2D Incompressible Flow in Lid-Driven Cavity

MAE471 – Computational Fluid Dynamics
Final Project, Spring 2014

By: **Andrej Simeunovic**

Table of Contents

1.0 Problem Statement

The goal of this project is to solve the two-dimensional lid-driven cavity problem. The problem geometry is a unit sized square box with no-slip, non-permeable walls on all sides. All sides except for the top are at rest. The top wall moves in the *x*-direction with speed *U*. Figure 1 summarizes the problem geometry and velocity boundary conditions.

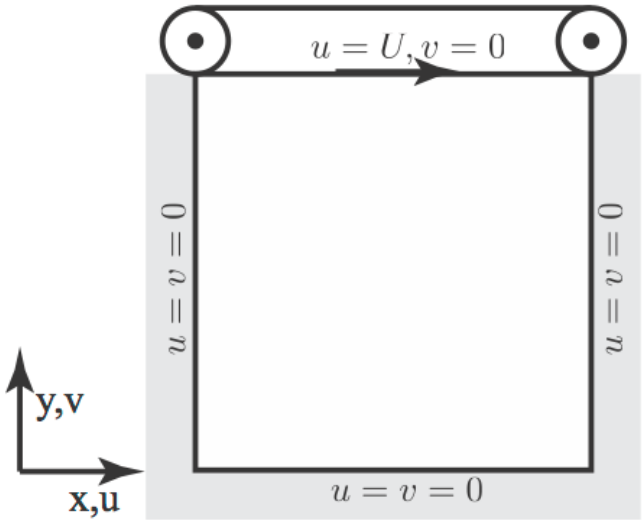

Figure 1: Lid Driven Cavity. Velocity boundary conditions are specified.

The Reynolds number uniquely determines the steady state of the flow:

$$Re = \frac{UL}{\upsilon}$$

where $L = 1$ is the length and height of the cavity and ν is the kinematic viscosity of the fluid inside the cavity. The flow is governed by the incompressible, 2D Navier-Stokes equations:

$$\begin{aligned} \nabla \cdot \boldsymbol{v} &= 0 \\ \frac{\partial \boldsymbol{v}}{\partial t} + \nabla \cdot (\boldsymbol{v}\boldsymbol{v}) &= -\nabla p + \frac{1}{Re}\nabla^2 \boldsymbol{v} \end{aligned}$$

This project involves 4 main tasks, detailed as follows:

A. Task 1: Determine steady state solution for $Re = 100$ using ANSYS Fluent.
B. Task 2: Determine steady state solution for $Re = 100$ using Fractional Step Method on a Staggered Grid.
C. Task 3: Describe numerical schemes and index notations used to solve Task 2.
D. Task 4: Include all codes used for this project.

These fours tasks are described in the following sections of the report. The tasks are also clearly identified in their respective sections.

2.0 ANSYS Fluent

The goal of this task, Task 1, was to determine the steady state solution for $Re = 100$ using ANSYS Fluent.

2.1 Set-Up

The lid-driven cavity geometry was built in the geometry modeler portion of ANSYS Fluent. The geometry was made to match the given dimensions outlined in the problem statement. The kinematic viscosity was set to $\nu = 0.01$ with density $\rho = 1$. Combined with the given Reynolds number, this corresponded to a lid velocity $U = 1$. Boundary conditions were set to match the given conditions from the problem statement.

2.2 Mesh

To ensure an accurate solution, several refined meshes were used to solve the problem. The results from the meshes were compared to each other and to Ghia (1982) to determine accuracy. The problem was first solved on the coarsest mesh, a 40 x 40 mesh. The results were compared to Ghia. Once they were satisfactory and clearly approaching the Ghia solutions, the entire Fluent module was copied. The mesh resolution was then refined in the new module and the problem was solved again. This was repeated a third time to ensure optimal accuracy. The medium and fine mesh resolutions were 80 x 80 and 160 x 160, respectively. Figure 2 presents the ANSYS Fluent workbench view for the different meshes. Figure 3 through Figure 5 show the actual meshes as viewed in the Mesh portion of ANSYS Fluent. Anomalies in the shape of the mesh cells are due to the built-in algorithms Fluent uses to mesh the geometry and did not have a negative effect on the final solutions.

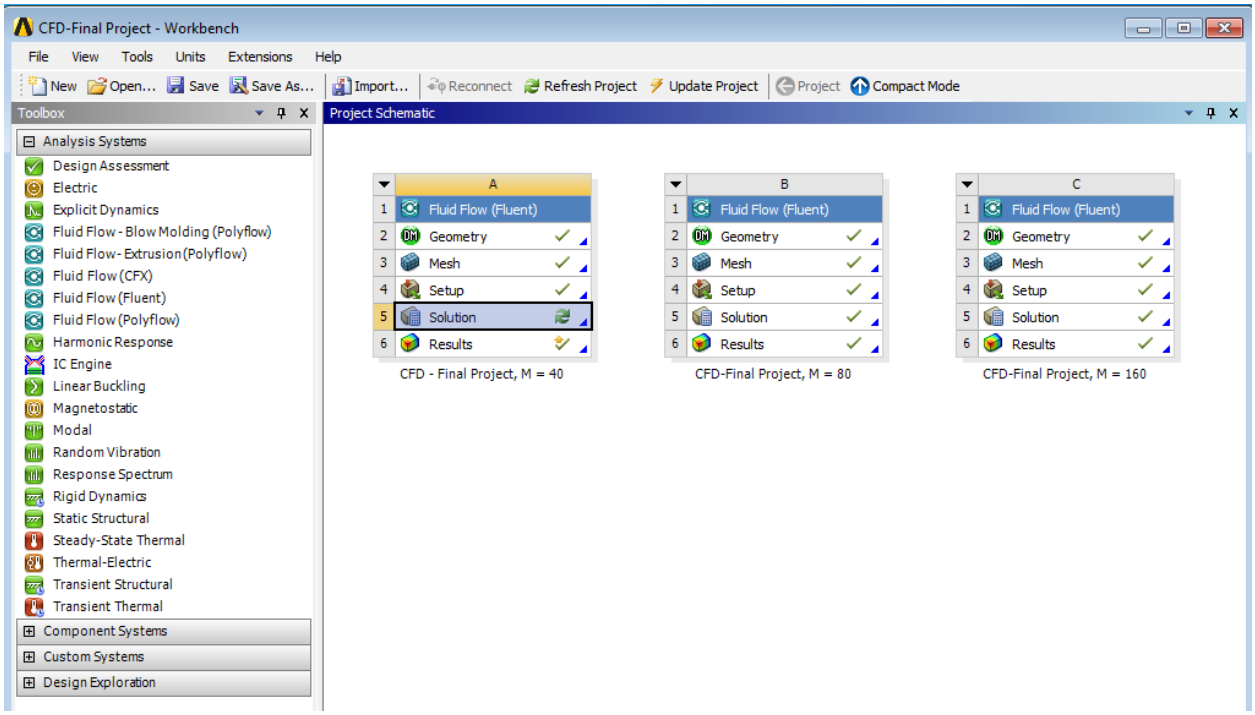

Figure 2: Mesh Refinement Set-up in ANSYS Fluent.

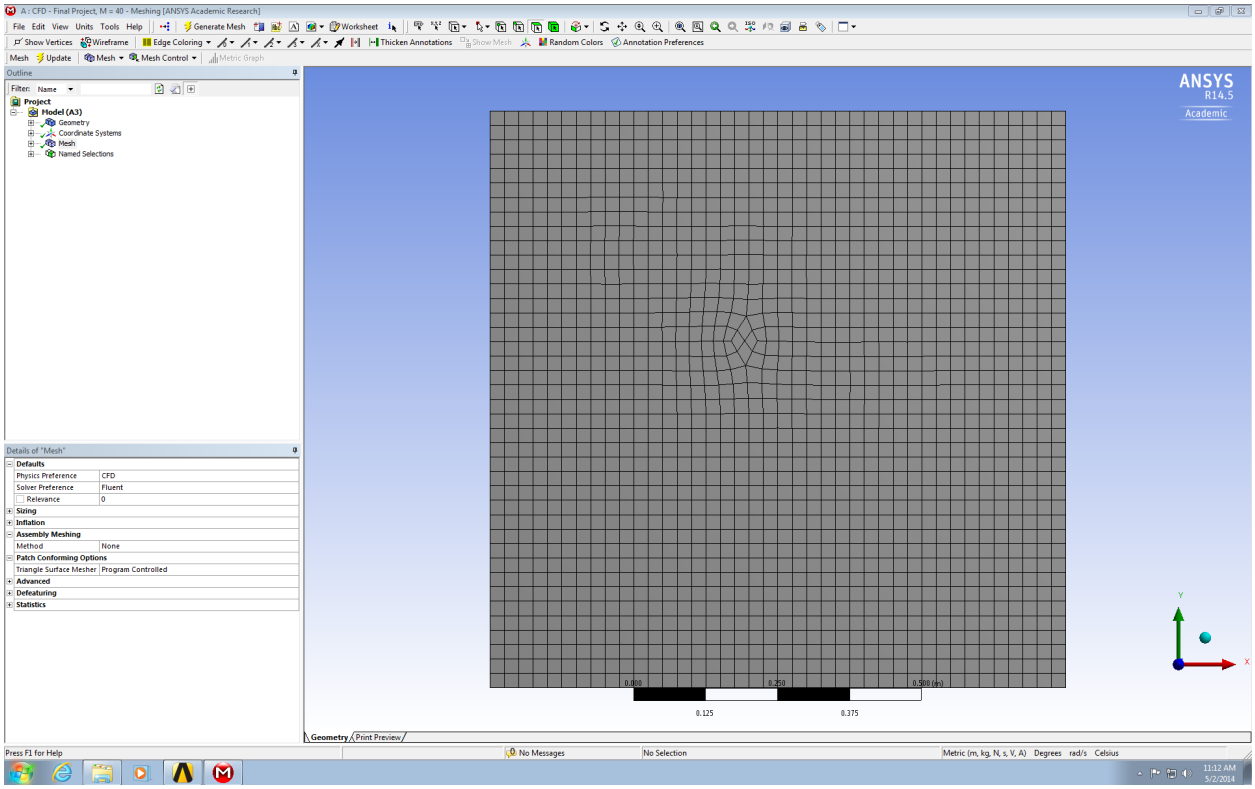

Figure 3: Coarse mesh view in Fluent. Mesh is 40 x 40 resolution.

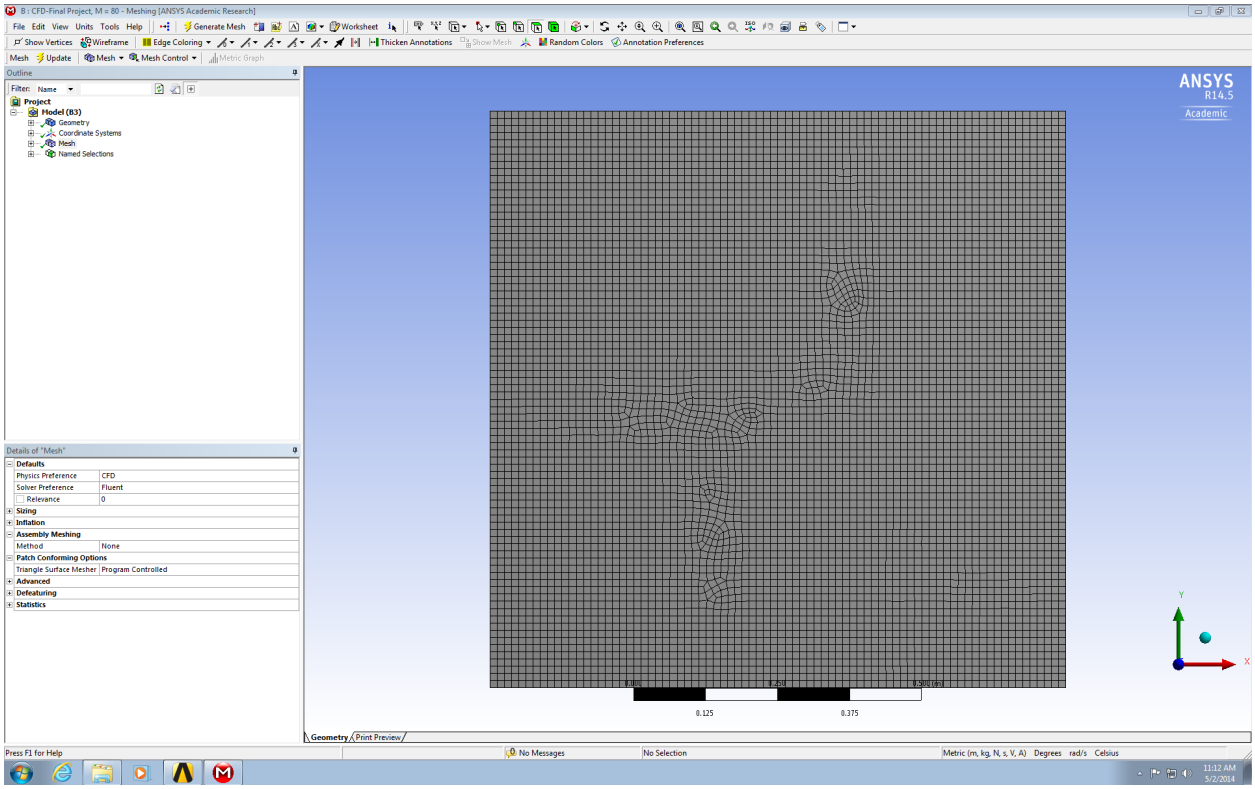

Figure 4: Medium mesh view in Fluent. Mesh is 80 x 80 resolution.

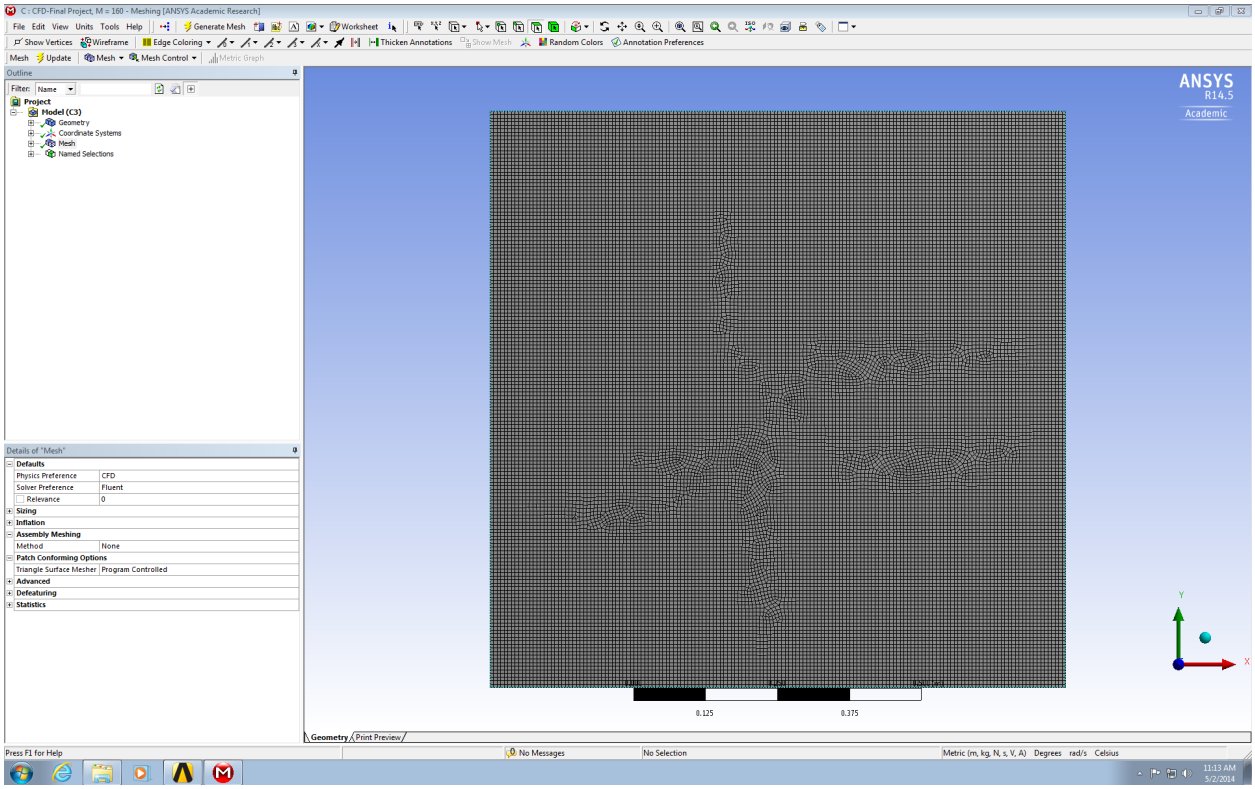

Figure 5: Fine mesh view in Fluent. Mesh is 160 x 160 resolution.

2.3 Numerical Schemes

Built-in Fluent solvers were used for the numerical schemes to solve the problem. The major pressure-velocity coupling solver was SIMPLE, which stands for Semi-Implicit Method for Pressure-Linked Equations. The SIMPLE algorithm is similar to the Fractional Step Method. It follows essentially the same 3 major steps:

A. Approximation of velocity field is obtained from solving the momentum equation.
B. The pressure equation is solved in order to obtain a new pressure distribution.
C. Velocities are corrected and a new set of conservative fluxes is calculated.

Differences between SIMPLE and FSM lie in the pressure calculations and corrections. Additionally ANSYS utilizes over and under relaxation factors as necessary to further improve the computational time, convergence, and accuracy of the solution. Spatial discretization methods are generally second-order and specific to the term being calculated. The numerical scheme set-up, as seen in the Fluent software, is presented in Figure 6.

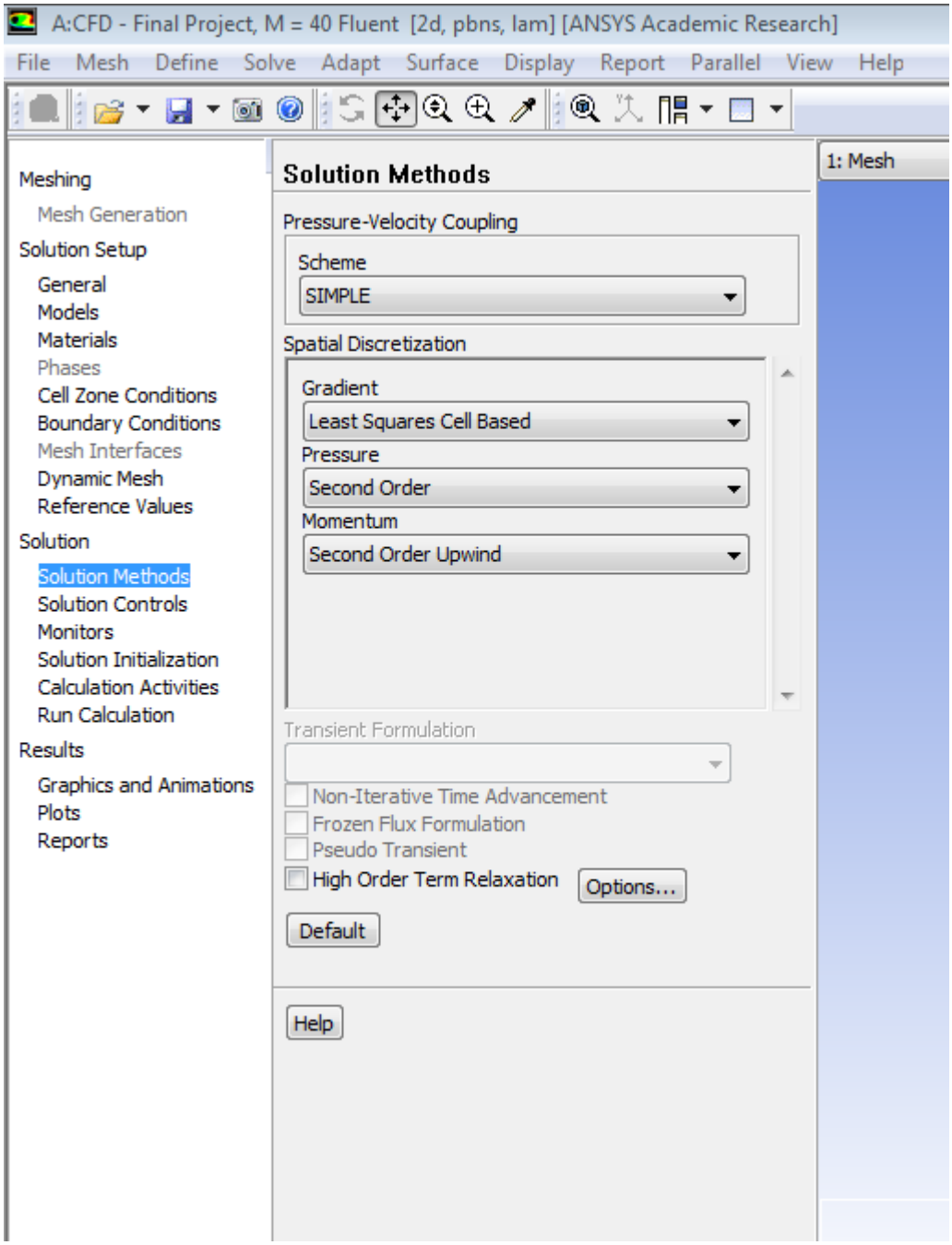

Figure 6: ANSYS Fluent Numerical Schemes.

2.4 Convergence and Accuracy

Convergence of the solutions was determined using Fluent's built-in convergence calculation algorithms. The convergence criteria were set to 10^{-7}, with a maximum number of iterations of 10,000. For all mesh resolution simulations, the maximum number of iterations was never reached and all solutions converged to the desired criteria. This implies a very good converged solution for the Fluent model. To further verify the accuracy of the final solutions, the percent differences between the most accurate Fluent results and the results found by Ghia were

determined. The largest percent difference for both solutions was determined. The results of this accuracy analysis are presented in Table 1.

Table 1: Accuracy analysis of Fluent and Ghia velocity solutions.

Compared Solutions	Maximum Percent Difference Between Ghia and Fluent
u-velocity	0.91%
v-velocity	1.02%

As seen in the table, the largest percent difference between the two results for both solutions was very small, further implying a very accurate solution.

2.5 Stream Function

Once an accurate solution of the lid-driven cavity problem was found using ANSYS Fluent, the streamline (Ψ) contours of the solution were plotted. The contours were plotted for the values of Ψ used by Ghia (Table 3 of Ghia paper) so that the contour plot could be compared to the Ghia results. It should be noted that the stream function is defined only up to a constant and depends on the boundary conditions used. Since Fluent and Ghia use different boundary conditions for plotting, the streamline contour plots look different. The solution itself is correct; it's the manner in which Fluent plots streamline contours that makes the plots look incorrect.

Fluent defaults to plotting all positive values for streamline contours, whereas Ghia uses negative values. To plot the true Ghia values in Fluent, the built-in offset that Fluent adds to produce all positive values was determined. This was done by finding the difference between the maximum stream function values in Fluent and Ghia. Once the offset was determined, it was added to the Ghia values to determine the Fluent equivalent of those contour values. The new contours were manually created in Fluent as iso-surfaces. These values were then plotted to create a streamline contour plot that more closely matched the Ghia results. Figure 7 presents the changes to the stream function contour plots following the use of the Ghia offset.

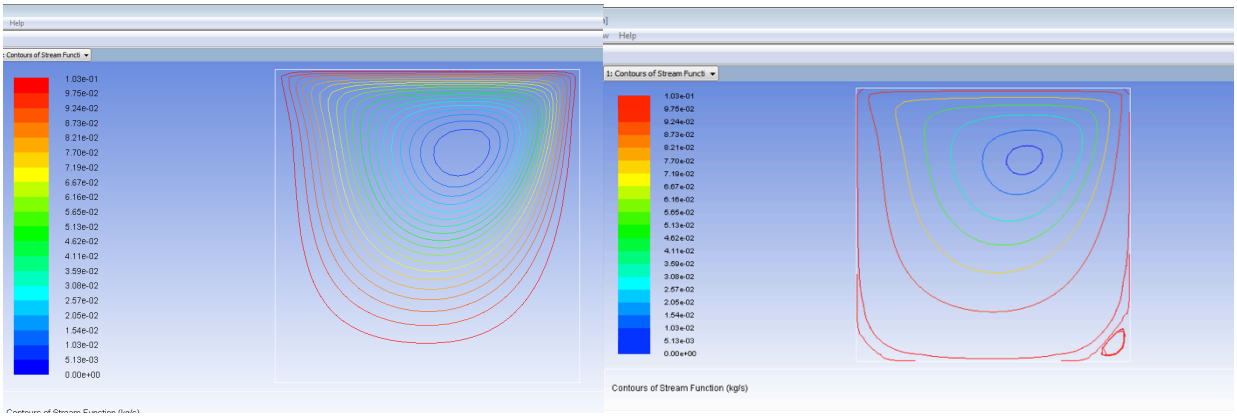

Figure 7: Change in stream function contour plot with use of Ghia offset.

The Fluent offset overrides the bottom corner vortices and the contours near the bottom of the cavity. The presented scale smears away the resolution necessary to see the fine flow details. By using the Fluent offset to manually plot the same stream function values as Ghia, a highly similar

plot may is found. The comparison of the Fluent and Ghia stream function contour plots is presented in Figure 8.

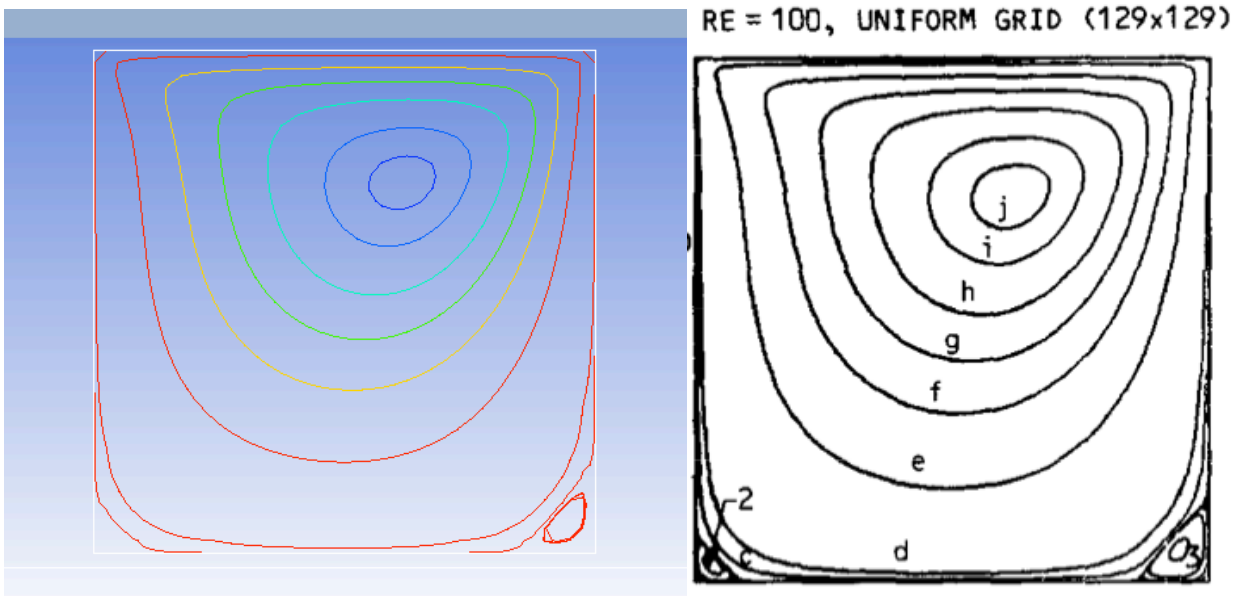

Figure 8: Comparison of Fluent and Ghia stream function contour plots.

Clearly the Fluent contour plot is highly similar to the Ghia plot. However, the left corner vortex and the lowest contour, closest to the bottom of the cavity, are not fully developed. This is again due to the display properties of the Fluent contour plotter. While the main offset brings back the resolution necessary to see most of the desired contour, the missing features are still "hidden" away in the fine details of the plot. There are methods to further tweak the Fluent display properties to display the missing contour features. Many of these methods were tried, but to no avail. However, the final solution was very closely matched the Ghia results, as seen in the velocity plots in a later section. These results, coupled with the lack of available time to further troubleshoot the Fluent display settings, is the reason a higher accuracy stream function contour plot was not found.

2.6 Vorticity

Once an accurate solution of the lid-driven cavity problem was found using ANSYS Fluent, the vorticity (ω) contours of the solution were plotted. The contours were plotted for the values of ω used by Ghia (Table 4 of Ghia paper) so that the contour plot could be compared to the Ghia results. Similar to the stream function formulation, Fluent does not properly display the vorticity contours and requires an offset to correctly display the Ghia vorticity contour values. This offset was determined in the same manner as stream function, but with the respective Ghia and Fluent vorticity values. As in the stream function plotting, the offset was added to the Ghia values in order to produce the desired vorticity contours in Fluent. Figure 9 presents in the vorticity contours produced by the Fluent with and without the offset.

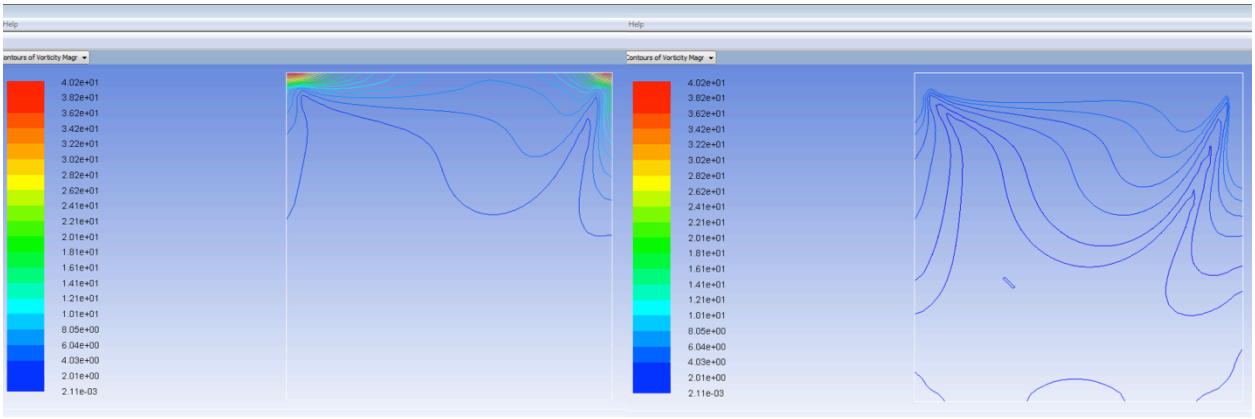

Figure 9: Change in vorticity contour plot with use of Ghia offset.

As in the stream function contour plot, the original Fluent plot did not have high enough detail to present most of the vorticity contours that Ghia presents. Manually plotting the desired values shows that the desired vorticity contours are all within the lowest color of the original Fluent colorbar scale. However, by plotting these values the vorticity contours seen by Ghia are clearly seen in the Fluent plot. Figure 10 presents a comparison of the Fluent (with offset) and the Ghia vorticity plots.

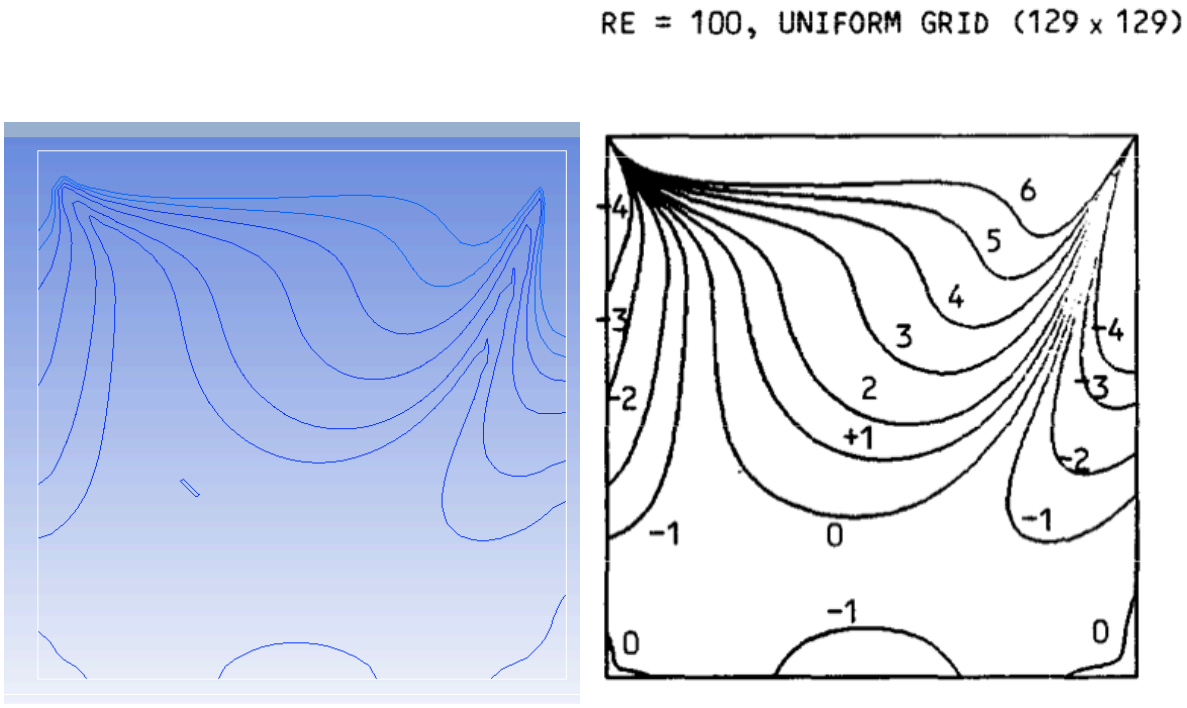

Figure 10: Comparison of Fluent and Ghia vorticity contour plots.

Clearly the Fluent contour plot is highly similar to the Ghia plot. However, the zero vorticity contour, near the middle of the cavity, is not fully developed. This is again due to the display properties of the Fluent contour plotter. While the main offset brings back the resolution necessary to see most of the desired contours, the missing features are still "hidden" away in the fine details of the plot. This hiding of features can be seen more acutely in the vorticity plot than

the stream function plot. The zero vorticity contour is there – albeit only a small part of it – but the full contour will not display due to the scale used by the Fluent plotter.

There are methods to further tweak the Fluent display properties to display the missing contour features. Many of these methods were tried, but to no avail. However, the final solution was very closely matched the Ghia results, as seen in the velocity plots in a later section. These results, coupled with the lack of available time to further troubleshoot the Fluent display settings, is the reason a higher accuracy vorticity contour plot was not found.

2.7 Velocity

To more directly compare the calculated Fluent solution to Ghia, the *u*-velocity along a vertical line through the geometric center of the cavity was plotted along with the corresponding Ghia results (Table I in Ghia paper). Additionally, the *v*-velocity along a horizontal line through the geometric center of the cavity was plotted along with the corresponding Ghia results (Table II in Ghia paper). The Ghia velocity values along these lines are reported, and thus may be plotted together with the Fluent results to directly compare the two methods. Figure 11 presents the *u*-velocity plot and Figure 12 presents the *v*-velocity plot.

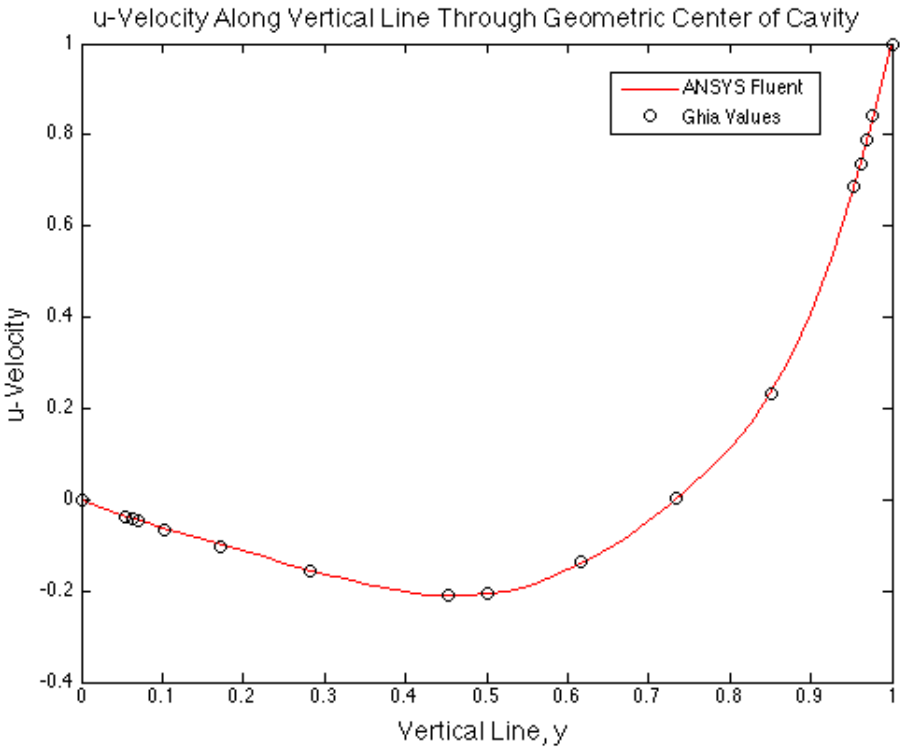

Figure 11: Comparison of *u*-velocity solution from Fluent to Ghia. Solution is along vertical line through geometric center of cavity.

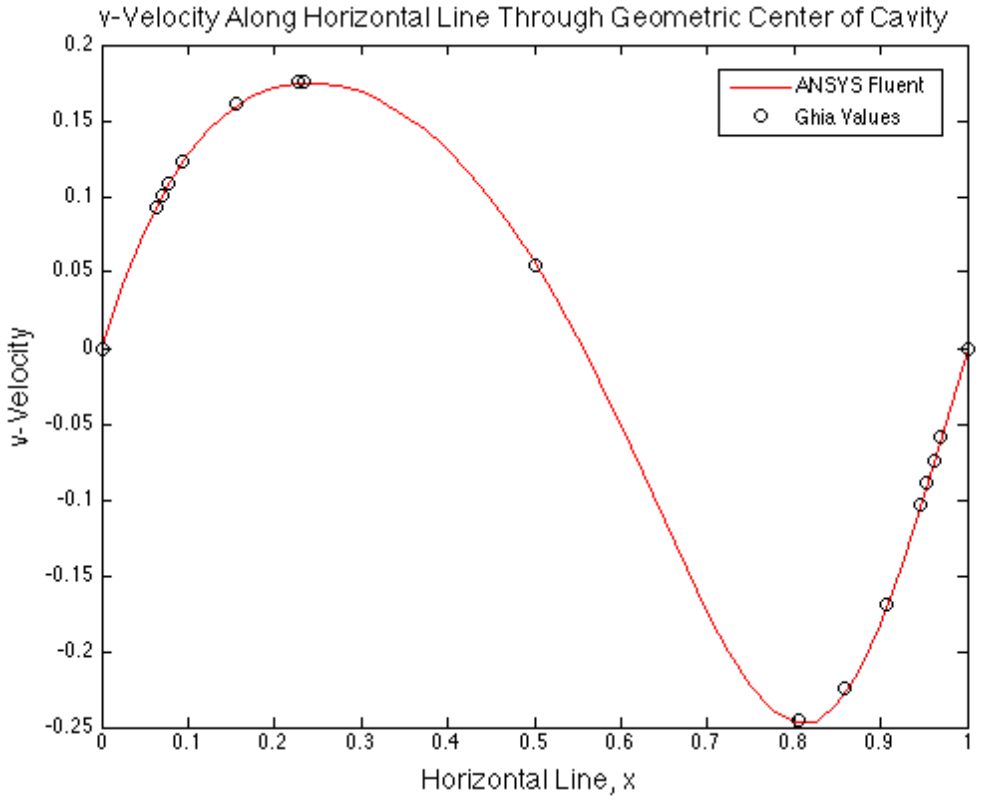

Figure 12: Comparison of *v*-velocity solution from Fluent to Ghia. Solution is along horizontal line through geometric center of cavity.

Both velocity plots very closely match the Ghia results for all specified points. This implies the Fluent simulation is an accurate simulation and accurately matches the solution found by Ghia. It validates the Fluent modeling performed in this section and the discrepancies between the Ghia and Fluent stream function and vorticity contour plots. Clearly the final Fluent velocity solutions are correct, but the Fluent modeler falsely hides away the correct stream function and vorticity values.

3.0 Fractional Step Method

The goal of this task, Task 2, was to determine the steady state solution for $Re = 100$ using the fractional step method on a staggered grid. Task 3 is also described in this section of the report. The goal of this task was to detail the numerical schemes utilized in the fractional step method including the index notation of all equations and shorthands used in the schemes.

The lid-driven cavity geometry and mesh was built using *x* and *y* direction vectors in MATLAB R2012b. All coding was written and run in MATLAB. The geometry was made to match the dimensions outlined in the problem statement. The kinematic viscosity was set to $\nu = 0.01$. Combined with the given Reynolds number, this corresponded to a lid velocity $U = 1$. Boundary conditions were set to match the given conditions from the problem statement. Further details on setting the boundary conditions and the staggered mesh utilized for the method are described in the corresponding sub-sections below. The fractional step method is a multi-part scheme that utilizes different numerical schemes for each step. The overall method is presented in the following section. Convergence and accuracy analysis, as well as the final results and solutions using the method are presented in the later sub-sections.

3.1 Method

The main idea of the Fractional Step method is to decouple the velocity and pressure terms by splitting the momentum equation into separate parts. The continuity equation is still enforced, but at strategic points within the method. The split momentum equation takes the form:

$$\frac{\partial \vec{v}}{\partial t} = N(\vec{v}) + \frac{1}{Re} L(\vec{v}) \qquad : \qquad \vec{v}^n \rightarrow \vec{v}^*$$

$$\frac{\partial \vec{v}}{\partial t} = -\nabla \varphi \qquad : \qquad \vec{v}^* \rightarrow \vec{v}^{n+1}$$

where

$$N(\vec{v}) = -\nabla \cdot (\vec{v}\vec{v})$$

$$L(\vec{v}) = \nabla^2 \vec{v}$$

The first part of the momentum equation finds the velocity terms at an intermediate time step, which will be referred to by a star symbol. The second part of the equation utilizes the pressure term and the intermediate velocities to determine the velocities at the next time step. However, due to this split, the velocity may not necessarily be in the subspace of solenoidal functions at the intermediate time step; it may violate the continuity equation. To counteract this problem, the intermediate time step velocities must be projected into the desired subspace. Additionally, the pressure term is no longer determined by a convection/diffusion equation. It is solely used to project the velocity into the subspace of solenoidal functions. As such, the term is no longer truly a pressure, but rather a Lagrange multiplier. The term is therefore changed to φ as seen in the above equation. To numerically solve the incompressible Navier-Stokes equations, the scheme is split into three parts:

I. Solve the first part of momentum equation (essentially viscous Burger's equation) to determine the velocities at the intermediate time step.
II. Calculate the Lagrange multiplier by solving the pressure Poisson equation using an implicit iterative method.
III. Use the final Lagrange multiplier to project the intermediate velocities into the subspace of solenoidal functions to determine the final solution.

The steady state solution is found using standard time advancement for an unsteady problem until the solution has sufficiently converged. The specific methods used to perform each of the three steps are described in the following sub-sections.

3.1.1 Step I

The first part of the momentum equation is solved using the forward in time central in space (FTCS) numerical scheme on a staggered grid. The steady state solution of the problem is desired and thus, high temporal accuracy is not important. Details of the staggered grid and its implications for the finite-difference equations for this step are detailed in the respective section below.

The momentum equation is split into its vector components and the conservative forms of these equations are used.

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

$$\frac{\partial v}{\partial t} = -\left(\frac{\partial uv}{\partial x} + \frac{\partial vv}{\partial y}\right) + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)$$

The solutions calculated during this step are the intermediate, star, velocities.

3.1.2 Step II

The second part of the FSM finds the Lagrange multiplier by solving the pressure Poisson equation using the Gauss-Seidel implicit method. This step has a separate convergence criteria it must meet for every time iteration.

The pressure Poisson equation takes the form:

$$\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} = \frac{1}{\Delta t}\left(\frac{\partial u^*}{\partial x} + \frac{\partial v^*}{\partial y}\right)$$

The solution calculated during this step is the converged Lagrange multiplier.

3.1.3 Step III

The third part of the FSM projects the intermediate velocities into the subspace of solenoidal functions using the new Lagrange multiplier. The equations for the velocities at the new time step take the form:

$$u = u^* - \Delta t \frac{\partial \varphi}{\partial x}$$

$$v = v^* - \Delta t \frac{\partial \varphi}{\partial u}$$

The solutions calculated during this step are the final velocities at the next time step. If the solutions have sufficiently converged following this calculation, these are the final calculated velocities.

3.2 Staggered Grid

To enforce conservation laws and to make the Fractional Step Method work, a staggered grid is used for the mesh. Velocities are defined at cell faces. Pressures (Lagrange multiplier) are defined at cell centers. Vorticities are defined at cell corners. Figure 13 presents an arbitrary cell on a staggered grid.

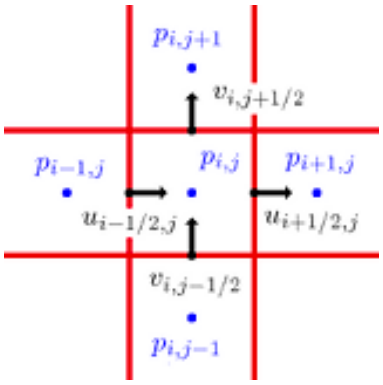

Figure 13: Staggered grid cell.

The biggest implication this has for the solution arrays is that the velocity terms are staggered and reside on half steps of the grid. In the code, the traditional whole indices for the velocity terms are changed as follows:

$$u(i,j) = u_{i+\frac{1}{2},j}$$

$$v(i,j) = v_{i,j+\frac{1}{2}}$$

The Lagrange multiplier terms remain at whole indices because they are at the cell centers. This leads to more complicated index notations and finite-difference equations for all terms. Additionally, ghost cell boundary conditions are necessary to determine the velocities. This leads to different sized arrays for the *u* and *v* velocities, as well as the Lagrange multiplier. The consequences for the numerical scheme index notation and the boundary conditions are described in detail in the following sections.

3.3 Numerical Schemes and Index Notation

3.3.1 Step I

The FTCS method discretizes time using a simple forward difference and discretizes space using a central difference. Together with the staggered mesh this leads to the following index notation equations:

$$u^{*}_{i+\frac{1}{2},j} = u^{n}_{i+\frac{1}{2},j} + \Delta t\left[-\left(\frac{\left(u^{n}_{i+1,j}\right)^2-\left(u^{n}_{i,j}\right)^2}{\Delta x}+\frac{(uv)^{n}_{i+\frac{1}{2},j+\frac{1}{2}}-(uv)^{n}_{i+\frac{1}{2},j-\frac{1}{2}}}{\Delta y}\right)+\frac{1}{Re}\left(\frac{u^{n}_{i+\frac{3}{2},j}-2u^{n}_{i+\frac{1}{2},j}+u^{n}_{i-\frac{1}{2},j}}{\Delta x^2}+\frac{u^{n}_{i+\frac{1}{2},j+1}-2u^{n}_{i+\frac{1}{2},j}+u^{n}_{i+\frac{1}{2},j-1}}{\Delta y^2}\right)\right]$$

$$v^{*}_{i,j+\frac{1}{2}} = v^{n}_{i,j+\frac{1}{2}} + \Delta t\left[-\left(\frac{(uv)^{n}_{i+\frac{1}{2},j+\frac{1}{2}}-(uv)^{n}_{i-\frac{1}{2},j+\frac{1}{2}}}{\Delta x}+\frac{\left(v^{n}_{i,j+1}\right)^2-\left(v^{n}_{i,j}\right)^2}{\Delta y}\right)+\frac{1}{Re}\left(\frac{v^{n}_{i+1,j+\frac{1}{2}}-2v^{n}_{i,j+\frac{1}{2}}+v^{n}_{i-1,j+\frac{1}{2}}}{\Delta x^2}+\frac{v^{n}_{i,j+\frac{3}{2}}-2v^{n}_{i,j+\frac{1}{2}}+v^{n}_{i,j-\frac{1}{2}}}{\Delta y^2}\right)\right]$$

However, because of the staggered grid, certain velocity locations must be determined from linear interpolation of the surrounding cells. These shorthands are given by the following equations:

$$u^{n}_{i+1,j}=\frac{1}{2}\left(u^{n}_{i+\frac{1}{2},j}+u^{n}_{i+\frac{3}{2},j}\right)$$

$$u^{n}_{i,j}=\frac{1}{2}\left(u^{n}_{i+\frac{1}{2},j}+u^{n}_{i-\frac{1}{2},j}\right)$$

$$u^{n}_{i+\frac{1}{2},j+\frac{1}{2}}=\frac{1}{2}\left(u^{n}_{i+\frac{1}{2},j}+u^{n}_{i+\frac{1}{2},j+1}\right)$$

$$u^{n}_{i+\frac{1}{2},j-\frac{1}{2}}=\frac{1}{2}\left(u^{n}_{i+\frac{1}{2},j}+u^{n}_{i+\frac{1}{2},j-1}\right)$$

$$u^{n}_{i-\frac{1}{2},j+\frac{1}{2}}=\frac{1}{2}\left(u^{n}_{i-\frac{1}{2},j}+u^{n}_{i-\frac{1}{2},j+1}\right)$$

$$v^n_{i,j+1} = \frac{1}{2}\left(v^n_{i,j+\frac{1}{2}} + v^n_{i,j+\frac{3}{2}}\right)$$

$$v^n_{i,j} = \frac{1}{2}\left(v^n_{i,j+\frac{1}{2}} + v^n_{i,j-\frac{1}{2}}\right)$$

$$v^n_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{1}{2}\left(v^n_{i,j+\frac{1}{2}} + v^n_{i+1,j+\frac{1}{2}}\right)$$

$$v^n_{i+\frac{1}{2},j-\frac{1}{2}} = \frac{1}{2}\left(v^n_{i,j-\frac{1}{2}} + v^n_{i+1,j-\frac{1}{2}}\right)$$

$$v^n_{i-\frac{1}{2},j+\frac{1}{2}} = \frac{1}{2}\left(v^n_{i-1,j+\frac{1}{2}} + v^n_{i,j+\frac{1}{2}}\right)$$

3.3.2 Step II

Gauss-Seidel is an implicit iterative method that utilizes the available solutions at the next time step (at previous special steps) as well as those solutions still at the previous step to determine the solution at the next time step. It iterates within its own loop until the Lagrange multiplier term's specific convergence is met.

It should be noted that the Lagrange multiplier terms are located at cell centers and do not face such dire results index changes due to the staggered mesh. The index notation equation is as follows:

$$\varphi^{n+1}_{i,j} = \frac{1}{4}\left(\varphi^{n+1}_{i-1,j} + \varphi^{n}_{i+1,j} + \varphi^{n+1}_{i,j-1} + \varphi^{n}_{i,j+1}\right) - \frac{1}{4}\frac{1}{\Delta t} f_{i,j}$$

where

$$f_{i,j} = \frac{\left(u^*_{i+\frac{1}{2},j} - u^*_{i-\frac{1}{2},j}\right)}{\Delta x} + \frac{\left(v^*_{i,j+\frac{1}{2}} - v^*_{i,j-\frac{1}{2}}\right)}{\Delta y}$$

The convergence of the Gauss-Seidel step is determined by the value of the Gauss-Seidel residual. The residual is found using the following equation:

$$Res_{GS} = \frac{\left(\varphi^n_{i+1,j} - 2\varphi^n_{i,j} + \varphi^n_{i-1,j}\right)}{\Delta x^2} + \frac{\left(\varphi^n_{i,j+1} - 2\varphi^n_{i,j} + \varphi^n_{i,j-1}\right)}{\Delta y^2} - \frac{1}{\Delta t}\left(\frac{\left(u^*_{i,j} - u^*_{i-1,j}\right)}{\Delta x} + \frac{\left(v^*_{i,j} - v^*_{i,j-1}\right)}{\Delta y}\right)$$

3.3.3 Step III

The last step of the Fractional Step Method projects the intermediate velocities into the subspace of solenoidal functions using the new Lagrange multiplier term and the following equations:

$$u_{i+\frac{1}{2},j}^{n+1} = u_{i+\frac{1}{2},j}^{*} - \frac{\Delta t}{\Delta x}\left(\varphi_{i+1,j}^{n} - \varphi_{i,j}^{n}\right)$$

$$v_{i,j+\frac{1}{2}}^{n+1} = v_{i,j+\frac{1}{2}}^{*} - \frac{\Delta t}{\Delta y}\left(\varphi_{i,j+1}^{n} - \varphi_{i,j}^{n}\right)$$

The determine whether to continue the iterative Fractional Step Method at this point, the convergence of the final velocity solutions is determined. At steady state the solution is in equilibrium and no longer changing. More specifically, the derivatives of the velocity solutions should be zero. Because of the finite difference approximation, zero will never truly be reached, and instead, the values of these derivatives are used to determine convergence. The index notation residual terms are then given by:

$$Res_u = \frac{u_{i+\frac{1}{2},j}^{n} - u_{i+\frac{1}{2},j}^{n-1}}{\Delta t}$$

$$Res_v = \frac{v_{i,j+\frac{1}{2}}^{n} - v_{i,j+\frac{1}{2}}^{n-1}}{\Delta t}$$

3.4 Boundary Conditions

3.4.1 Velocities

Because of the staggered mesh, velocities tangential to the boundaries are not known. Ghost cell boundary conditions must be used to determine these boundary conditions. Figure 14 presents a diagram of a ghost cell boundary condition; the ghost cell is designated by the gray color.

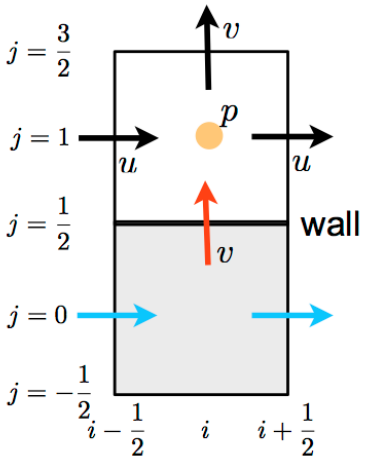

Figure 14: Ghost Cell Boundary Condition.

The example in the diagram is at the bottom boundary of the cavity. The *v*-velocity is normal to the boundary and thus it is directly known from the boundary conditions specified in the problem statement. However, the *u*-velocity does not reside on the wall itself, but instead on the face of the ghost cell. Thus, the known velocity at the wall and linear interpolation of velocities on either side of the wall must be used to determine the ghost cell boundary condition. This relation takes the following form:

$$u_{wall} = \frac{1}{2}\left(u_{i+\frac{1}{2},1} + u_{i+\frac{1}{2},0}\right)$$

$$\Rightarrow \quad u_{i+\frac{1}{2},0} = 2u_{wall} - u_{i+\frac{1}{2},1}$$

As a result, the velocity arrays are different sizes. More specifically, for a mesh of M x N cells, the arrays are of the following size:

u-velocity → M+1 x N+2

v-velocity → M+2 x N+1

Applying the above principles to the two velocities yields the following boundary condition equations:

$$u_{i+\frac{1}{2},N+2} = 2U - u_{i+\frac{1}{2},N+1}$$

$$u_{i+\frac{1}{2},0} = -u_{i+\frac{1}{2},1}$$

$$u_{0,j} = 0$$

$$u_{M+1,j} = 0$$

$$v_{i,N+1} = 0$$

$$v_{i,0} = 0$$

$$v_{0,j+\frac{1}{2}} = -v_{1,j+\frac{1}{2}}$$

$$v_{M+2,j+\frac{1}{2}} = -v_{M+1,j+\frac{1}{2}}$$

The boundary conditions are enforced immediately following the calculation of a new set of velocity arrays. This is done for the star velocities as well as the final solution velocities.

3.4.2 Lagrange Multiplier

The boundary conditions for the Lagrange multiplier are much simpler compared to the velocities. The boundaries for the multiplier are taken to be Neumann boundary conditions and defined as:

$$\varphi_{i,0}^{n+1} = \varphi_{i,1}^{n+1}$$

$$\varphi_{i,N+2}^{n+1} = \varphi_{i,N+1}^{n+1}$$

$$\varphi_{0,j}^{n+1} = \varphi_{1,j}^{n+1}$$

$$\varphi_{M+2,j}^{n+1} = \varphi_{M+1,j}^{n+1}$$

These conditions imply a third array size for the Lagrange multiplier term, which is of the following size:

$$\varphi \rightarrow \text{M+2 x N+2}$$

The boundary conditions are enforced immediately following the calculation of the Gauss-Seidel step, prior to the residual calculation. They are enforced for every Gauss-Seidel iteration loop.

3.5 Stable Time Step

The FTCS method has stability constraints that limit the spatial and time discretization used to determine the solution. In this case, the method has constraints due to the parabolic and hyperbolic parts of the equations. To simplify the constraints and determine a stable time step, the spatial steps were set such that $dx = dy = h$. The parabolic constraint for the 2D method is then given as:

$$\frac{\nu \Delta t}{h^2} \leq \frac{1}{4}$$

Likewise, the hyperbolic constraint for the 2D method is simplified and given as:

$$\frac{(a+b)\Delta t}{h} \leq 1$$

Additionally, the method has a third constraint based on the cell Reynolds number. This Reynolds number is based on the cell length and cell velocity for each finite difference cell element. This constraint is given by the following equation:

$$\frac{(a+b)h}{\nu} \leq 2$$

The variables a and b are based on the solutions u and v. In a traditional viscous Burger's equation, with no pressure term, a and b are different for the two velocities. However, because of

the pressure term in the Navier-Stokes equations, the enforcement of the continuity equation, *a* and *b* are the same for both *u* and *v* and given by the follow equations:

$$a = \max(u)$$

$$b = \max(v)$$

Thus, to determine the stable time step, a spatial step is chosen. The code determines two stable time steps, one for the parabolic constraint and one for the hyperbolic constraint. The smallest time of the two will satisfy both constraints and is chosen as the stable time step to find the solution. This calculation is also performed within the solution time loop to ensure stability throughout the temporal domain. The equations for the two time steps are given as:

$$\Delta t_1 = \frac{h^2}{4\nu}$$

$$\Delta t_2 = \frac{h}{(a+b)}$$

The Reynolds cell number constraint checks the selected spatial step for stability. It does not calculate any value, but the code will output an error message if the constraint is not met. Because the time step is based on the selected spatial step, the stability of the scheme is really based on the spatial step, as a suitably small discretization must be chosen to meet this constraint. More specifically, the stability is based on the number of elements in the *x* and *y* vectors, *M* (which =*N*), as this is the input at the start of the code.

3.6 Convergence

3.6.1 Velocity Solutions

The main solution loop increases in time by the iteration specific time step until steady state is reached. Steady state is determined by the residuals of the velocity equations. As described in Section 3.3.3, the first order time derivatives of the velocity equations are used for the residuals. The code creates two solution residual arrays, one for each velocity. The maximum value of each array is combined into a final solution residual array with two indices. The maximum of these two values determines whether the codes continues or stops.

To ensure an accurate solution, the convergence criterion for the main solution was set to 10^{-7}. Once the largest residual is smaller than this value, the code stops. Additionally, the maximum residual for both velocities is stored at each time step to plot a time history of convergence of the two velocity solutions. Figure 15 through Figure 17 present the time history of convergence for the three mesh sizes used to determine the final solutions in this project.

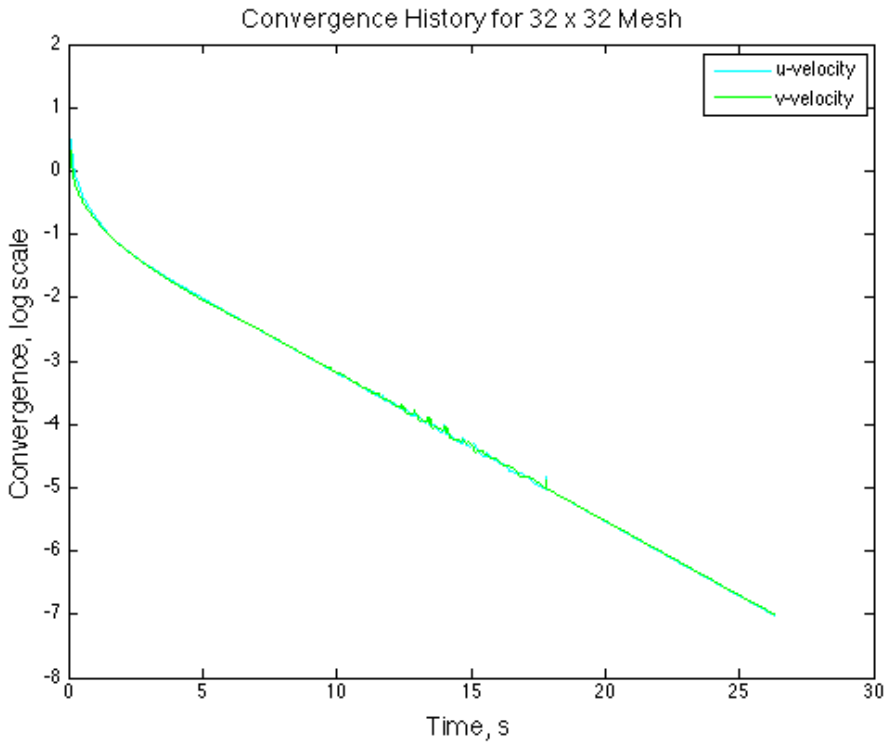

Figure 15: Convergence Time History of Velocity Solutions for Coarse Mesh, 32 x 32.

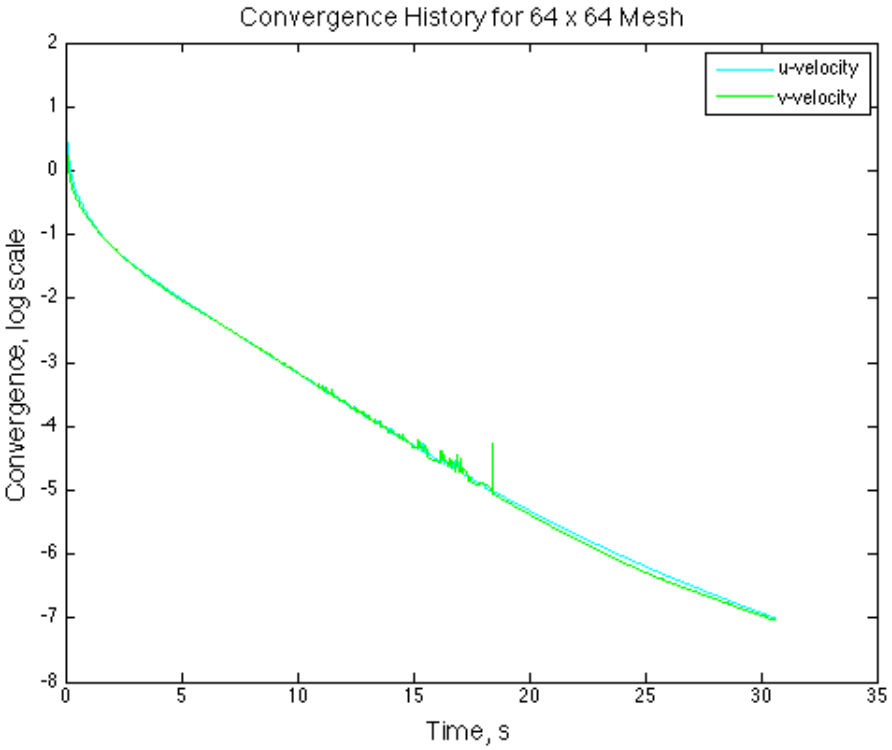

Figure 16: Convergence Time History of Velocity Solutions for Medium Mesh, 64 x 64.

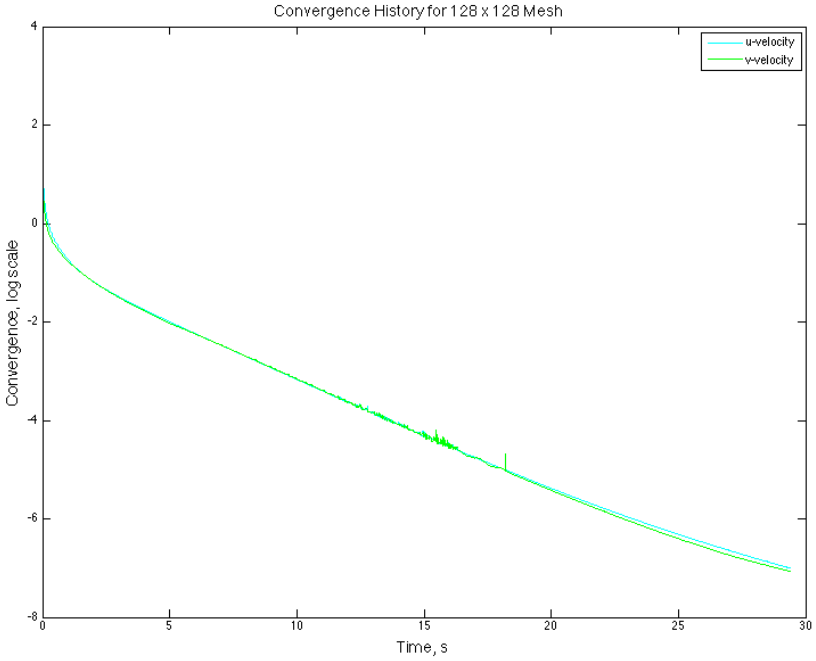

Figure 17: Convergence Time History of Velocity Solutions for Fine Mesh, 128 x 128.

All three mesh refinements converged in a similar manner. The only difference between the meshes was the computational time to solution, which increased dramatically as the resolution was doubled each time. Additionally, while the time to convergence was roughly the same for all three meshes, the number of time iterations increased as the mesh refinement increased. This is expected, as the stable time step is determined by the spatial discretization. A smaller spatial discretization implies a small time step.

3.6.2 Lagrange Multiplier

The Lagrange multiplier Gauss-Seidel iteration loop increases until its own convergence criterion is met, at which point the Lagrange multiplier array is used to determine the final velocity solutions and begin the main solution loop anew. To increase computational speed, the Gauss-Seidel iteration loop uses a dynamic convergence criterion. The range for the convergence criterion is chosen to be:

$$Conv_{GS} = 10^{-3} \geq 10^{-2} * Res_{u,v} \geq 10^{-7}$$

Thus, the Gauss-Seidel convergence criterion starts out relaxed and gets progressively stricter as the velocity solutions approach steady state. When the main solution residual is large, a highly accurate Gauss-Seidel is unnecessary and will prolong the computational time of a MATLAB code of this nature by 5 to 10x. At this stage of the main solution, the Gauss-Seidel convergence criterion is set to 10^{-3}. As the solution residual approaches its own convergence criterion it is more important for the Gauss-Seidel solution to be highly accurate and be subject to a stricter convergence criterion. At this stage of the solution, the Gauss-Seidel convergence criterion

changes to 10^{-7}. The factor 10^{-2} is used to determine whether the solution is approaching either end of the residual spectrum.

3.7 Accuracy

Analysis of the accuracy of the solution is performed using Richardson Extrapolation and Grid Convergence Index as described in Roache (1993). The analysis requires scalar values for comparison and thus the maximum value of the velocity solutions along their respective geometric centerlines are used. The goal is have the accuracy of the solution be within 2%.

The analysis requires 3 values of maximum velocity for each solution, each with increasing grid refinement. For the most accurate solution, a grid refinement of 2 is chosen for each refinement level. The refinement is normalized to the finest mesh such that the grid spacing, h, is 1 for the finest mesh, 2 for the medium mesh, and 4 for the coarsest mesh. This corresponds to M = 128, 64, and 32 for the finest, medium, and coarsest mesh, respectively.

Let the finest, medium, and coarsest mesh be identified as grids 1, 2, and 3, respectively. Let f be defined as the maximum velocities specified above with subscripts representing the mesh refinement used. For the chosen grid refinement at each refinement level, $r = 2$, the order of convergence, p, is calculated using the following equation:

$$p = \ln\left(\frac{f_3 - f_2}{f_2 - f_1}\right) / \ln(r)$$

The Richardson extrapolation, the estimated true solution (at $h = 0$) may then be found using the following equation:

$$f_{h=0} \approx f_1 + \frac{f_1 - f_2}{r^p - 1}$$

Using this value, the Grid Convergence Index formula may be used to determine the error band of the solution. GCI is performed from one grid refinement level to the next, i.e. from 3 to 2 or 2 to 1. The following equation presents the GCI from the medium to the fine mesh, which is the error band that is compared to the desired accuracy level, but a GCI from the coarsest to the medium mesh may also be found using the same formula, but with the relevant f solutions changed.

$$GCI_{12} = 1.25\frac{\left|\frac{f_1 - f_2}{f_1}\right|}{r^p - 1}$$

Following this analysis, the accuracy of the solutions was determined and presented in Figure 18 and Figure 19.

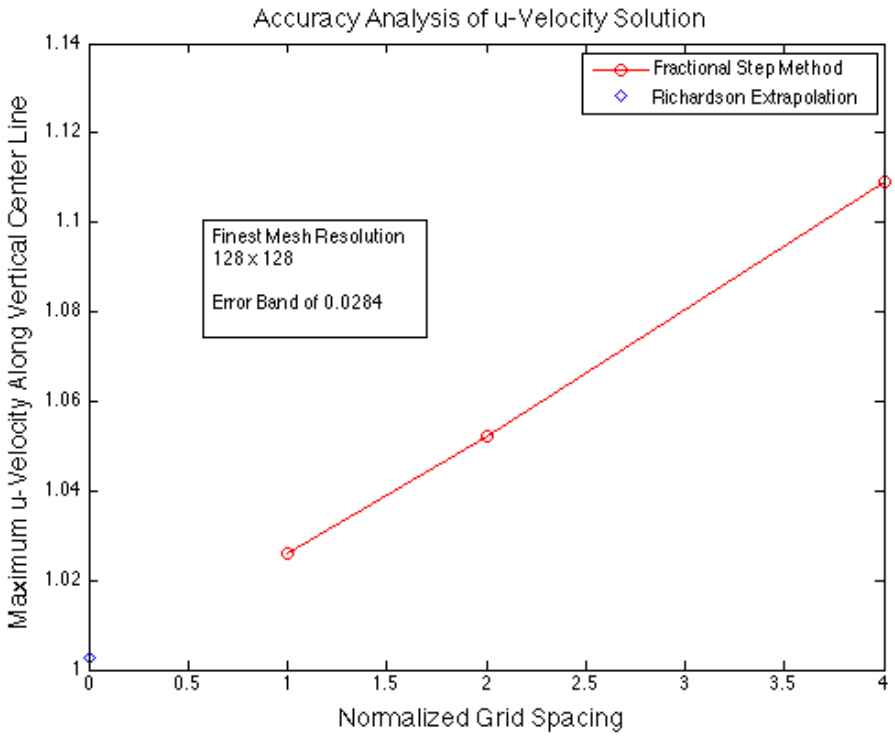

Figure 18: Accuracy analysis of the *u*-velocity solution.

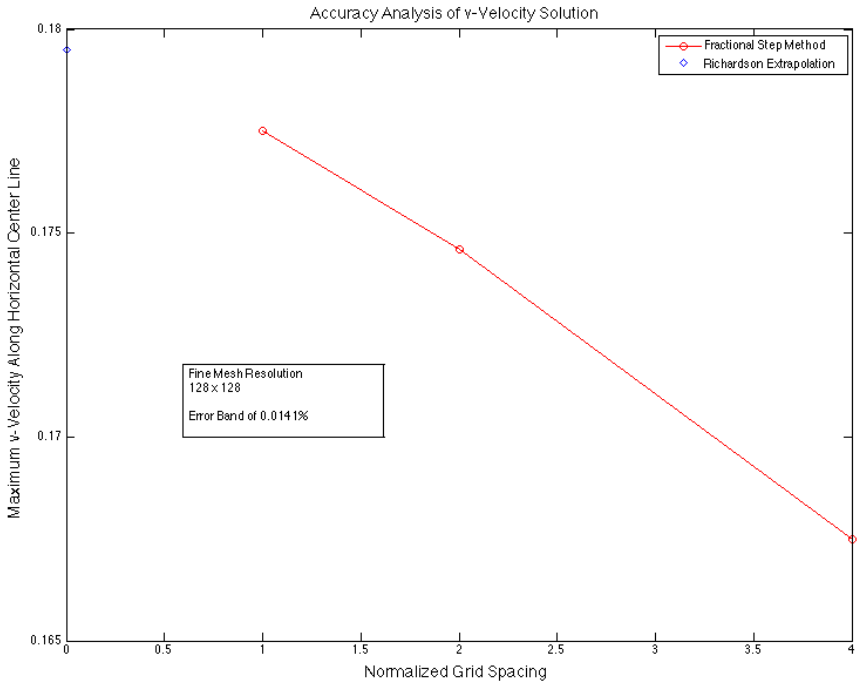

Figure 19: Accuracy analysis of the *v*-velocity solution.

From this analysis, it is clear the finest mesh resolution, $M = 128$, is within the desired accuracy, as it has error bands of **0.0284%** and **0.0141%**, for the *u* and *v* velocities, respectively. Additionally, the estimated true solutions from the Richardson extrapolation were found to be **1.0025** and **0.1795** for the *u* and *v* velocities.

Similar to the ANSYS Fluent task, the maximum percent difference between the Ghia and Fractional Step Method velocity results may be determined. The results of this accuracy analysis are presented with the Fluent percent difference results in Table 2.

Table 2: Accuracy analysis of FSM and Ghia, as well as Fluent and Ghia, velocity solutions.

Compared Solutions	Maximum Percent Difference Between Ghia and FSM	Maximum Percent Difference Between Ghia and Fluent
u-velocity	0.91%	1.59%
v-velocity	1.02%	1.89%

As expected, the Fluent results are slightly closer to Ghia than the Fractional Step Method. This is due Fluent's higher order solving algorithms and over/under relaxation optimization in its main solver algorithms. Both methods are highly accurate and satisfy any accuracy requirement of the project. Additionally, the GCI accuracy goal of 2% was clearly met by the very low GCI values found for both velocities.

3.8 Vorticity

The vorticity is defined as the curl of the velocity vector and given by:

$$\Omega = \nabla \text{ x } \vec{v}$$

$$\Omega = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$$

Because of the staggered grid, vorticities are on cell corners and the vorticity array is size (M+1 x N+1). The vorticity is found from the derivatives of the final solution velocities once steady state has been reached. A central difference is used to approximate the derivatives, resulting in the following index notation equation for vorticity:

$$\Omega_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{v_{i+1,j+\frac{1}{2}} - v_{i,j+\frac{1}{2}}}{\Delta x} + \frac{u_{i+\frac{1}{2},j+1} - v_{i+\frac{1}{2},j}}{\Delta y}$$

Because the vorticity array is exactly one element smaller in each dimension, the boundary conditions may be calculated along with the interior of the array. In other words, the spatial step loops for vorticity both loop from 1 to N+1, which sets the boundary conditions based on the interior points. For each mesh refinement level, the final velocity solutions were used to calculate the vorticity distribution for the cavity. Vorticity contours were then plotted for the same values Ghia uses in Table 3 of the Ghia paper. The calculated vorticity contours are presented together with the Ghia values (from Table 4 in the Ghia paper) in Figure 20 through Figure 22.

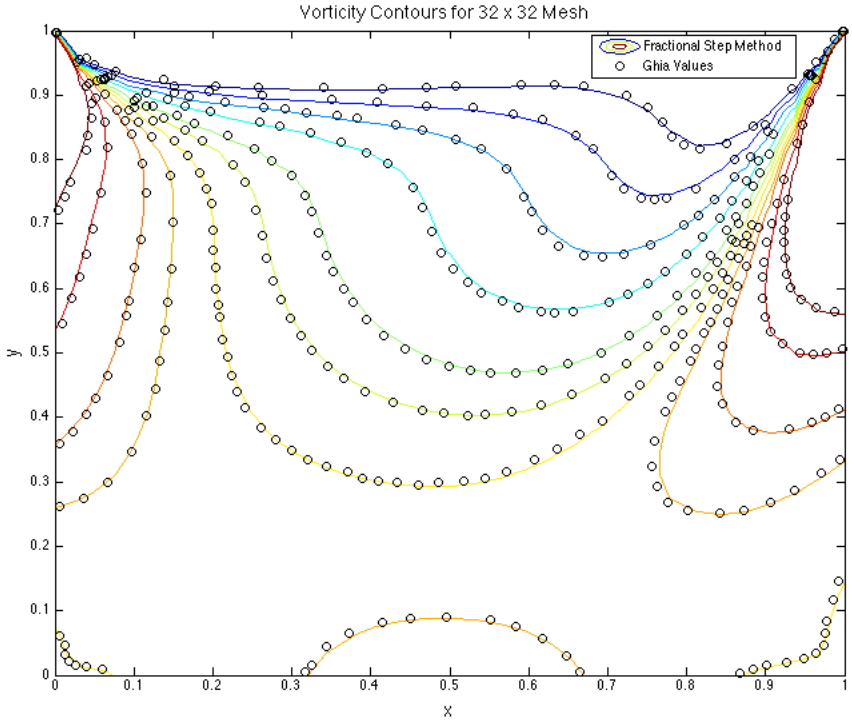

Figure 20: Comparison of Fractional Step Method and Ghia vorticity contours for coarse mesh, 32 x 32.

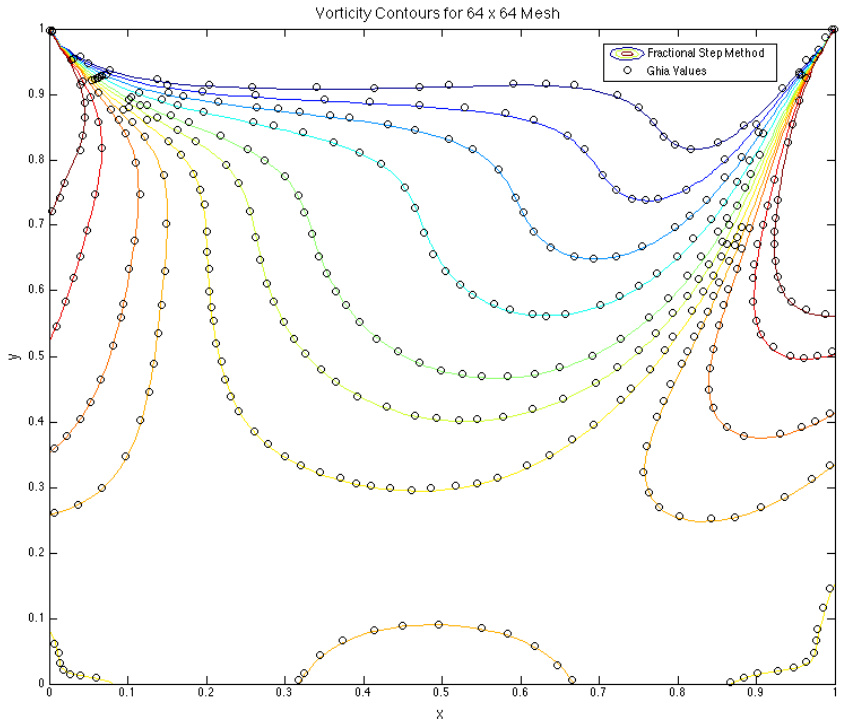

Figure 21: Comparison of Fractional Step Method and Ghia vorticity contours for medium mesh, 64 x 64.

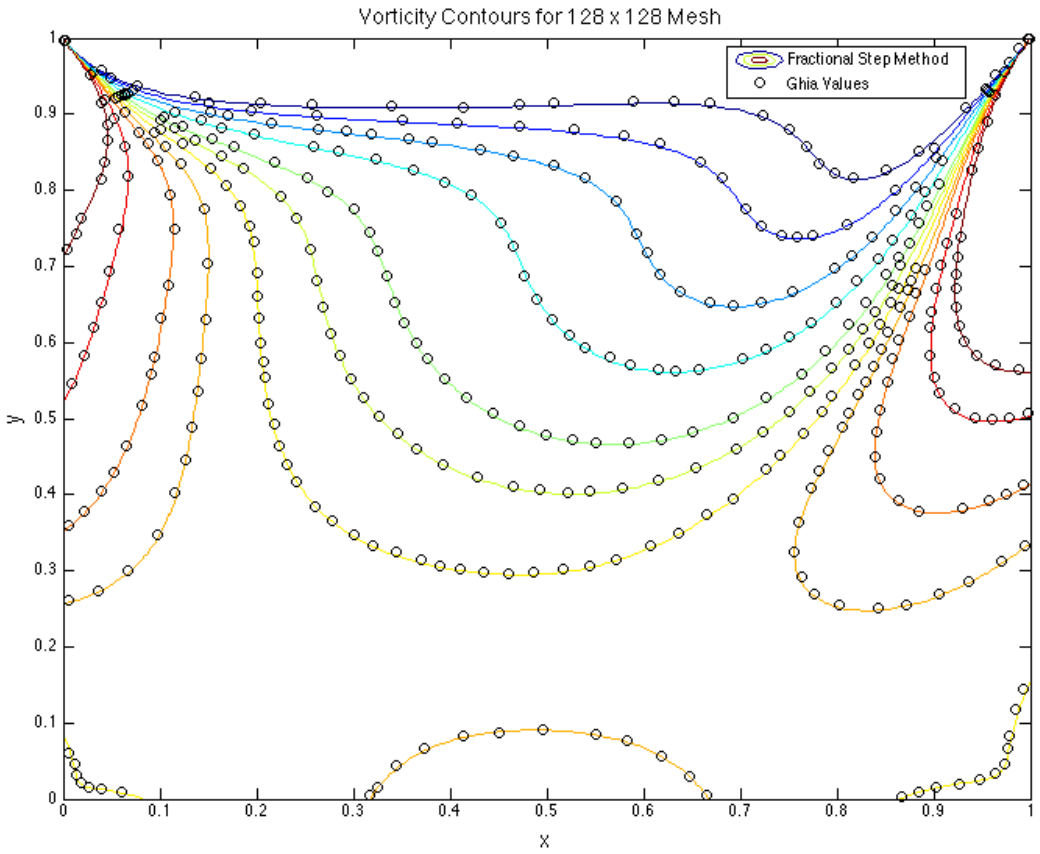

Figure 22: Comparison of Fractional Step Method and Ghia vorticity contours for fine mesh, 128 x 128.

Even at the medium resolution mesh, the accuracy of the Fractional Step Method in comparison to Ghia is very good. It is difficult to see much improvement between the fine and medium mesh because of the high accuracy of the medium mesh. Overall, this shows good quality results using the Fractional Step Method and a good comparison to the expected Ghia values.

3.9 Velocities

Just as in the Fluent simulation task, to more directly compare the calculated FSM solution to Ghia, the *u*-velocity along a vertical line through the geometric center of the cavity was plotted along with the corresponding Ghia results (Table I in Ghia paper). Additionally, the *v*-velocity along a horizontal line through the geometric center of the cavity was plotted along with the corresponding Ghia results (Table II in Ghia paper). The Ghia velocity values along these lines are reported, and thus are plotted together with the FSM results to directly compare the two methods. This comparison was conducted for all three mesh resolutions. Figure 23 through Figure 25 present the *u*-velocity plots and Figure 26 through Figure 28 present the *v*-velocity plots.

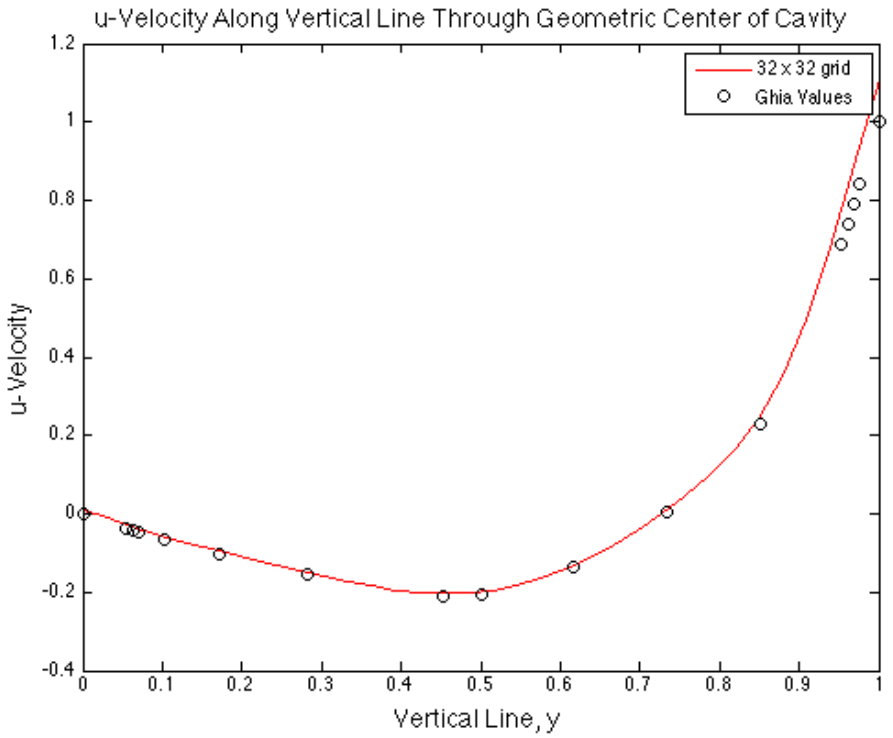

Figure 23: Comparison of *u*-velocity solution from FSM to Ghia. Solution is along vertical line through geometric center of cavity. Solution is for coarse mesh, 32 x 32.

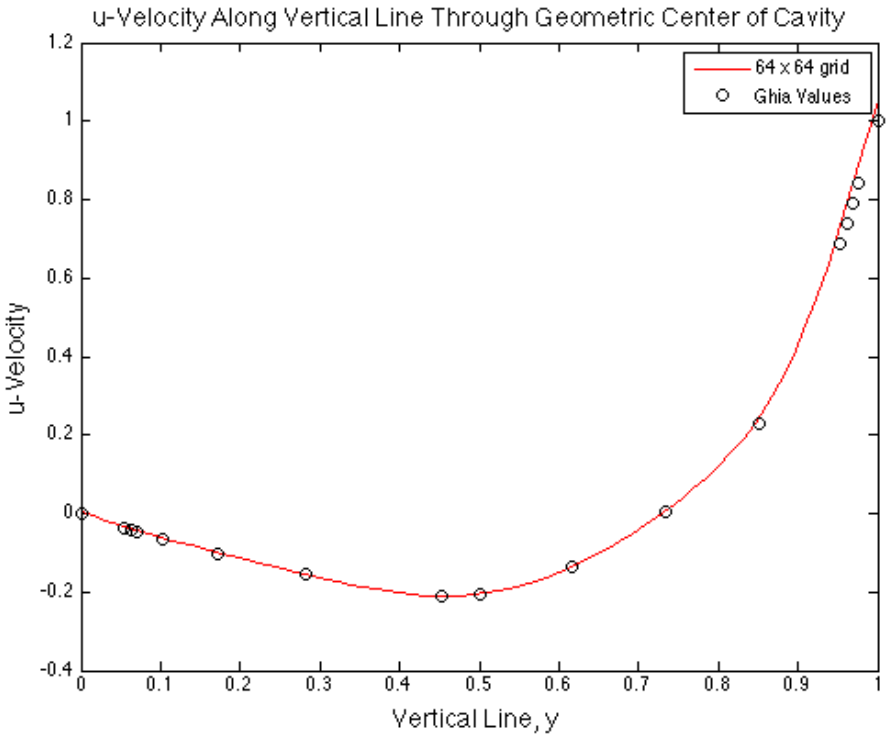

Figure 24: Comparison of *u*-velocity solution from FSM to Ghia. Solution is along vertical line through geometric center of cavity. Solution is for medium mesh, 64 x 64.

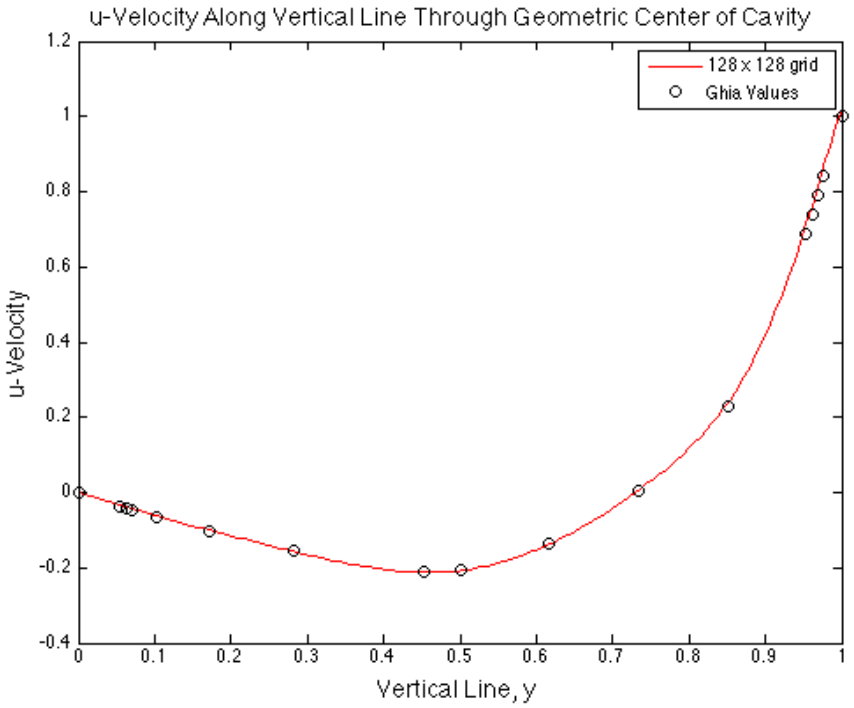

Figure 25: Comparison of *u*-velocity solution from FSM to Ghia. Solution is along vertical line through geometric center of cavity. Solution is for fine mesh, 128 x 128.

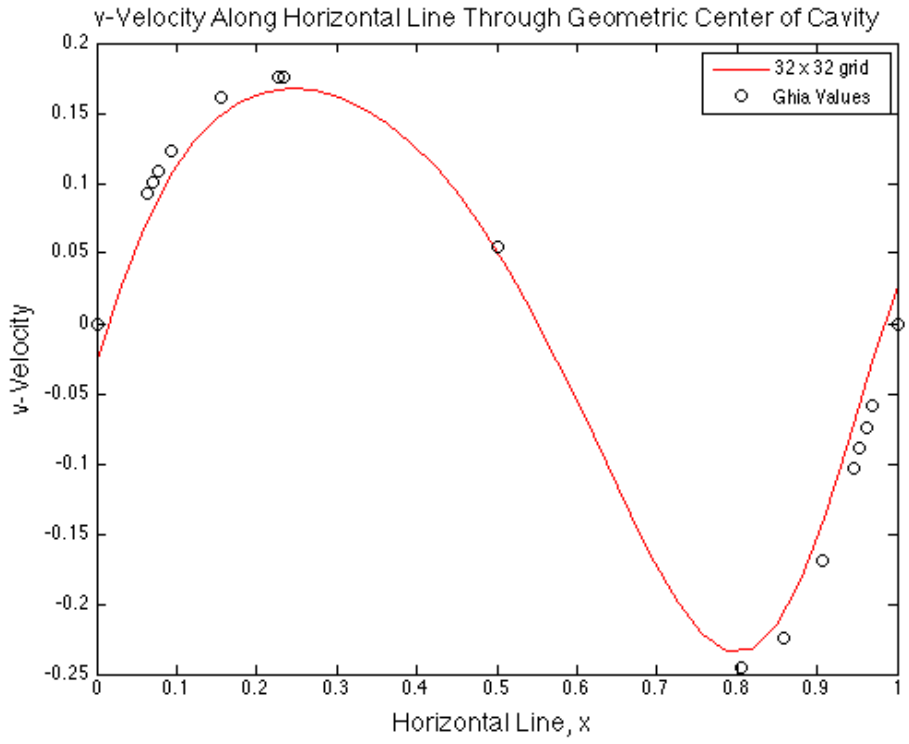

Figure 26: Comparison of *v*-velocity solution from FSM to Ghia. Solution is along horizontal line through geometric center of cavity. Solution is for coarse mesh, 32 x 32.

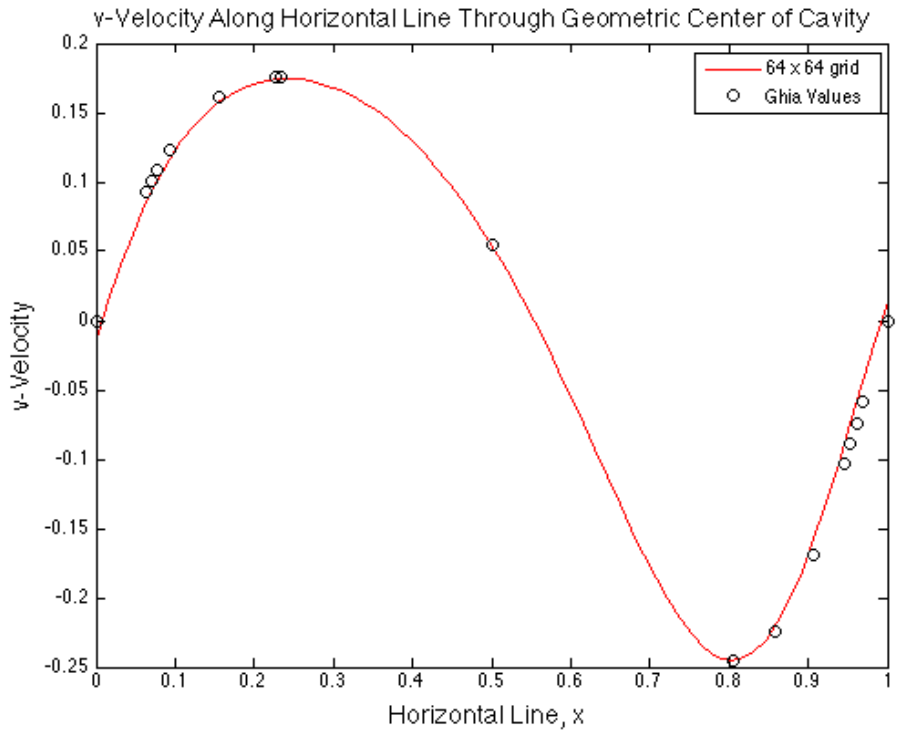

Figure 27: Comparison of *v*-velocity solution from FSM to Ghia. Solution is along horizontal line through geometric center of cavity. Solution is for medium mesh, 64 x 64.

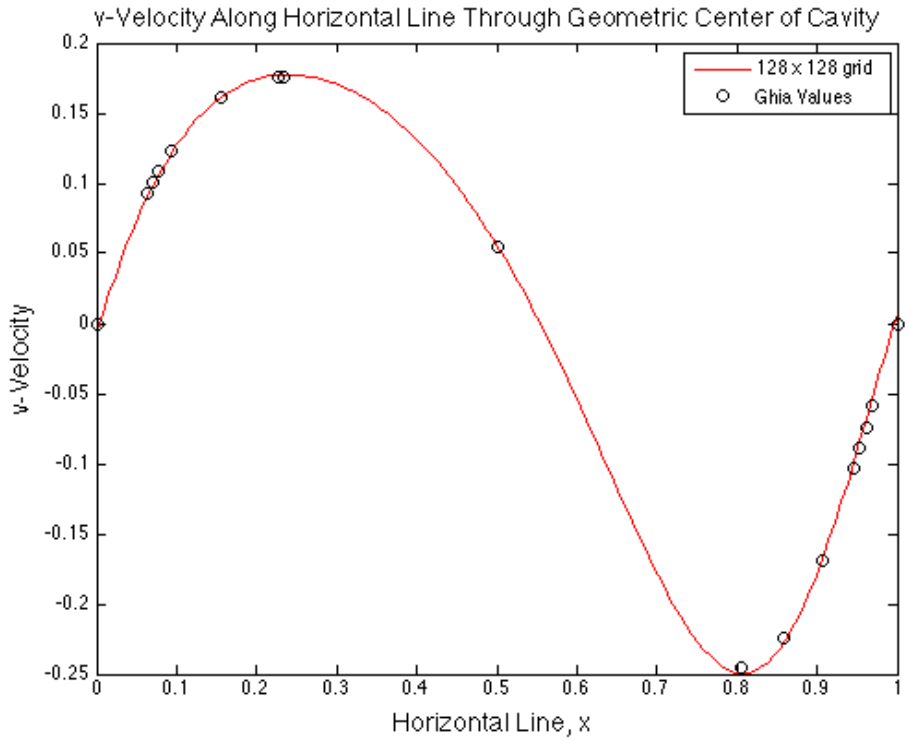

Figure 28: Comparison of *v*-velocity solution from FSM to Ghia. Solution is along horizontal line through geometric center of cavity. Solution is for fine mesh, 128 x 128.

For both solutions, FSM solution is not quite as good as Ghia for the coarsest mesh. However, when the resolution doubles to the medium mesh, the solutions increase to be very close, if not exactly the same as, the Ghia values. The accuracy increases further for the fine mesh. These results, together with the maximum percent differences from Ghia presented in Table 2 show the final velocity solutions are highly accurate and of good quality when compared to Ghia's results. The GCI results for both velocities were very low which further implied high accuracy of the final mesh resolution solution. While Fluent was slightly closer to Ghia than the Fractional Step for the solution velocities, both methods produced very accurate results that would surely satisfy accuracy requirements of such a simulation problem.

4.0 Code

4.1 Solver

```
% MAE471 - CFD
% Final Project
% Solving incompressible, 2D Navier-Stokes with Fractional Step Method on a
% Staggered Grid

%Method involves 3 steps:
%   I)      Solve Viscous Burgers Equation using FTCS
%   II)     Calculate Lagrange multiplier using Gauss-Seidel
%   III)    Project solution into subspace of solenoidal velocity fields

%%

clear all
clc

tic

%%%%%%%%%%%%    Inputs
Re = 100; %Reynolds number
L = 1; %length and height of cavity
nu = 0.01; %kinematic viscosity
U = Re*nu/L; %velocity at top wall

M = 32; %number of cells in x
N = 32; %number of cells in y

conv_crit = 10^-7; %convergence criteria

%%%%%%%%%%%%    Discretization
dx = L/M;
dy = L/N;
rh = 1/dx; %recipricol of h (dx = dy)
rh2 = 1/dx^2; %recipricol of h^2
rRe = 1/Re; %recipricol of Re

%%%%%%%%%%%%    Initial Conditions
u = zeros(M+1,N+2);
v = zeros(M+2,N+1);
u_star = u;
v_star = v;
phi = zeros(M+2,N+2);
res_gs = phi;
gs_RHS = phi;
gs_RHS2 = phi;
omega = zeros(M+1,N+1);
u_plot = zeros(1,N+1);
v_plot = zeros(1,N+1);

n = 0; %counter
t = 0; %time
n_gs_t = 0; %gauss-seidel counter (total number of iterations for all time)
conv = 1; %initial convergence value
```

```
%%%%%%%%%%%%    Boundary Conditions
%u velocity
u(:,N+2) = 2*U-u(:,N+1); %top wall
u(:,1) = -u(:,2); %bottom wall
u(1,:) = 0; %left wall
u(M+1,:) = 0; %right wall
%v velocity
v(:,N+1) = 0; %top wall
v(:,1) = 0; %bottom wall
v(1,:) = -v(2,:); %left wall
v(M+2,:) = -v(M+1,:); %right wall

%%%%%%%%%%%%    Calculate Solutions

while conv > conv_crit

    %%%%% find dt from stability checks
    %a & b values for constraints
    a_grid = u;
    b_grid = v;
    a = max(max(abs(a_grid)));
    b = max(max(abs(b_grid)));

    %following constraints, assume dx = dy

    %first stability constraint, parabolic part
    dt1 = 0.25*dx^2/nu;

    %second stability constraint, hyperbolic part
    dt2 = dx/(a+b);

    %pick smallest dt
    if dt1 >= dt2
        dt = dt2;
    else
        dt = dt1;
    end

    %check third stability constraint, cell Reynolds number
    if (a+b)*dx/nu > 2
        disp('Unstable - Cell Reynolds Number')
    end

    %%%%% increase counters
    t = t+dt; %increase time by stable dt
    n = n+1; %counter
    t_plot(n) = t; %time for plotting
    n_gs = 0; %gauss-seidel counter for each set of loops

    %   Step I of Fractional Step Method

    %%%%% store solution at n for convergence check
    r_u = u;
    r_v = v;
```

```
%%%%% find solutions to u @ time step n_star using FTCS
for j = 2:N+1
    for i = 2:M
        %averages because of staggered mesh
        ua = 0.5*(u(i+1,j)+u(i,j)); %u_i+1,j
        ub = 0.5*(u(i,j)+u(i-1,j)); %u_i,j
        uc = 0.5*(u(i,j+1)+u(i,j)); %u_i+1/2,j+1/2
        ud = 0.5*(u(i,j-1)+u(i,j)); %u_i+1/2,j-1/2

        va = 0.5*(v(i+1,j)+v(i,j)); %v_i+1/2,j+1/2
        vb = 0.5*(v(i+1,j-1)+v(i,j-1)); %v_i+1/2,j-1/2

        %u-velocity star value
        u_star(i,j) = u(i,j)+dt*(-((ua^2-ub^2)*rh+(uc*va-ud*vb)*rh)...
            +rRe*((u(i+1,j)-2*u(i,j)+u(i-1,j))*rh2+(u(i,j+1)-2*u(i,j)...
            +u(i,j-1))*rh2));
    end
end

%%%%% find solutions to v @ time step n_star using FTCS
for j = 2:N
    for i = 2:M+1
        %averages because of staggered mesh
        uc = 0.5*(u(i,j+1)+u(i,j)); %u_i+1/2,j+1/2
        ue = 0.5*(u(i-1,j)+u(i-1,j+1)); %u_i-1/2,j+1/2

        va = 0.5*(v(i+1,j)+v(i,j)); %v_i+1/2,j+1/2
        vc = 0.5*(v(i,j+1)+v(i,j)); %v_i,j+1
        vd = 0.5*(v(i,j)+v(i,j-1)); %v_i,j
        vf = 0.5*(v(i,j)+v(i-1,j)); %v_i-1/2,j+1/2

        %v-velocity star value
        v_star(i,j) = v(i,j)+dt*(-((uc*va-ue*vf)*rh+(vc^2-vd^2)*rh)...
            +rRe*((v(i+1,j)-2*v(i,j)+v(i-1,j))*rh2+(v(i,j+1)-2*v(i,j)+...
            v(i,j-1))*rh2));
    end
end

%%%%%%%%%%%    Boundary Conditions
%u_star velocity
u_star(:,N+2) = 2*U-u_star(:,N+1); %top wall
u_star(:,1) = -u_star(:,2); %bottom wall
u_star(1,:) = 0; %left wall
u_star(M+1,:) = 0; %right wall
%v_star velocity
v_star(:,N+1) = 0; %top wall
v_star(:,1) = 0; %bottom wall
v_star(1,:) = -v_star(2,:); %left wall
v_star(M+2,:) = -v_star(M+1,:); %right wall

%   Step II of Fractional Step Method

%%%%% GS Convergence Condition
%dynamically changing convergence criteria for Gauss-Seidel loop
```

```
    conv_gs = 1; %initial convergence value for Step II
    if conv*10^-2 >= 10^-3
        conv_gs_limit = 10^-3;
    elseif conv*10^-2 <= 10^-7
        conv_gs_limit = 10^-7;
    end

    %%%%% find solution to Langrange multiplier using Gauss-Seidel
    %iterates until Gauss-Seidel convergence criteria is met
    %pre-calculate right-hand sides of GS and residual equations for speed
    for j = 2:N+1
        for i = 2:M+1
            gs_RHS(i,j) = 0.25*(dx/dt)*(u_star(i,j)-u_star(i-
1,j)+v_star(i,j)-v_star(i,j-1));
            gs_RHS2(i,j) = (1/dt)*(((u_star(i,j)-u_star(i-
1,j))*rh)+((v_star(i,j)-v_star(i,j-1))*rh));
        end
    end

    while conv_gs > conv_gs_limit
        n_gs = n_gs+1; %gauss-seidel iteration counter
        n_gs_t = n_gs_t+1; %gauss-seidel iteration counter (total)

        %Gauss-Seidel
        for j = 2:N+1
            for i = 2:M+1
                phi(i,j) = 0.25*(phi(i-1,j)+phi(i+1,j)+phi(i,j-1)...
                    +phi(i,j+1))-gs_RHS(i,j);
            end
        end

        %%%%% update boundary conditions - Langrange multiplier ("pressure")
        phi(:,1) = phi(:,2); %bottom wall
        phi(:,N+2) = phi(:,N+1); %top wall
        phi(1,:) = phi(2,:); %left wall
        phi(M+2,:) = phi(M+1,:); %right wall


        %%%%% check convergence of Lagrange multiplier
        for j = 2:N+1
            for i = 2:M+1
                %Residual for entire mesh
                res_gs(i,j) = (phi(i+1,j)-2*phi(i,j)+phi(i-1,j))*rh2 ...
                +(phi(i,j+1)-2*phi(i,j)+phi(i,j-1))*rh2 ...
                -gs_RHS2(i,j);
            end
        end

        conv_gs = max(max(abs(res_gs))); %check

    end

    %   Step III of Fractional Step Method
```

```
    %%%%% find u velocity solution at n+1
    for j = 2:N+1
        for i = 2:M
            u(i,j) = u_star(i,j)-(dt*rh)*(phi(i+1,j)-phi(i,j));
        end
    end

    %%%%% find v velocity solution at n+1
    for j = 2:N
        for i = 2:M+1
            v(i,j) = v_star(i,j)-(dt*rh)*(phi(i,j+1)-phi(i,j));
        end
    end

    %%%%% update boundary conditions - velocities
    %u velocity
    u(:,N+2) = 2*U-u(:,N+1); %top wall
    u(:,1) = -u(:,2); %bottom wall
    u(1,:) = 0; %left wall
    u(M+1,:) = 0; %right wall
    %v velocity
    v(:,N+1) = 0; %top wall
    v(:,1) = 0; %bottom wall
    v(1,:) = -v(2,:); %left wall
    v(M+2,:) = -v(M+1,:); %right wall

    %%%%% convergence of final solution
    res_u = (u-r_u)/dt; %du/dt
    res_v = (v-r_v)/dt; %dv/dt
    conv_u = max(max(abs(res_u)));
    conv_v = max(max(abs(res_v)));
    conv = max([conv_u conv_v]);

    %   Get Ready for Next Iteration

    %%%%% convergence history
    conv_hist_u(n) = conv_u;
    conv_hist_v(n) = conv_v;

    %%%%% Output Running Update
    fprintf('Convergence of u = %6.8f, Convergence of v =
%6.8f\n',conv_u,conv_v);
    fprintf('Iteration %d, %d GS Iterations, %d Total GS
Iterations\n',n,n_gs,n_gs_t);
end

toc

%%

%%%%%%%%%%%    Plots

%%%%% vorticity
%calculate vorticity array
for j = 1:N+1
```

```
    for i = 1:M+1
        omega(i,j) = rh*(v(i+1,j)-v(i,j)-u(i,j+1)+u(i,j));
    end
end

%import ghia vorticity data
ghia_vor_6 = xlsread('ghia_vort.xlsx',1,'A3:B27');
ghia_vor_5 = xlsread('ghia_vort.xlsx',1,'C3:D25');
ghia_vor_4 = xlsread('ghia_vort.xlsx',1,'E3:F33');
ghia_vor_3 = xlsread('ghia_vort.xlsx',1,'G3:H39');
ghia_vor_2 = xlsread('ghia_vort.xlsx',1,'I3:J42');
ghia_vor_1 = xlsread('ghia_vort.xlsx',1,'K3:L41');
ghia_vor_0 = xlsread('ghia_vort.xlsx',1,'M3:N69');
ghia_vor_n1 = xlsread('ghia_vort.xlsx',1,'O3:P57');
ghia_vor_n2 = xlsread('ghia_vort.xlsx',1,'Q3:R37');
ghia_vor_n3 = xlsread('ghia_vort.xlsx',1,'S3:T31');
ghia_vor_n4 = xlsread('ghia_vort.xlsx',1,'U3:V28');

%plot vorticity
xy_vor = linspace(0,L,M+1);
con = [0.0 -0.5 0.5 -1.0 1.0 -2.0 2.0 -3.0 3.0 -4.0 -5.0]; %ghia vorticity
values
figure
contour(xy_vor,xy_vor,omega',con) %plots contour at ghia values
hold on
%plot ghia values
plot(ghia_vor_6(:,1),ghia_vor_6(:,2),'ok')
plot(ghia_vor_5(:,1),ghia_vor_5(:,2),'ok')
plot(ghia_vor_4(:,1),ghia_vor_4(:,2),'ok')
plot(ghia_vor_3(:,1),ghia_vor_3(:,2),'ok')
plot(ghia_vor_2(:,1),ghia_vor_2(:,2),'ok')
plot(ghia_vor_1(:,1),ghia_vor_1(:,2),'ok')
plot(ghia_vor_0(:,1),ghia_vor_0(:,2),'ok')
plot(ghia_vor_n1(:,1),ghia_vor_n1(:,2),'ok')
plot(ghia_vor_n2(:,1),ghia_vor_n2(:,2),'ok')
plot(ghia_vor_n3(:,1),ghia_vor_n3(:,2),'ok')
plot(ghia_vor_n4(:,1),ghia_vor_n4(:,2),'ok')
%labeling
title(sprintf('Vorticity Contours for %d x %d Mesh',M,N),'fontsize',14)
xlabel('x','fontsize',14); ylabel('y','fontsize',14)
legend('Fractional Step Method','Ghia Values')

%%%%% velocities along lines through geometric center
x = linspace(0,L,M+2); %horizontal line
y = linspace(0,L,N+2); %vertical line

for q = 1:N+2
    u_plot(q) = 0.5*(u(N/2,q)+u((N+2)/2,q)); %u along vertical line
    v_plot(q) = 0.5*(v(q,(N+2)/2)+v(q,N/2)); %v along horizontal line
end

%ghia values for u and v along same lines
u_ghia = xlsread('ghia_u&v.xlsx',1,'A3:B19');
v_ghia = xlsread('ghia_u&v.xlsx',1,'D3:E19');

%plot velocities
```

```
figure
plot(y,u_plot,'-r'); hold on
plot(u_ghia(:,1),u_ghia(:,2),'ok');
title('u-Velocity Along Vertical Line Through Geometric Center of
Cavity','fontsize',14);
xlabel('Vertical Line, y','fontsize',14); ylabel ('u-Velocity','fontsize',14)
legend(sprintf('%d x %d grid',M,N),'Ghia Values')

figure
plot(x,v_plot,'-r'); hold on
plot(v_ghia(:,1),v_ghia(:,2),'ok');
title('v-Velocity Along Horizontal Line Through Geometric Center of
Cavity','fontsize',14);
xlabel('Horizontal Line, x','fontsize',14); ylabel('v-
Velocity','fontsize',14)
legend(sprintf('%d x %d grid',M,N),'Ghia Values')

%plot convergence history for velocities
figure
plot(t_plot,log10(conv_hist_u),'-c'); hold on
plot(t_plot,log10(conv_hist_v),'-g');
title(sprintf('Convergence History for %d x %d Mesh',M,N),'fontsize',14)
xlabel('Time, s','fontsize',14); ylabel('Convergence, log
scale','fontsize',14);
legend('u-velocity','v-velocity')
```

4.2 GCI Accuracy Analysis

```
% MAE471 - CFD
% Final Project
% Solving incompressible, 2D Navier-Stokes for Lid-Driven Cavity

% Richard Extrapolation and GCI Analysis
% Determines accuracy for velocity solutions along geometric center lines

r = 2; %ratio between meshes, constant for 32 and 21

%u-velocity
f3u = 1.1089; %coarsest mesh, M = 32
f2u = 1.0523; %medium mesh, M = 64
f1u = 1.0258; %finest mesh, M = 128

p_u = log((f3u-f2u)/(f2u-f1u))/log(r); %order of convergence
f_u = f1u+((f1u-f2u)/(r^p_u-1)); %richardson extrapolation

GCI_12u = 1.25*(abs((f1u-f2u)/f1u)/(r^p_u-1));
GCI_23u = 1.25*(abs((f2u-f3u)/f2u)/(r^p_u-1));

fs_u = [f1u f2u f3u];

figure
plot([1 2 4],fs_u,'-or'); hold on
plot(0,f_u,'d');
title('Accuracy Analysis of u-Velocity Solution','fontsize',14);
ylabel('Maximum u-Velocity Along Vertical Center Line','fontsize',14);
xlabel('Normalized Grid Spacing','fontsize',14)
```

```
legend('Fractional Step Method','Richardson Extrapolation')

%v-velocity
f3v = 0.1675;
f2v = 0.1746;
f1v = 0.1775;

p_v = log((f3v-f2v)/(f2v-f1v))/log(r);
f_v = f1v+((f1v-f2v)/(r^p_v-1));

GCI_12v = 1.25*(abs((f1v-f2v)/f1v)/(r^p_v-1));
GCI_23v = 1.25*(abs((f2v-f3v)/f2v)/(r^p_v-1));

fs_v = [f1v f2v f3v];

figure
plot([1 2 4],fs_v,'-or'); hold on
plot(0,f_v,'d');
title('Accuracy Analysis of v-Velocity Solution','fontsize',14);
ylabel('Maximum v-Velocity Along Horizontal Center Line','fontsize',14);
xlabel('Normalized Grid Spacing','fontsize',14)
legend('Fractional Step Method','Richardson Extrapolation')
```

4.3 ANSYS/Ghia Plotter

```
% MAE471 - CFD
% Final Project
% Solving incompressible, 2D Navier-Stokes in lid-drive cavity

%Compares solution found using ANSYS Fluent to Ghia 1982

clear all
clc

%%%%%   Import Data
%import Ghia data
u_ghia = xlsread('ghia_ANSYS.xlsx',1,'E4:F20');
v_ghia = xlsread('ghia_ANSYS.xlsx',1,'B4:C20');

%import ANSYS data
u_ANSYS = xlsread('ghia_ANSYS.xlsx',1,'E24:F66');
v_ANSYS = xlsread('ghia_ANSYS.xlsx',1,'B24:C66');

%u-velocity plot
figure
plot(u_ANSYS(:,2),u_ANSYS(:,1),'-r'); hold on
plot(u_ghia(:,1),u_ghia(:,2),'ok')
title('u-Velocity Along Vertical Line Through Geometric Center of
Cavity','fontsize',14);
xlabel('Vertical Line, y','fontsize',14); ylabel ('u-Velocity','fontsize',14)
legend('ANSYS Fluent','Ghia Values')

%v-velocity plot
figure
plot(v_ANSYS(:,1),v_ANSYS(:,2),'-r'); hold on
plot(v_ghia(:,1),v_ghia(:,2),'ok')
```

```
title('v-Velocity Along Horizontal Line Through Geometric Center of
Cavity','fontsize',14);
xlabel('Horizontal Line, x','fontsize',14); ylabel('v-
Velocity','fontsize',14)
legend('ANSYS Fluent','Ghia Values')
```