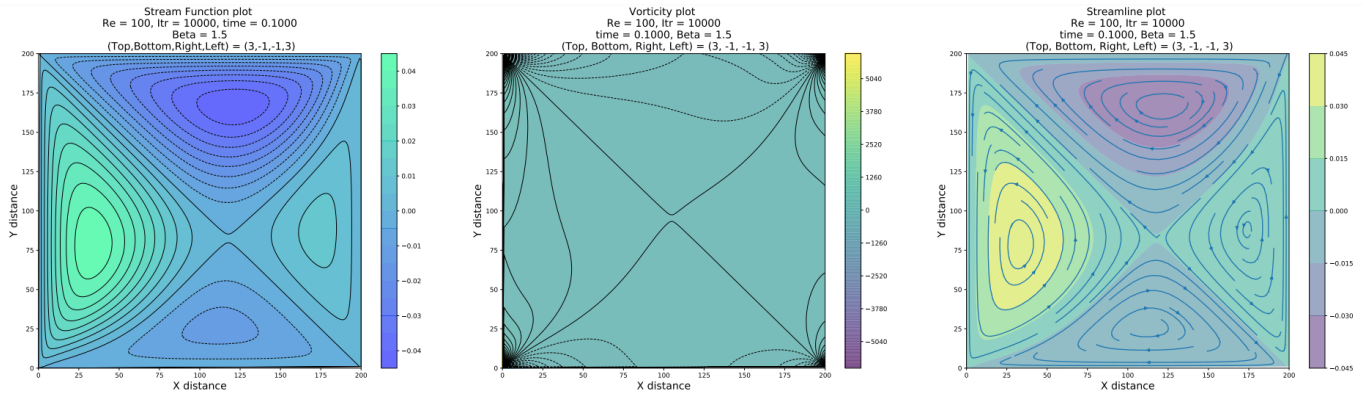

Figure 59: Scenario 22; Stream function, vorticity and, Stream line plots

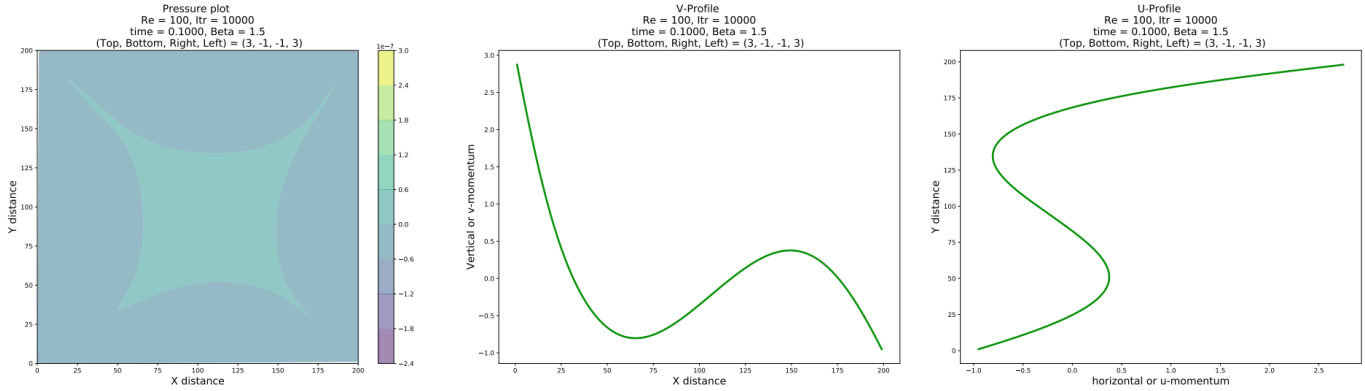

Figure 60: Scenario 22; Pressure, v-profile and, u-profile plots.

Scenario #	Convergence	Iterations	Execution Time	Error <Threshold (1e-5)
17	Yes	3656	4.8335 Minutes	9.99883e-06
18	No	10000	53.7806 Minutes	0.00011
21	Yes	5300	7.1402 Minutes	9.99170e-06
22	No	10000	53.4193 Minutes	0.00050

3.11 Complete convergence scenario

In this scenario, all the wall spin in the same direction. The expectation is that the fluid should converge to one big vortex that should appear symmetrical.

From the figure below, we can see that a perfectly symmetrical single vortex has been formed at the center of the cavity. And the vorticity plot, shows the direction of the vectors spinning in the same direction. The U-profile and V-profile graphs appear almost like a slope. Indicating that the velocities across X and Y planes were symmetrical.

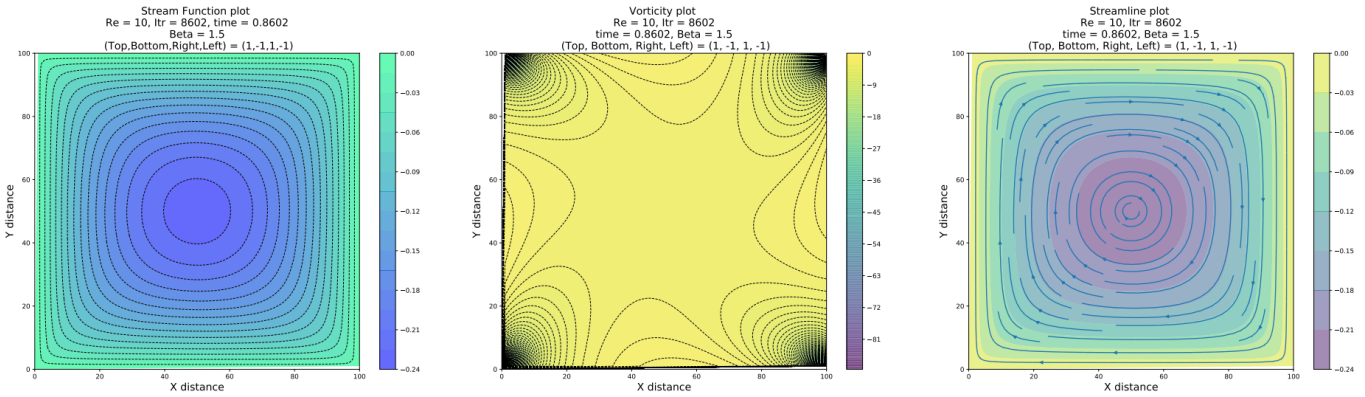

Figure 61: Scenario 20; Stream function, vorticity and, Stream line plots

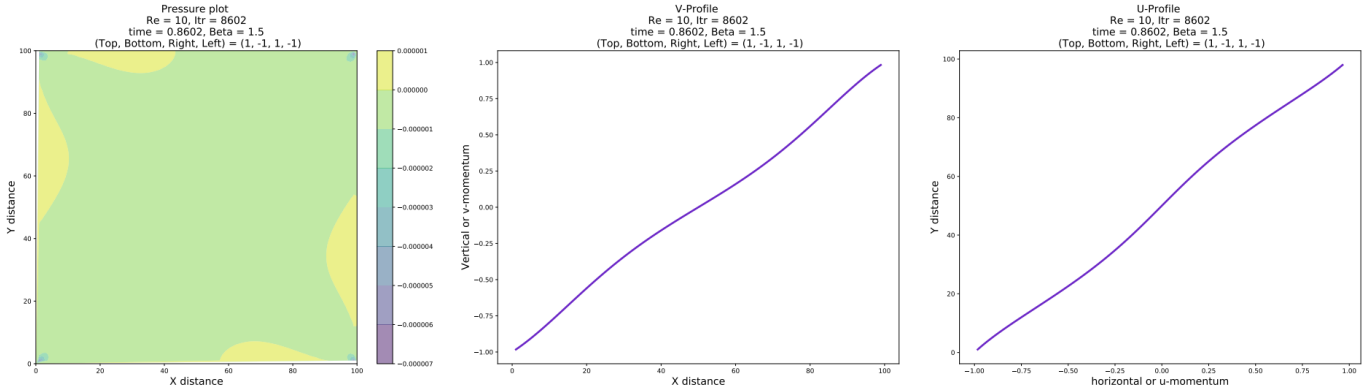

Figure 62: Scenario 20; Pressure, v-profile and, u-profile plots.

Scenario #	Convergence	Iterations	Execution Time	Error <Threshold (1e-5)
20	Yes	8602	11.2044 Minutes	9.99354e-06

3.12 Influence of Re on momentum

A noteworthy observation is that, at higher Reynolds number the momentum plots had better contour formation in-comparison to lower Reynolds value. The Figure 63 shows the scenario that led to this observation.

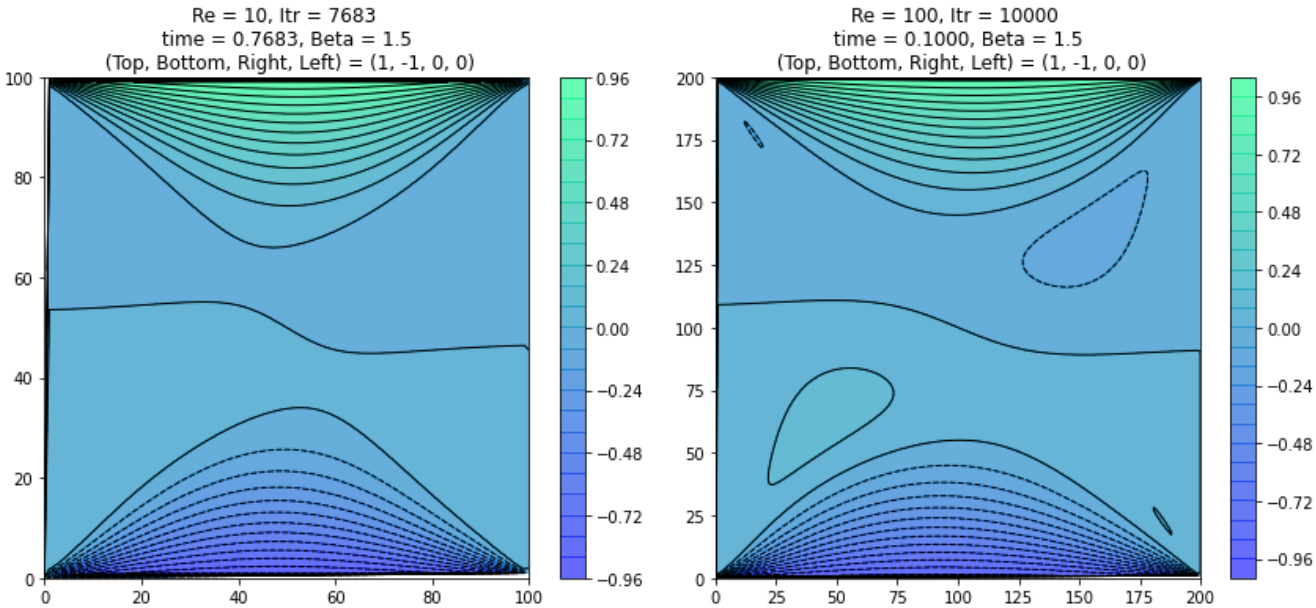

Figure 63: Pressure, v-profile and, u-profile plots.

4 Conclusion

The assignment attempts to simulate the Lid driven cavity problem using the Navier-stokes equations. The report discusses the fundamentals of the Lid driven cavity problem, to provide the reader with a basic understanding of the problem statement itself. Followed by the discussion of Navier-stokes equations and their discretized form. The Central differential approximation and forward

differential approximations are also discussed to provide the reader with additional information that were implicitly used in discretization step.

The report also lists the pseudo-code and the relevant code snippets that was used to realize this project. The project was developed as a framework, that exhibits re-usability and automation, the boiler plate tasks like plotting, storage, and orchestration is automatically handled by the code.

The result and discussion section contains an exhaustive list of various different scenarios that were tested and the observations were recorded. Some scenarios were explicitly tested for an expected state which has been pleasantly satisfied by the implementation. A notable result is the 4 moving wall scenario who's vortex centers along with their u-profile and v-profile were compared against reference data or benchmark data.

The report provides multiple scenarios that can be studied and/or observed to understand how fluids interact in an incompressible no slip condition. A limitation was that it is increasingly difficult to find the exact reference benchmark data for a specific scenario, nevertheless this report contains sufficient benchmark results to confidently say that a satisfactory implementation of the Lid driven cavity problem was achieved.

From the metrics captured, we see that the simulation converges for Re $= 10$ and fails to converge for Re $= 100$, on closer inspection we see that the tolerance error $1e-5$ might have been slightly too ambitious, as from the error's observed for Re $= 100$ scenarios are of the threshold $1e-3$, with a lower error threshold the Re $= 100$ scenarios will converge more often. This also hints that variability of the system will increase as the Reynolds number increases.

We also discussed the influence of Re over the momentum plot. A higher Re will result in more accurate plot.

It is also observed that the execution time for a higher Re value is significantly higher in-comparison to a lower value. The accuracy is certain scenarios were marginally better for Re 100 when compared to Re 10, but the gain cannot be justified with the 5x times longer execution time and failed to reach convergence.

I hope the reader has been given sufficient information to gain a basic understanding of the project and how Computational fluid dynamics works and has motivated them to pursue their own exploratory analysis on the same.

References

[1] C. A. Ardagna, M. Cremonini, S. D. C. di Vimercati, and P. Samarati. An obfuscation-based approach for protecting location privacy. *IEEE Transactions on Dependable and Secure Computing*, 8(1):13–27, 2009.

[2] C. Azwadi, A. Rajab, and A. Sofianuddin. Four-sided lid-driven cavity flow using time splitting method of adams-bashforth scheme. *International Journal of Automotive and Mechanical Engineering*, 9:1501, 2014.

[3] A. J. Chorin. Numerical solution of the navier-stokes equations. *Mathematics of computation*, 22(104):745–762, 1968.

[4] D. Harnik, Y. Ishai, and E. Kushilevitz. How many oblivious transfers are needed for secure multiparty computation? In *Annual International Cryptology Conference*, pages 284–302. Springer, 2007.

[5] Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer–efficiently. In *Annual international cryptology conference*, pages 572–591. Springer, 2008.

[6] S. Khorasanizade and J. M. Sousa. A detailed study of lid-driven cavity flow at moderate reynolds numbers using incompressible sph. *International Journal for Numerical Methods in Fluids*, 76(10):653–668, 2014.

[7] J. Kilian. Founding crytpography on oblivious transfer. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 20–31, 1988.

[8] C. H. Marchi, R. Suero, and L. K. Araki. The lid-driven square cavity flow: numerical solution with a 1024 x 1024 grid. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 31(3):186–198, 2009.

[9] M. O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptol. ePrint Arch.*, 2005(187), 2005.

[10] D. Wagner. *Advances in Cryptology-CRYPTO 2008: 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008, Proceedings*, volume 5157. Springer, 2008.

[11] Wikipedia contributors. Incompressible flow — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Incompressible_flow`, 2020. [Online; accessed 06-November-2020].

[12] Wikipedia contributors. No-slip condition — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/No-slip_condition`, 2020. [Online; accessed 06-November-2020].

[13] Wikipedia contributors. Stream function — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Stream_function`, 2020. [Online; accessed 06-November-2020].

[14] Wikipedia contributors. Successive over relaxation — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Successive_over-relaxation`, 2020. [Online; accessed 06-November-2020].

[15] Wikipedia contributors. Vortex — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Vortex`, 2020. [Online; accessed 06-November-2020].

[16] Wikipedia contributors. Vorticity — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Vorticity`, 2020. [Online; accessed 06-November-2020].

A Code snippets

A.1 Lid cavity class structure

```
1
2 class LidCavity:
3
4   def __init__(self, h, l, Re, err_tol, dt, end_time, Beta, max_itr, U, run=True,
5                top=1, bottom =0, left=0, right=0, dir_suffix="", disp_separate =
    False):
6     self.Re = Re;                                        # Reynolds number
7     self.h = h;                                          # stable 0.1/Re, grid step
8     self.l = l;                                          # stable, 10/Re, grid length
9     self.n = int(self.l/self.h);                         # Grid size
10    self.L = self.n;                                     # Grid size
11    self.err_tol = err_tol;                              # Error Tolereance
12    self.max_itr = max_itr;                              # Set Maximum iteration
13    self.Beta = Beta;                                    # Relaxation factor for SOR.
14    self.dt = dt;                                        # Time step
15    self.end_time = end_time;                            # End time; if required to
    terminate a loop using time; alt to max_itr
16    self.M = self.n;                                     # Grid M size
17    self.N = self.n;                                     # Grid N size
18    self.xi = np.linspace(0, self.L, int(self.M));       # X ticks
19    self.yi = np.linspace(0, self.L, int(self.N));       # Y ticks
20    self.A = np.zeros((self.M, self.N+1));               # Temporary array to set X
    and Y ticks
21    self.X = np.zeros((self.M,self.N));                  # X ticks accumulator
22    self.Y = np.zeros((self.M, self.N));                 # Y ticks accumulator
23    self.U = U;                                          # Magnitude at which the Lid
    is moved. Note: Keep it and use Top, Bottom, Left, Right parameters to further
     tweak different magnitudes.
24    self.top = top;                                      # Top lid control, 0 -> off,
    1-> move with U magnitude, n -> move with n * U magnitude where n = +/-
    1,2,3.....
25    self.bottom = bottom;                                # Bottom lid control, 0 ->
    off, 1-> move with U magnitude, n -> move with n * U magnitude where n = +/-
    1,2,3.....
26    self.left = left;                                    # Left lid control, 0 -> off,
     1-> move with U magnitude, n -> move with n * U magnitude where n = +/-
    1,2,3.....
27    self.right = right;                                  # Right lid control, 0 -> off
    , 1-> move with U magnitude, n -> move with n * U magnitude where n = +/-
    1,2,3.....
28
29    # Setup X and Y axis Co-ordinates
30    for i in range(1, self.M):
31      for j in range(1, self.N):
32        self.A[i, j+1] = self.yi[j];
33        self.A[i,1] = self.xi[i];
34
35    for i in range(1, self.M):
36      for j in range(1, self.N):
37        self.X[i, j] = self.A[i, 1];
38        self.Y[i, j] = self.A[1, j+1];
39
40    self.sf = np.zeros((self.M, self.N));          # Stream function acumulator
```

```
    self.w = np.zeros((self.M, self.N));            # Vorticity acumulator
    self.u = np.zeros((self.M, self.N));            # u-momemntum accumulator
    self.v = np.zeros((self.M, self.N));            # v-momentum accumulator
    # self.w_rhs = np.zeros((self.M, self.N));      # RHS of Vorticity, if
   required ???
    self.P = np.zeros((self.M, self.N));            # Pressure acumulator
    self.pn = np.zeros((self.M, self.N));           # Stores old pressure value
    self.rhs = np.zeros((self.M, self.N));          # RHS of pressure
    self.R = np.zeros((self.M, self.N));            # Check if needed
    self.h_sqr = self.h*self.h;                     # h*h for simplicity and less
   clutter in equations
    self.run = run;                                 # Flag indicating if the
   scenario should run or fetch results from storage.
    self.stable = False;                            # Convergence flag to initiate
    early termination
    self.itr = 0;                                   # Iteration counter to compare
    vs max_itr
    self.t = 0;                                     # Time variable
    self.old_sf = self.sf.copy();                   # Previous or n-1 step stream
   function values
    self.old_w = self.w.copy();                     # Previous or n-1 step
   Vorticity values
    self.old_u = self.u.copy();                     # Previous or n-1 step u-
   momentum values
    self.old_v = self.v.copy();                     # Previous or n-1 step v-
   momentum values
    self.exec_time_s = 0;                           # record the start time of
   scenario
    self.exec_time_e = 0;                           # record the end time of
   scenario
    self.exec_time = 0;                             # record the total Execution
   time of scenario
    self.dir_suffix = dir_suffix;                   # directory name modifier for
   result storage
    self.disp_separate = disp_separate;             # A flag that when true will
   display the scenario as separate in some plots
```

A.2 Stream function

```
def init_stream_function_vorticity(obj):
  for i in range(0, obj.n-1):
    for j in range(0, obj.n-1):
      obj.sf[i, j] = 0;
      obj.w[i,j]=0;
      if(j==obj.n-1):
        obj.sf[i, j] = 0;
        obj.w[i,j]=-2*obj.U/obj.h;


def stream_function(n, stream_func_acc, vorticity, h_sqr, Beta):
  for i in range(0, n):
    for j in range(0, n):
      if(j !=0 and j!= n-1 and i!=0 and i!=n-1):
        streamfunc_comp = stream_func_acc[i+1, j] +stream_func_acc[i-1, j] +
   stream_func_acc[i, j+1] +stream_func_acc[i, j-1]
        vorticity_comp = (h_sqr*vorticity[i,j])
```

```
        stream_func_acc[i, j]= Beta*0.25*(streamfunc_comp + vorticity_comp)+(1-Beta)*stream_func_acc[i,j];
      else:
        stream_func_acc[i, j] = 0;
  return stream_func_acc;
```

A.3 Vorticity function

```
def vorticity(n, stream_func_acc, vorticity, h, h_sqr, Re, dt, U, top, bottom, left, right):
  for i in range(0, n):
    for j in range(0, n):
      if(j != 0 and j != n-1 and i != 0 and i != n-1):
        first_comp = ((stream_func_acc[i, j+1]-stream_func_acc[i, j-1]) * ( vorticity[i+1, j] - vorticity[i-1, j]))/(4*h_sqr);
        second_comp = ((stream_func_acc[i+1, j]-stream_func_acc[i-1, j])*( vorticity[i,j+1]-vorticity[i,j-1]))/(4*h_sqr);
        third_comp = (vorticity[i+1, j]+vorticity[i-1, j]+vorticity[i, j+1]+ vorticity[i, j-1]-4*vorticity[i, j])/h_sqr;
        vorticity[i,j] = vorticity[i, j] + dt * (second_comp - first_comp + ( third_comp/Re));
      elif (j==0 and i!=0 and i!=n-1):
        vorticity[i,j]=2*(stream_func_acc[i,j]-stream_func_acc[i,j+1])/h_sqr + bottom*2*U/h; #Bottom
      elif ( j==n-1 and i!=0 and i!=n-1):
        vorticity[i,j]=2*(stream_func_acc[i,j]-stream_func_acc[i,j-1])/h_sqr - top*2*U/h; # Top
      elif ( i==0 and j!=0 and j!=n-1):
        vorticity[i,j]=2*(stream_func_acc[i,j]-stream_func_acc[i+1,j])/h_sqr + left*2*U/h; # Left
      elif (i==n-1 and j!=0 and j!=n-1):
        vorticity[i,j]=2*(stream_func_acc[i,j]-stream_func_acc[i-1,j])/h_sqr - right*2*U/h; #Right
      elif (i==0 and j==0):
        vorticity[i,j] = 0; # Corners
      elif  (i==n-1 and j==0):
        vorticity[i,j] = 0;
      elif  (i==0 and j == n-1):
        vorticity[i,j] = vorticity[i+1, j];
      elif (i == n-1 and j==n-1):
        vorticity[i, j] = vorticity[i-1, j];
  return vorticity;
```

A.4 Momentum function

```
def calculate_x_y_momentum(n, stream_func_acc, x_momentum, y_momentum, h):
  for i in range(0, n):
    for j in range(0, n):
      if (j!=0 and j != n-1 and i != 0 and i != n-1):
        x_momentum[i, j] = (stream_func_acc[i, j+1] - stream_func_acc[i, j-1]) /(2*h);
        y_momentum[i, j] = (stream_func_acc[i+1, j] - stream_func_acc[i-1, j]) /(2*h);
      elif (j == n-1):
```

```
        x_momentum[i, j] == 1;
        y_momentum[i, j] == 0;
      else:
        x_momentum[i, j] = 0;
        y_momentum[i, j] = 0;
  return (x_momentum, y_momentum);
```

A.5 Pressure function

```
def calculate_pressure(n, pressure, pressure_old, stream_func_acc, h, rhs):
  for i in range(0, n-1):
    for j in range(0, n-1):
      rhs[i, j] = (((stream_func_acc[i-1, j] - 2*stream_func_acc[i,j] +
   stream_func_acc[i+1, j])/(h*h) )) * ((stream_func_acc[i, j-1] - 2*
   stream_func_acc[i, j] +stream_func_acc[i, j+1])/ (h*h)) - (stream_func_acc[i
   +1, j+1] - stream_func_acc[i+1, j-1]-stream_func_acc[i-1, j+1]+stream_func_acc
   [i-1, j-1])/(4*(h*h));
      pressure[i, j] = (0.25*(pressure_old[i+1, j] + pressure_old[i-1, j] +
   pressure_old[i, j+1] + pressure_old[i, j-1]) - 0.5*((rhs[i, j] *(h*h * h*h))))
   ;
    pressure_old = pressure.copy();
  return (pressure, pressure_old);
```

A.6 Smart plot function

One function to generate a collated plot of the simulation.

```
def smart_plot(data):
  n_plots = np.shape(data)[0];
  #n_plots = 2;
  fig, ax = plt.subplots(nrows = n_plots*2, ncols=3);
  fig.set_figheight(17*n_plots);
  fig.set_figwidth(30);
  i = 0;
  plot = 0;
  while(i < n_plots*2):
    cs = ax[i][0].contourf(data[plot][0].X, data[plot][0].Y, data[plot][0].sf,
   20, cmap=plt.cm.winter, alpha=0.6)
    ax[i][0].contour(cs, colors='k', linewidths=1);
    ax[i][0].set_title("Stream Function plot \n Re = {}, Itr = {}, time = {:.4f}
   \n Beta = {} \n (T,B,L,R) = ({},{},{},{})".format(data[plot][0].Re,
                                                    data[plot][0].itr,
                                                    data[plot][0].t,
                                                    data[plot][0].Beta,
                                                    data[plot][0].top,
                                                    data[plot][0].bottom,
                                                    data[plot][0].left,
                                                    data[plot][0].right));
```

```
20      ax[i][0].set_ylabel('Y distance')
21      ax[i][0].set_xlabel('X distance')
22      plt.colorbar(cs, ax=ax[i, 0]);
23
24      cs = ax[i][1].contourf(data[plot][0].X, data[plot][0].Y, data[plot][0].w,
        200, cmap=cm.viridis, alpha=0.6)
25      ax[i][1].contour(cs, colors='k', linewidths=1);
26      ax[i][1].set_title("Vorticity plot \n Re = {}, Itr = {} \n time = {:.4f},
        Beta = {} \n (Top, Bottom, Right, Left) = ({}, {}, {}, {})"
27              .format(data[plot][0].Re, data[plot][0].itr, data[plot][0].t, data[
        plot][0].Beta, data[plot][0].top, data[plot][0].bottom, data[plot][0].right,
        data[plot][0].left));
28      ax[i][1].set_ylabel('Y distance')
29      ax[i][1].set_xlabel('X distance')
30      plt.colorbar(cs, ax=ax[i, 1]);
31
32      dsize = np.shape(data[plot][0].u)[0];
33      x = np.linspace(0, dsize, dsize);
34      y = np.linspace(0, dsize, dsize);
35      cs = ax[i][2].contourf(data[plot][0].X, data[plot][0].Y, data[plot][0].sf,
        alpha=0.5, cmap=cm.viridis);
36      ax[i][2].streamplot(x, y, np.transpose(data[plot][0].u), np.transpose(-data[
        plot][0].v));
37      ax[i][2].set_title("Streamline plot \n Re = {}, Itr = {} \n time = {:.4f},
        Beta = {} \n (Top, Bottom, Right, Left) = ({}, {}, {}, {})"
38              .format(data[plot][0].Re, data[plot][0].itr, data[plot][0].t, data[
        plot][0].Beta, data[plot][0].top, data[plot][0].bottom, data[plot][0].right,
        data[plot][0].left));
39      ax[i][2].set_ylabel('Y distance')
40      ax[i][2].set_xlabel('X distance')
41      plt.colorbar(cs, ax=ax[i, 2]);
42
43
44      cs = ax[i+1][0].contourf(data[plot][0].X, data[plot][0].Y, data[plot][0].P,
        cmap=cm.viridis, alpha=0.5)
45      ax[i+1][0].set_title("Pressure plot \n Re = {}, Itr = {} \n time = {:.4f},
        Beta = {} \n (Top, Bottom, Right, Left) = ({}, {}, {}, {})"
46              .format(data[plot][0].Re, data[plot][0].itr, data[plot][0].t, data[
        plot][0].Beta, data[plot][0].top, data[plot][0].bottom, data[plot][0].right,
        data[plot][0].left));
47      ax[i+1][0].set_ylabel('Y distance')
48      ax[i+1][0].set_xlabel('X distance')
49      plt.colorbar(cs, ax=ax[i+1, 0]);
50
51
52      ax[i+1][1].plot(data[plot][0].X[1:int(data[plot][0].n-1),int(data[plot][0].n
        /2)],
53                      data[plot][0].v[1:int(data[plot][0].n-1),int(data[plot][0].
        n/2)],
54                      colors[plot%number_of_colors], label="Re = {}, Beta = {}, (
        T,B,R,L) = ({},{},{},{})".format(data[plot][0].Re, data[plot][0].Beta, data[
        plot][0].top, data[plot][0].bottom,
55
                                         data[plot][0].right, data[plot][0].left));
56      ax[i+1][1].set_title("V-Profile \n Re = {}, Itr = {} \n time = {:.4f}, Beta =
         {} \n (Top, Bottom, Right, Left) = ({}, {}, {}, {})"
57              .format(data[plot][0].Re, data[plot][0].itr, data[plot][0].t, data[
```

```
    plot][0].Beta, data[plot][0].top, data[plot][0].bottom, data[plot][0].right,
    data[plot][0].left));
58     #ax[i][j].legend();
59     ax[i+1][1].set_ylabel('Vertical or v-momentum')
60     ax[i+1][1].set_xlabel('X distance')
61
62
63     ax[i+1][2].plot(  data[plot][0].u[int(data[plot][0].n/2)][1:-2],
64                     data[plot][0].Y[int(data[plot][0].n/2)][1:-2],
65                       linewidth=3, color = colors[plot%number_of_colors],
66                       label="Re = {}, Beta = {}, (T,B,R,L) = ({},{},{},{})".
    format(data[plot][0].Re, data[plot][0].Beta, data[plot][0].top, data[plot][0].
    bottom, data[plot][0].right, data[plot][0].left));
67     ax[i+1][2].set_title("U-Profile \n Re = {}, Itr = {} \n time = {:.4f}, Beta =
     {} \n (Top, Bottom, Right, Left) = ({}, {}, {}, {})"
68              .format(data[plot][0].Re, data[plot][0].itr, data[plot][0].t, data[
    plot][0].Beta, data[plot][0].top, data[plot][0].bottom, data[plot][0].right,
    data[plot][0].left));
69     # ax[i+1][2].legend();
70     ax[i+1][2].set_xlabel('horizontal or u-momentum')
71     ax[i+1][2].set_ylabel('Y distance')
72     i = i+2;
73     plot = plot + 1;
74   return fig;
75
76
77 ##### FUNCTION CALL ######
78 fig = smart_plot(scenarios);
```

A.7 Captured Metrics

Scenario #	Convergence	Iterations	Execution Time	Error <Threshold (1e-5)
1	Yes	6776	8.9959 Minutes	9.99250e-06
2	No	10000	52.4460 Minutes	0.00040
3	Yes	5230	6.7916 Minutes	9.99312e-06
4	No	10000	52.3923 Minutes	0.00034
5	Yes	7683	9.9977 Minutes	9.99277e-06
6	No	10000	52.4148 Minutes	0.00057
7	Yes	7747	10.0208 Minutes	9.99890e-06
8	No	10000	52.9004 Minutes	0.00125
9	Yes	8723	11.3908 Minutes	9.99435e-06
10	No	10000	52.3150 Minutes	0.00161
11	Yes	7692	9.8463 Minute	9.99273e-06
12	No	10000	52.6329 Minutes	0.00069
13	Yes	6439	8.5565 Minutes	9.99864e-06
14	No	10000	12.7968 Minutes	3.68405e-05
15	Yes	5580	7.2476 Minutes	9.98911e-06
16	Yes	5966	7.9394 Minutes	9.98979e-06
17	Yes	3656	4.8335 Minutes	9.99883e-06
18	No	10000	53.7806 Minutes	0.00011
19	No	10000	51.8109 Minutes	0.00031
20	Yes	8602	11.2044 Minutes	9.99354e-06
21	Yes	5300	7.1402 Minutes	9.99170e-06
22	No	10000	53.4193 Minutes	0.00050

A.8 GitHub

https://github.com/Swastik-RUG/ComputationalPhysics