MIE1210 FINAL PROJECT

2D Incompressible Navier-Stokes Solver

Kyu Mok Kim
998745381

Table of Contents

1. Introduction

The development and applications of computational fluid dynamics (CFD) in predicting internal and external flows has risen dramatically in the past decade. The foundation of the modern CFD comes from all the significant work from 18th and 19th century to describe the motion of fluids mathematically. Leonhard Euler (1707-1783) founded the Euler equations, which describes the conservation of momentum for an inviscid fluid and mass. Then, Claude Navier (1785-1836) and George Stokes (1819-1903) introduced viscosity into the Euler equations, resulting in the Navier-Stokes equation. The use of CFD eliminates the need to do experimental fluid dynamics and saves both costs and time, and also can be done in any physical condition. The main goal of CFD is to solve five governing equations with given boundary conditions. The five governing equations include: continuity equation, $\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{V}) = 0$, Navier stokes equation for all 3 directions, $\rho \frac{\partial \vec{V}}{\partial t} + \rho(\vec{V} \cdot \nabla)\vec{V} = -\nabla p + \rho \vec{g} + \nabla \cdot \tau_{ij}$, and energy equation $\frac{\partial \rho E}{\partial t} + (\rho u_\alpha E)_{,\alpha} = (u_\alpha \tau_{\alpha\beta})_{,\beta} + (kT_{,\alpha})_{,\alpha} + \rho u_\alpha f_\alpha^b + \rho q$. It is impossible to solve these equations analytically for most engineering problems. Therefore, CFD discretizes a given problem and solves the equations for each discretization.

When using CFD codes, there are three main elements: a pre-processor, a solver, and a post -processor. The pre-processor includes definition of geometry, grid generation, and the boundary conditions. Grid generation is important because the number of elements/nodes in a grid determines the accuracy and the speed/cost at which the CFD solves. This part of CFD code is where over 50% of the time is spent in industries on a CFD project. (H.K.Versteeg, 1995)This step is crucial as it determines the productivity, accuracy, and efficiency of a CFD code. In the solver part of the CFD code, there are three distinct streams of numerical solution techniques: finite difference, finite element, and spectral methods. They are different in the way in which the flow variables are approximated and how each node in the geometry is discretized. The most common method is the finite volume method which is used in four main commercially available CFD codes: PHOENICS, FLUENT, FLOW3D, and STAR-CD. In this method, governing equations are solved for each and every control volume (around the nodes in the grid) and discretized using integrated equation representing flow processes such as convection and diffusion. Then these algebraic equations from the integral equations are solved using iterative method.

In post-processing, many CFD codes and packages include data visualization tools such as geometry and grid display with contour plots showing flow conditions such as velocity vectors, temperature, and pressure. (H.K.Versteeg, 1995)

The method I will be using for this project is the finite volume method using SIMPLE algorithm with Python. The SIMPLE algorithm, or Semi-implicit method for pressure-linked equations, was founded by Patankar and Spalding (1972) and is an iterative method to guess and correct pressure on the staggered grid. The lid driven cavity problem will be solved using this method and will be compared with literature values from the paper from (U. Ghia, 1982).

2. Numerical Methodology

2.1 Nondimensionalization

For this project, 2D incompressible Navier-Stokes equation will be solved for various model problems. One of the governing equations for theses problems are the non-linear momentum equation:

$$\nabla \cdot \vec{u}\vec{u} = \nabla \cdot \nu \nabla \vec{u} - \frac{1}{\rho}\nabla p \quad \text{(Eq. 2.1.1)}$$

Where $\vec{u}$ is the velocity, ρ is the fluid density, ν = μ/ρ is the kinematic viscosity and p is the pressure. This equation can be nondimensionalized using scaling parameters as follows:

Scaling Parameter	Description	Primary Dimensions
L	Characteristic Length	{L}
V	Characteristic Speed	$\{Lt^{-1}\}$
P_0-P_∞	Reference Pressure Difference	$\{Lt^{-2}\}$

(Fig. 1.1.1)

The nondimensional variables can be defined as following using the table from Fig. 1.1.1:

$\vec{u}^* = \vec{u}/u$ (Eq. 2.1.2)

$P^* = (P-P_\infty)/(P_0-P_\infty)$ (Eq. 2.1.3)

$\nabla^* = L\nabla$ (Eq. 2.1.4)

Which can be rearranged to: $\vec{u} = u\vec{u}^*$ (Eq. 2.1.5)

$P = P_\infty + (P_0 - P_\infty)P^*$ (Eq. 2.1.6)

$\nabla = \nabla^*/L$ (Eq. 2.1.7)

Then these can be substituted in the Eq. 1.1.1 to obtain:

$$\nabla^* \vec{u}^* \vec{u}^* = [\mu/(\rho u L)] \nabla^{*2} \vec{u}^* - [(P_0-P_\infty)/(\rho u^2)] \nabla^* P^* \quad \text{(Eq. 2.1.8)}$$

We can see from Eq. 1.1.2 that Euler number, $Eu = (P_0-P_\infty)/(\rho u^2)$ and inverse of Reynolds number, $Re = \rho u L/\mu$, are present in the equation. Therefore, the final Navier-Stokes equation in nondimensional form would be:

$$\nabla^* \vec{u}^* \vec{u}^* = (1/Re) \nabla^{*2} \vec{u}^* - (Eu) \nabla^* P^* \quad \text{(Eq. 2.1.9)}$$

Then, these equations in 2D would be: $\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$ (Eq. 2.1.10) for continuity

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

(Eq. 2.1.11) for x-direction

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

(Eq. 2.1.12) for y-direction

2.2 Discretization

Solving Eq. 2.1.10 to 2.1.12 requires discretization due to non-linearity of the velocity and the need to couple pressure and velocity. To discretize the three equations, the both equations are integrated over the control volume of a cell. For a 2D grid, integrating Eq. 2.1.11 gives the following equation:

$$\iint_v \left[\frac{\partial}{\partial x}(u^2)+\frac{\partial}{\partial y}(uv)\right]dxdy=\frac{1}{Re}\iint_v\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)dxdy-\iint_v\frac{\partial}{\partial x}(p)dxdy \quad \text{(Eq. 2.2.1)}$$

or by approximation,

$$\frac{\Delta x\Delta y}{\Delta x}(u^2)+\frac{\Delta x\Delta y}{\Delta y}(uv)=\frac{\Delta x\Delta y}{Re}\left[\frac{1}{\Delta x}\left(\frac{\partial u}{\partial x}\right)+\frac{1}{\Delta y}\left(\frac{\partial u}{\partial y}\right)\right]-\left(\frac{\partial p}{\partial x}\right)\Delta x\Delta y \quad \text{(Eq. 2.2.2)}$$

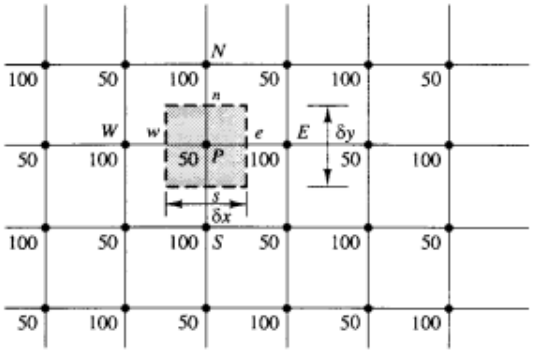
In a hypothetical grid/discretization shown in Fig. 2.2.1 below, the pressure is not properly represented in the discretized momentum equation due to the central differencing of the pressure gradient formula

$$\frac{\partial p}{\partial x}=\frac{p_e-p_w}{\delta x}=\frac{\left(\frac{p_E+p_P}{2}\right)-\left(\frac{p_P+p_W}{2}\right)}{\delta x}=\frac{p_E-p_W}{2\delta x} \quad \text{(Eq. 2.2.3)}$$

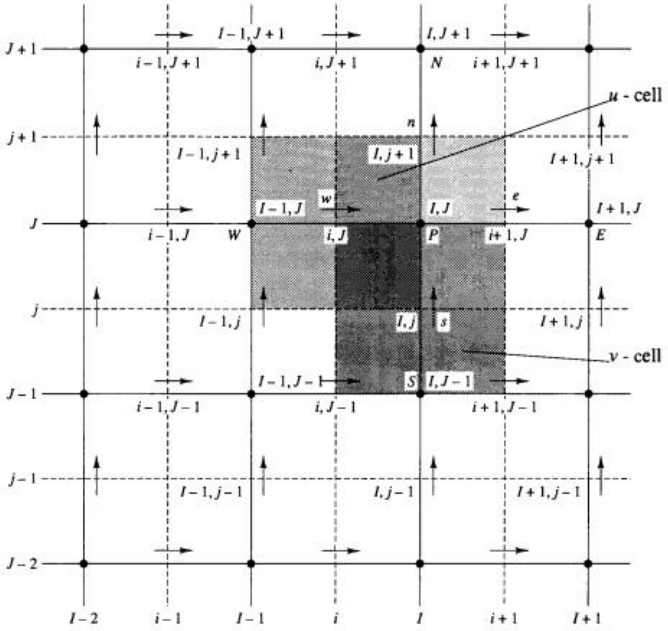
(it will be zero all around): for u- momentum and

$$\frac{\partial p}{\partial y}=\frac{p_N-p_S}{2\delta y} \quad \text{(Eq.2.2.4)}$$

for v-momentum. Therefore, a staggered grid must be used, shown in Fig 2.2.2 below.



(Fig. 2.2.1)



(Fig. 2.2.2)

As seen in Fig. 2.2.2, the nodes marked with dots represents the pressure, horizontal arrows represent the u-velocities, and v-velocities for vertical arrows.

In this staggered grid arrangement, the pressure gradient term is given by: $\frac{\partial p}{\partial x} = \frac{p_P - p_W}{\delta x_u}$ (Eq. 2.2.5) for u-control volume and $\frac{\partial p}{\partial y} = \frac{p_P - p_S}{\delta y_v}$ (Eq. 2.2.6) for v-control volume. Then, in this staggered grid, Eq.2.2.2 can be written as following:

$$[(u^2)\Delta y]_w^e - [(uv)\Delta x]_s^n = \left[\frac{\Delta y}{Re}\left(\frac{\partial u}{\partial x}\right)\right]_w^e + \left[\frac{\Delta x}{Re}\left(\frac{\partial u}{\partial y}\right)\right]_s^n - \left(\frac{\partial p}{\partial x}\right)\Delta x \Delta y \quad \text{(Eq. 2.2.7)}$$

The diffusive flux in and out of the control volume will be approximating by using the central difference method (Eq. 2.2.8):

$$\left[\frac{\Delta y}{Re}\left(\frac{\partial u}{\partial x}\right)\right]_w^e + \left[\frac{\Delta x}{Re}\left(\frac{\partial u}{\partial y}\right)\right]_s^n = \left(\frac{\Delta y}{Re}\right)_e\left(\frac{u_E - u_P}{\Delta x}\right) - \left(\frac{\Delta y}{Re}\right)_w\left(\frac{u_P - u_W}{\Delta x}\right) + \left(\frac{\Delta x}{Re}\right)_n\left(\frac{u_N - u_P}{\Delta y}\right) - \left(\frac{\Delta x}{Re}\right)_s\left(\frac{u_P - u_S}{\Delta y}\right)$$

Through linearization, the diffusive equation becomes:

$$\left[\frac{\Delta y}{Re}\left(\frac{\partial u}{\partial x}\right)\right]_w^e + \left[\frac{\Delta x}{Re}\left(\frac{\partial u}{\partial y}\right)\right]_s^n = a_e^d u_E + a_w^d u_W + a_n^d u_N + a_s^d u_S - a_p^d u_P \quad \text{(Eq. 2.2.9)}$$

Where $a_e^d = \Delta y/(\Delta x Re)_e$, $a_w^d = \Delta y/(\Delta x Re)_w$, $a_n^d = \Delta x/(\Delta y Re)_n$, $a_s^d = \Delta x/(\Delta y Re)_s$, and $a_p^d = a_e^d + a_w^d + a_n^d + a_s^d$ Similarly, convective terms can be approximated as following:

$$[(u^2)\Delta y]_w^e - [(uv)\Delta x]_s^n = a_e^c u_E + a_w^c u_W + a_n^c u_N + a_s^c u_S - a_p^c u_P \quad \text{(Eq. 2.2.10)}$$

Where $a_e^c = (u\Delta y)_e$, $a_w^c = (u\Delta y)_w$, $a_n^c = (v\Delta x)_n$, $a_s^c = (v\Delta x)_s$, and $a_p^d = a_e^c + a_w^c + a_n^c + a_s^c$

Therefore, diffusive-convective term becomes: $a_p u_P = a_e u_E + a_w u_W + a_n u_N + a_s u_S$ (Eq. 2.2.11)

Where the "a" coefficients are the sum of the diffusive and convective coefficients (e.g. $a_e = a_e^d + a_e^c$)

To approximate this coefficient, several methods are available as shown in figure below (taken from (H.K.Versteeg, 1995)). Hybrid scheme will be used in this project.

	Scheme		
	Forward	Central	Hybrid
a_e	$a_e^d + \max[0, -a_e^c]$	$a_e^d - \frac{a_e^c}{2}$	$\max\left[-a_e^c, \left(a_e^d - \frac{a_e^c}{2}\right), 0\right]$
a_w	$a_w^d + \max[0, a_w^c]$	$a_w^d + \frac{a_w^c}{2}$	$\max\left[a_w^c, \left(a_w^d + \frac{a_w^c}{2}\right), 0\right]$
a_n	$a_n^d + \max[0, -a_n^c]$	$a_n^d - \frac{a_n^c}{2}$	$\max\left[-a_n^c, \left(a_n^d - \frac{a_n^c}{2}\right), 0\right]$
a_s	$a_s^d + \max[0, a_s^c]$	$a_s^d + \frac{a_s^c}{2}$	$\max\left[a_s^c, \left(a_s^d + \frac{a_s^c}{2}\right), 0\right]$
a_p	$a_e + a_w + a_n + a_s + (a_e^c - a_w^c + a_n^c - a_s^c)$	$a_e + a_w + a_n + a_s + (a_e^c - a_w^c + a_n^c - a_s^c)$	$a_e + a_w + a_n + a_s + (a_e^c - a_w^c + a_n^c - a_s^c)$

Substituting the pressure equation, Eq. 2.2.3 and Eq. 2.2.4, momentums equation can be discretized as following: $$a_{i,J}u_{i,J} = \sum a_{nb}u_{nb} - \frac{p_{I,J} - p_{I-1,J}}{\delta x_u}\Delta V_u + \bar{S}\,\Delta V_u$$ (Eq. 2.2.12), or

$$a_{i,J}u_{i,J} = \sum a_{nb}u_{nb} + (p_{I-1,J} - p_{I,J})A_{i,J} + b_{i,J}$$ (Eq. 2.2.13) for u-momentum, and similarly,

$$a_{I,j}v_{I,j} = \sum a_{nb}v_{nb} + (p_{I,J-1} - p_{I,J})A_{I,j} + b_{I,j}$$ (Eq. 2.2.14) for v-momentum.

2.3 Rhie-Chow Interpolation

To determine the expressions for the face velocities, the momentum interpolation or Rhie-Chow interpolation is used to approximate the derivatives. As seen from the Eq. 2.2.13 and Eq. 2.2.14, each cell has discretized equation in the form of: $$a_p\vec{v}_P = \sum_{neighbours} a_l\vec{v}_l - \frac{\nabla p}{V}$$ (Eq. 2.3.1), and for continuity,

$$\sum_{faces}\left[\frac{1}{a_p}H\right]_{face} = \sum_{faces}\left[\frac{1}{a_p}\frac{\nabla p}{V}\right]_{face}$$ (Eq. 2.3.2), where $$H = \sum_{neighbours} a_l\vec{v}_l$$. This interpolation of variables H and ∇p based on coefficients a_p for pressure velocity coupling is called Rhie-Chow interpolation. (CFDOnline, n.d.) This is shown in SIMPLE algorithm in the next section.

2.4 SIMPLE Method

For this project, Semi-Implicit Method for Pressure-Linked Equations method, or SIMPLE method, will be used to solve the problems. It starts with guessing of a pressure field p^* and the discretized momentum equations Eq. 1.2.7 and Eq. 1.2.8 are solved using this pressure field to obtain velocity components u^* and v^* as following:

$$a_{i,J}u^*_{i,J} = \sum a_{nb}u^*_{nb} + (p^*_{I-1,J} - p^*_{I,J})A_{i,J} + b_{i,J}$$ (Eq. 2.4.1) for u-momentum and

$$a_{I,j}v^*_{I,j} = \sum a_{nb}v^*_{nb} + (p^*_{I,J-1} - p^*_{I,J})A_{I,j} + b_{I,j}$$ (Eq. 2.4.2) for v-momentum

Then, correction term p' is defined as $$p = p^* + p'$$ (Eq. 2.4.3) and correction terms for the velocities as well as following: $$u = u^* + u'$$ (Eq. 2.4.4) and $$v = v^* + v'$$ (Eq. 2.4.5)

Subtracting guessed momentum equations (Eq. 2.4.1 and Eq. 2.4.2) from the correct momentum equations (Eq. 2.2.7 and Eq. 2.2.8) gives the following:

$$a_{i,J}(u_{i,J} - u^*_{i,J}) = \sum a_{nb}(u_{nb} - u^*_{nb}) + \left[(p_{I-1,J} - p^*_{I-1,J}) - (p_{I,J} - p^*_{I,J})\right]A_{i,J}$$ (Eq. 2.4.6) for u-momentum

$$a_{I,j}(v_{I,j} - v^*_{I,j}) = \sum a_{nb}(v_{nb} - v^*_{nb}) + \left[(p_{I,J-1} - p^*_{I,J-1}) - (p_{I,J} - p^*_{I,J})\right]A_{I,j}$$ (Eq. 2.4.7) for v-momentum

and using these equation with Eq. 2.4.3, Eq. 2.4.4 and Eq. 2.4.5, following equations are obtained:

$$a_{i,J}u'_{i,J} = \sum a_{nb}u'_{nb} + \left(p'_{I-1,J} - p'_{I,J}\right)A_{i,J}$$ (Eq. 2.4.8) for u-momentum

$$a_{I,j}v'_{I,j} = \sum a_{nb}v'_{nb} + \left(p'_{I,J-1} - p'_{I,J}\right)A_{I,j}$$ (Eq. 2.4.9) for v-momentum

In SIMPLE algorithm, $\sum a_{nb}u'_{nb}$ and $\sum a_{nb}v'_{nb}$ are omitted for approximation. This gives the following formulas:

$$u'_{i,J} = d_{i,J}\left(p'_{I-1,J} - p'_{I,J}\right)$$ (Eq. 2.4.10) for u-momentum and

$$v'_{I,j} = d_{I,j}\left(p'_{I,J-1} - p'_{I,J}\right)$$ (Eq. 2.4.11) for v-momentum, where $d_{i,J} = \frac{A_{i,J}}{a_{i,J}}$ and $d_{I,j} = \frac{A_{I,j}}{a_{I,j}}$.

Then, the equations Eq. 2.4.10 and Eq. 2.4.11, which describe the corrections can be applied to Eq.2.4.4. and Eq. 2.4.5., which gives:

$$u_{i,J} = u^*_{i,J} + d_{i,J}\left(p'_{I-1,J} - p'_{I,J}\right)$$ (Eq. 2.4.12) for u-momentum, and

$$v_{I,j} = v^*_{I,j} + d_{I,j}\left(p'_{I,J-1} - p'_{I,J}\right)$$ (Eq. 2.4.13) for v-momentum

Similarly, for $u_{i+1,J}$ and $v_{I,j+1}$:

$$u_{i+1,J} = u^*_{i+1,J} + d_{i+1,J}\left(p'_{I,J} - p'_{I+1,J}\right)$$ (Eq. 2.4.14), $$v_{I,j+1} = v^*_{I,j+1} + d_{I,j+1}\left(p'_{I,J} - p'_{I,J+1}\right)$$ (Eq. 2.4.15)

where $d_{i+1,J} = \frac{A_{i+1,J}}{a_{i+1,J}}$ and $d_{I,j+1} = \frac{A_{I,j+1}}{a_{I,j+1}}$.

The discretization of continuity equation gives

$$\left[(\rho uA)_{i+1,J} - (\rho uA)_{i,J}\right] + \left[(\rho vA)_{I,j+1} - (\rho vA)_{I,j}\right] = 0$$ (Eq. 2.4.16)

Substituting the corrected velocities equations Eq. 2.4.12 to Eq. 2.4.15 into the equation Eq. 2.4.16,

$$\begin{aligned}&\left[\rho_{i+1,J}A_{i+1,J}\left(u^*_{i+1,J} + d_{i+1,J}\left(p'_{I,J} - p'_{I+1,J}\right)\right)\right.\\&\quad\left.-\rho_{i,J}A_{i,J}\left(u^*_{i,J} + d_{i,J}\left(p'_{I-1,J} - p'_{I,J}\right)\right)\right]\\&\quad+\left[\rho_{I,j+1}A_{I,j+1}\left(v^*_{I,j+1} + d_{I,j+1}\left(p'_{I,J} - p'_{I,J+1}\right)\right)\right.\\&\quad\left.-\rho_{I,j}A^*_{I,j}\left(v^*_{I,j} + d_{I,j}\left(p'_{I,J-1} - p'_{I,J}\right)\right)\right] = 0\end{aligned}$$ (Eq. 2.4.17), or re-arranged to give

$$\left[(\rho dA)_{i+1,J} + (\rho dA)_{i,J} + (\rho dA)_{I,J+1} + (\rho dA)_{I,j}\right]p'_{I,J}$$
$$= (\rho dA)_{i+1,J}p'_{I+1,J} + (\rho dA)_{i,J}p'_{I-1,J} + (\rho dA)_{I,J+1}p'_{I,J+1}$$
$$+ (\rho dA)_{I,j}p'_{I,J-1}$$
$$+ \left[(\rho u^* A)_{i,J} - (\rho u^* A)_{i+1,J} + (\rho v^* A)_{I,j} - (\rho v^* A)_{I,J+1}\right] \quad \text{(Eq. 2.4.18)}$$

Then, identifying the coefficients of p' gives:

$$a_{I,J}p'_{I,J} = a_{I+1,J}p'_{I+1,J} + a_{I-1,J}p'_{I-1,J} + a_{I,J+1}p'_{I,J+1} + a_{I,J-1}p'_{I,J-1} + b'_{I,J} \quad \text{(Eq. 2.4.19) where}$$

$a_{I,J} = a_{I+1,J} + a_{I-1,J} + a_{I,J+1} + a_{I,J-1}$ (Eq. 2.4.20), where the coefficients are shown below:

$a_{I+1,J}$	$a_{I-1,J}$	$a_{I,J+1}$	$a_{i,J-1}$	$b'_{I,J}$
$(\rho dA)_{i+1,J}$	$(\rho dA)_{i,J}$	$(\rho dA)_{I,J+1}$	$(\rho dA)_{I,j}$	$\left(\rho u^* A\right)_{i,J} - \left(\rho u^* A\right)_{i+1,J} + \left(\rho v^* A\right)_{I,j} - \left(\rho v^* A\right)_{I,J+1}$

Fig. 2.4.1

After the p' equation, Eq. 2.4.19, the correct pressure is obtained using Eq. 2.4.3, and also velocity components using Eq. 2.4.12 to Eq. 2.4.15.

For convergence, relaxation factor must be used during the iterative process for new, improved pressures p^{new} using: $p^{new}=p^*+\alpha_p p'$ (Eq. 2.4.21), where α_p is the pressure relaxation factor. Under relaxation between 0 and 1 must be used to ensure stable computation. Similarly, velocities are also under-relaxed using:

$$u^{new} = \alpha_u u + (1-\alpha_u)u^{(n-1)} \quad \text{(Eq. 2.4.22)}$$

and $$v^{new} = \alpha_v v + (1-\alpha_v)v^{(n-1)} \quad \text{(Eq. 2.4.23)}$$

The process of SIMPLE method is summarized on the right figure taken from (H.K.Versteeg, 1995).

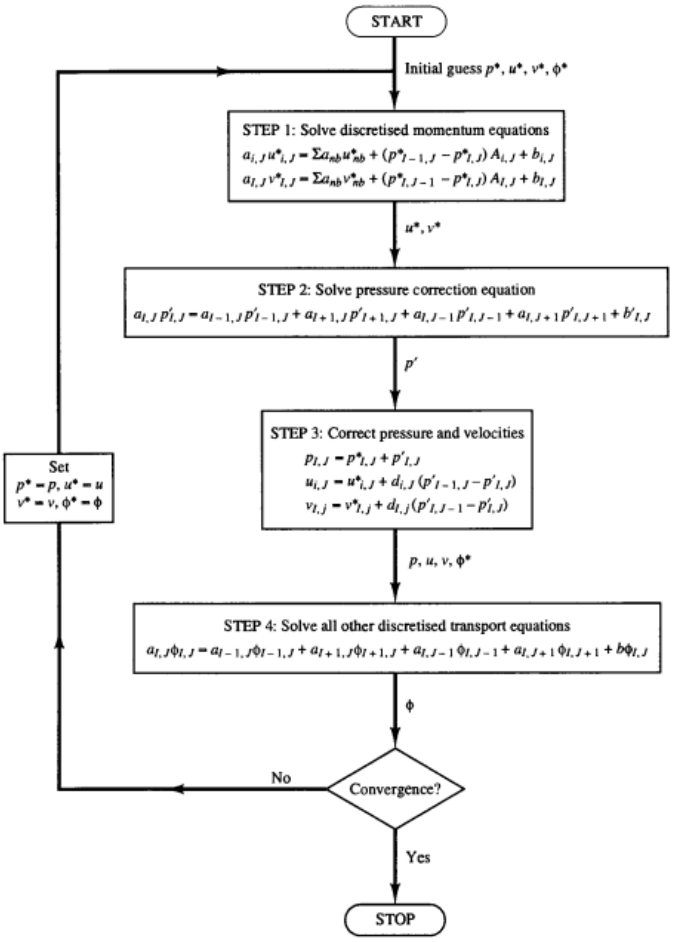


Fig. 2.4.2

2.5 Using Python

For this project, Python is used to code the SIMPLE algorithm. It is attached with this report as "SIMPLE-lid-driven-cavity FINAL.py". Refer to the file for the complete codes. I have used iterative method for all the steps and have stored all iterations to do a convergence test at the end.

For solving u and v momentum equations, Central Difference method was used to calcualte convective coefficients, then hybrid method was used to calculate the velocity coefficients. These momentum equations with these coefficients were solved iteratively, and error for each iteration is stored for graphing during post processing step. The maximum error for u and v momentum equation solution is set to 10^-3 with maximum iteration of 100. This is shown in figure on the right. (Fig. 2.5.1)

```
#step1: solving momentum equations
for i in range (1,nx):
    for j in range(1,ny+1):
        #convective coefficients CD method
        uace=dy*(uOld[i,j]+uOld[i+1,j])/2
        uacw=dy*(uOld[i,j]+uOld[i-1,j])/2
        uacn=dx*(vOld[i,j]+vOld[i+1,j])/2
        uacs=dx*(vOld[i,j-1]+vOld[i+1,j-1])/2
        #coefficients using hybrid
        udx=dy/dx/Re
        udy=dx/dy/Re
        uae[i,j]=max(-uace,(udx-uace/2),0)
        uaw[i,j]=max(uacw,(udx+uacw/2),0)
        uan[i,j]=max(-uacn,(udy-uacn/2),0)
        uas[i,j]=max(uacs,(udy+uacs/2),0)
        uap[i,j]=(uae[i,j]+uaw[i,j]+uan[i,j]+uas[i,j])/rel
        du[i,j]=dy/uap[i,j]
#putting previous u as new guess
for i in range (1,nx+2):
    for j in range(1,ny+3):
        us[i-1,j-1]=uOld[i-1,j-1]
m=0
err=1
while (err>merr):
    m+=1
    print (n,'-U Iteration#:',m)
    for i in range(1,nx):
        for j in range(1,ny+1):
            us[i,j]=(uae[i,j]*us[i+1,j]+uaw[i,j]*us[i-1,j]+
    if m>1:
        err=error(us,usOld,us,usOld,nx,ny)
        print (n,'-u Error:', err)
        if n>10:
            uerr=np.append(uerr,err)
    for i in range(1,nx):
        for j in range(1,ny+1):
            usOld[i,j]=us[i,j]
    if m>miter:
        break
```

(Fig. 2.5.1)

Similarly, pressure correction equation was also solved using iterative method, with the same maximum error and iteration as u and v momentum solutions. Error for these were stored as well.

Then, the pressure and velocities were corrected and sent back to step 1 to loop and iterative until convergence is met in step 4 as shown in Fig. 2.5.2.

This style of coding of seperating the SIMPLE algorithm with the problem allows flexibility to apply this code to any problems. To solve different problems, one would just need to change boundary conditions depending on the problem and can be solved with ease. The error for each iteration for overall SIMPLE algorithm is also stored and graphed later for convergence test. The maximum acceptable error was se to 10^-5, with maximum iteration of 2000.

```
#step 3:correcting pressure and velocities
for i in range(1,nx):
    for j in range (1,ny+1):
        up[i,j]=du[i,j]*(pp[i,j]-pp[i+1,j])
for i in range (1,nx+1):
    for j in range (1,ny):
        vp[i,j]=dv[i,j]*(pp[i,j]-pp[i,j+1])
for i in range (1,nx+1):
    for j in range (1,ny+1):
        p[i,j]=ps[i,j]+pp[i,j]*relp
for i in range(1,nx):
    for j in range (1,ny+1):
        u[i,j]=us[i,j]+up[i,j]
for i in range (1,nx+1):
    for j in range (1,ny):
        v[i,j]=vs[i,j]+vp[i,j]

#step 4: check for convergence
if n>10:
    err=error(u,uOld,v,vOld,nx,ny)
    print (n,'-SIMPLE Error is:', err)
    serr=np.append(serr,err)

#step 5: loop
for i in range (1,nx+2):
    for j in range(1,ny+3):
        uOld[i-1,j-1]=u[i-1,j-1]
for i in range (1,nx+3):
    for j in range(1,ny+2):
        vOld[i-1,j-1]=v[i-1,j-1]
for i in range (1,nx+3):
    for j in range (1,ny+3):
        ps[i-1,j-1]=p[i-1,j-1]
```

(Fig. 2.5.2)

3. Results and Discussion

3.1 Lid Driven Cavity

The lid driven cavity is a problem where there is an incompressible Newtonian fluid in a 2D cavity with stationary walls on the right, left, and bottom, with moving wall at the top. This is shown on the right figure Fig.3.1.1. Reynolds number for this problem is defined as:

$$Re=uL/v \qquad \text{(Eq. 3.1.1)}$$

where u= lid velocity, L=height/width of cavity, and v=kinematic viscosity. Since we're using nondimensionalization, all the variables u, L, v=1.

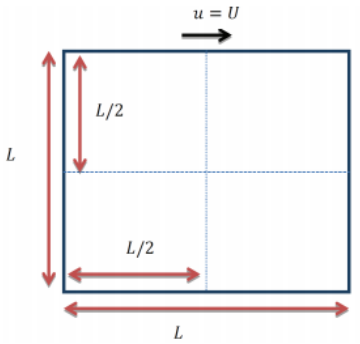


Fig. 3.1.1

The boundary condition for the walls will have u and v velocity of zero, while the pressure gradient on the walls will be zero also.

The following runs were done in python:

Mesh Size	Iterations	Time Taken	Comments
65 by 65	689	350 s (~6 min)	The run successfully converged. Accurate solution is expected.
129 by 129	2000	3803 s (~1 hr)	Since the iteration stopped before converging, due to the max of 2000 iterations, it is expected to have some errors.
257 by 257	2000	17911 s (~5 hrs)	It also did not converge yet due to the limit, and therefore, it is expected to have more errors than 129 by 129.

Fig. 3.1.2.

As shown above, due to computing power and the length of time to converge, I have done a run in coarser mesh, 65 by 65, to efficiently compare results with that of Ghia's paper. It is expected to have some errors for the finer meshes because the iteration was stopped before converging, due to taking too long of time. This error is shown in next sections where I present the results.

3.2 Contour Plots of Velocity and Pressure

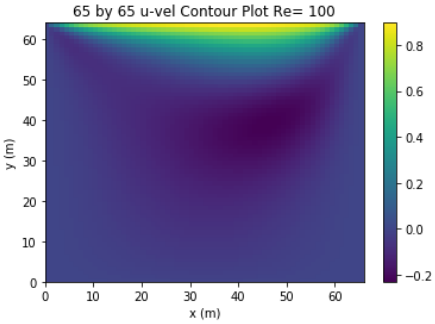
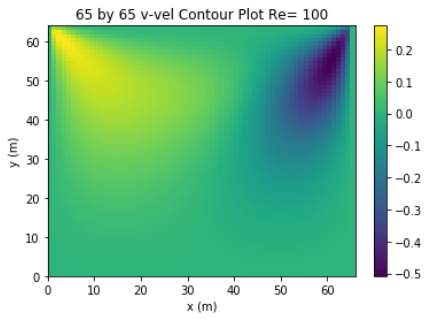
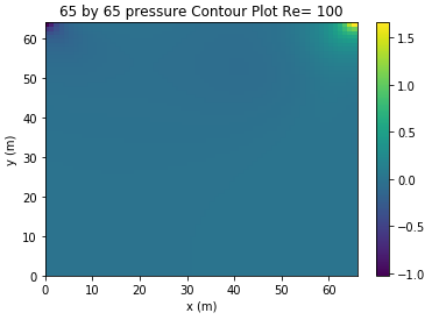
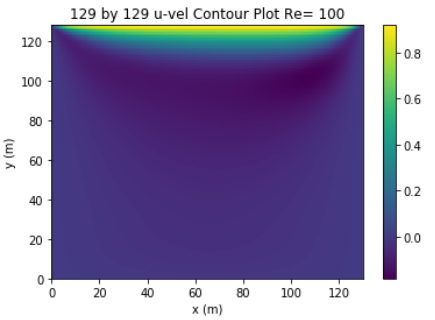
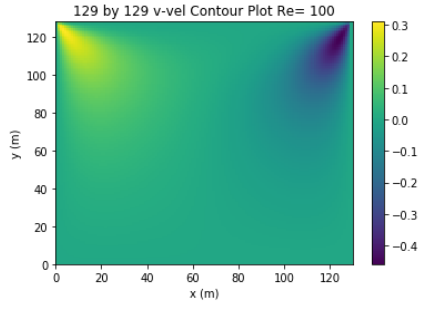
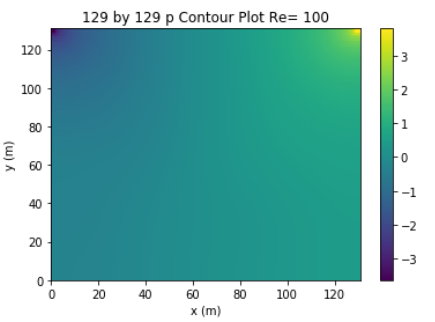
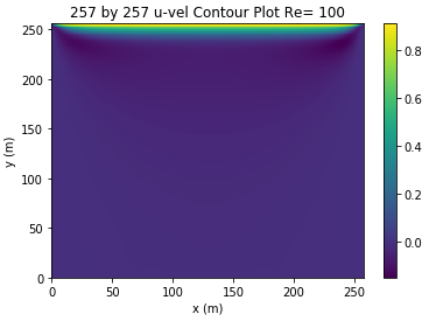
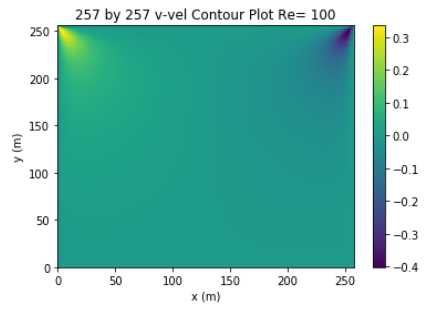
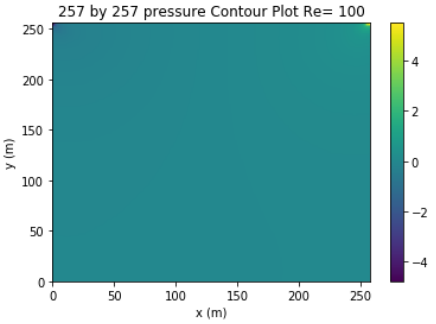
Size	U-Velocity Contour	V-Velocity Contour	Pressure Contour
65x65			
129x129			
257x257			

Fig. 3.2.1

As seen in above figure, the results for 65 by 65 mesh looks more accurate as it went through complete convergence as opposed to other mesh sizes, which did not converge completely yet.

3.3 Flow Streamlines

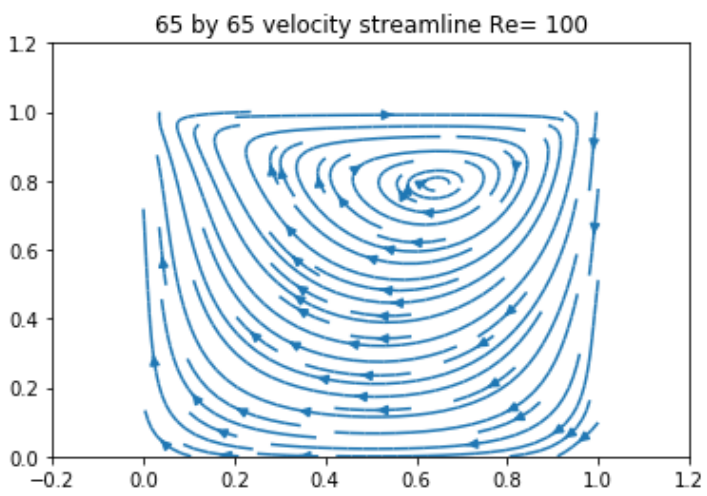
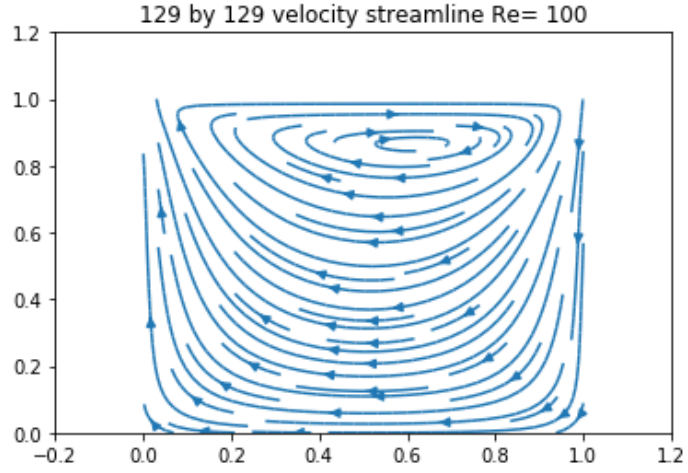
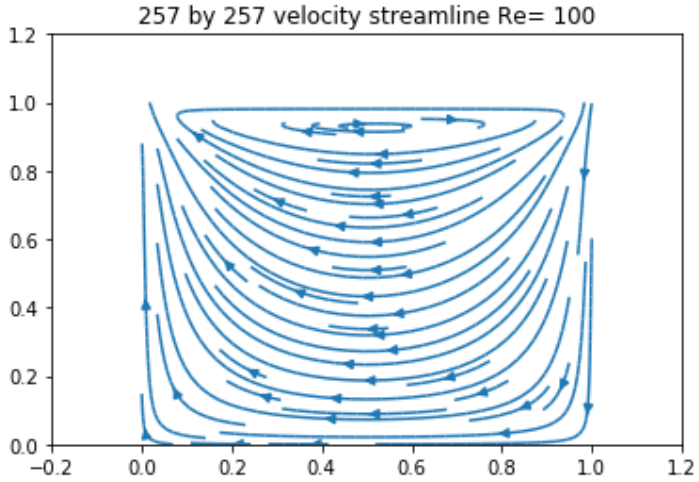
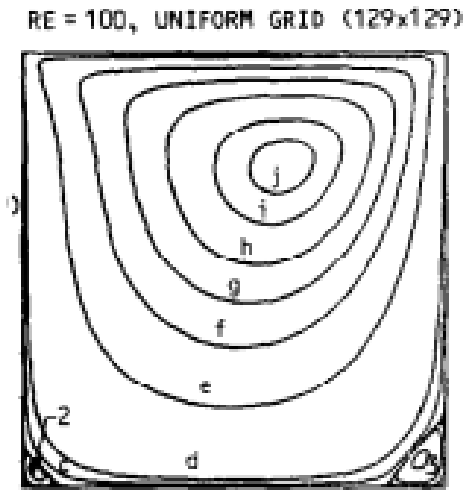
65 by 65	129 by 129
	
257 by 257	**Ghia 129 by 129**
	

Fig. 3.3.1

As expected, the flow for 65 by 65 looks the closest to Ghia's as it went through complete convergence. The two secondary flows on the bottom right and left are too small to be seen in the flow streams. The reason why it can be seen in Ghia's is because the method used in Ghia's paper is finite difference method as opposed to finite volume method used here. The finite difference method results in more accurate results and the two flows on the corners can't be seen in finite volume method unless I increase the Reynold's number, or zoom into the corners. This is discussed in detail in the conclusion section.

3.4 Plot of v-velocity along a horizontal line passing through the center of the cavity

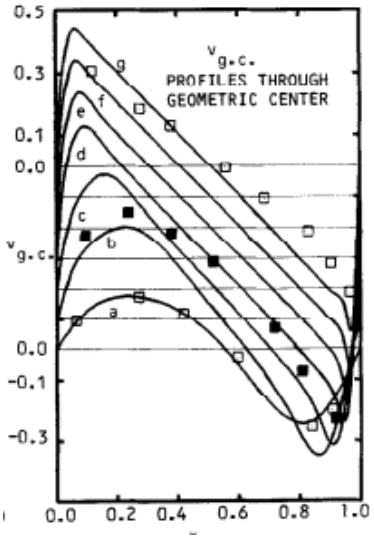


Fig. 3.4.1

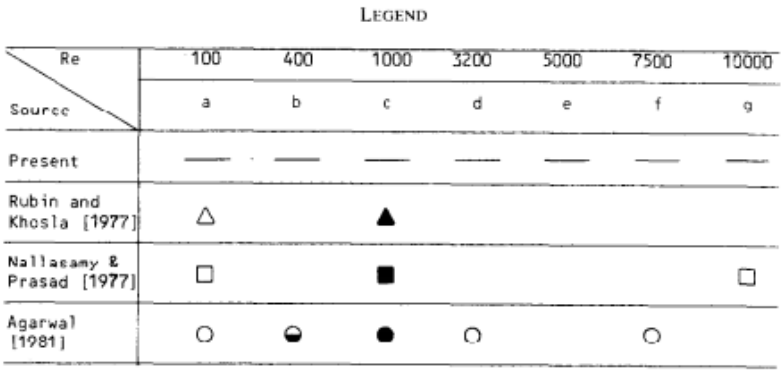


Fig.3.4.2

The v-velocity is graphed along the horizontal line passing through the geometric center of the cavity. This is compared with the graph from Ghia's paper shown in above figures. To compare with accuracy, the exact points from the graph (shown in Fig. 3.4.3 below) are taken and graphed with my results. The results are shown on the next page as Fig.3.4.4.

Results for v-Velocity along Horizontal Line through Geometric Center of Cavity

129-grid pt. no.	x	Re 100	400	1000	3200	5000	7500	10,000
129	1.0000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
125	0.9688	–0.05906	–0.12146	–0.21388	–0.39017	–0.49774	–0.53858	–0.54302
124	0.9609	–0.07391	–0.15663	–0.27669	–0.47425	–0.55069	–0.55216	–0.52987
123	0.9531	–0.08864	–0.19254	–0.33714	–0.52357	–0.55408	–0.52347	–0.49099
122	0.9453	–0.10313	–0.22847	–0.39188	–0.54053	–0.52876	–0.48590	–0.45863
117	0.9063	–0.16914	–0.23827	–0.51550	–0.44307	–0.41442	–0.41050	–0.41496
111	0.8594	–0.22445	–0.44993	–0.42665	–0.37401	–0.36214	–0.36213	–0.36737
104	0.8047	–0.24533	–0.38598	–0.31966	–0.31184	–0.30018	–0.30448	–0.30719
65	0.5000	0.05454	0.05186	0.02526	0.00999	0.00945	0.00824	0.00831
31	0.2344	0.17527	0.30174	0.32235	0.28188	0.27280	0.27348	0.27224
30	0.2266	0.17507	0.30203	0.33075	0.29030	0.28066	0.28117	0.28003
21	0.1563	0.16077	0.28124	0.37095	0.37119	0.35368	0.35060	0.35070
13	0.0938	0.12317	0.22965	0.32627	0.42768	0.42951	0.41824	0.41487
11	0.0781	0.10890	0.20920	0.30353	0.41906	0.43648	0.43564	0.43124
10	0.0703	0.10091	0.19713	0.29012	0.40917	0.43329	0.44030	0.43733
9	0.0625	0.09233	0.18360	0.27485	0.39560	0.42447	0.43979	0.43983
1	0.0000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

Fig. 3.4.3

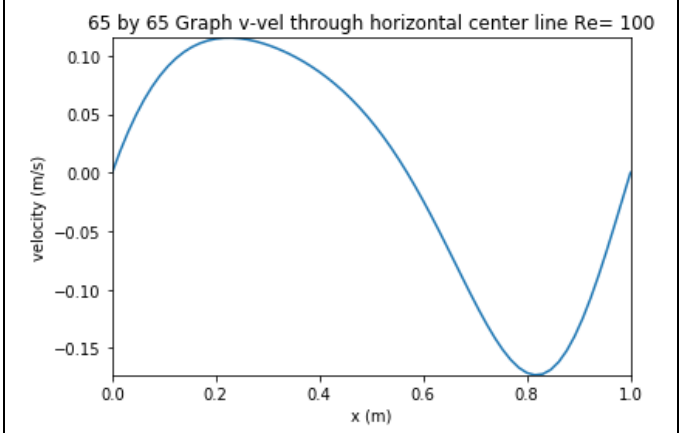
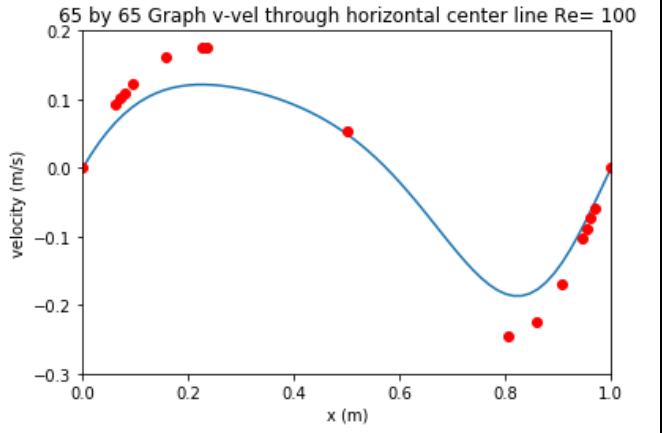
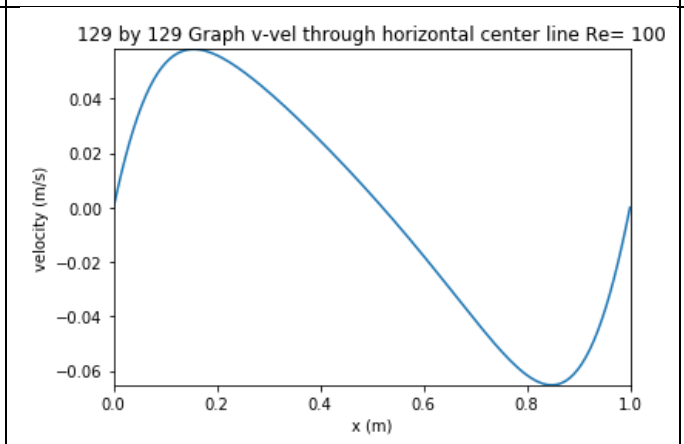
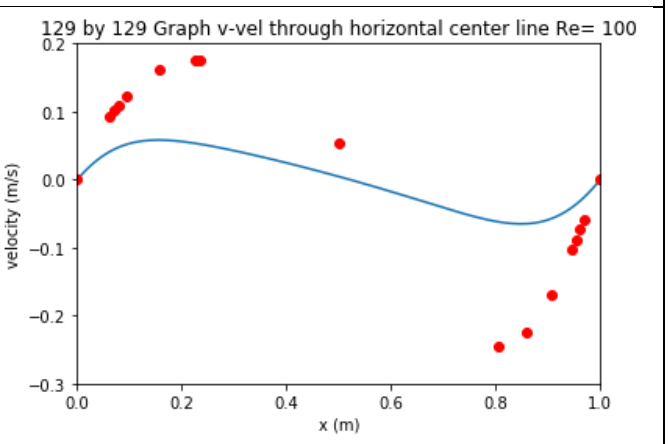
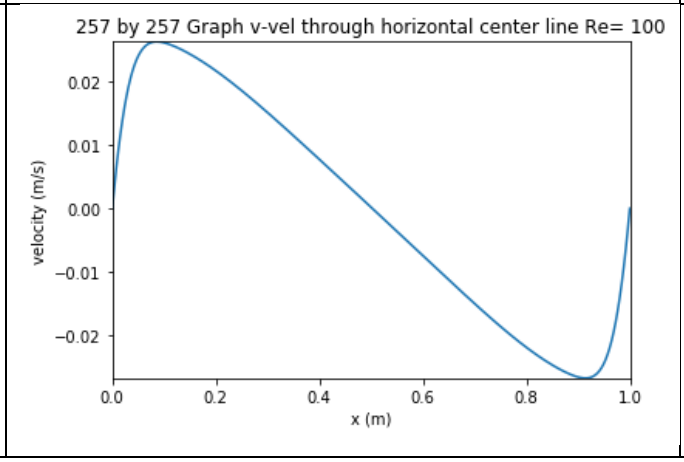
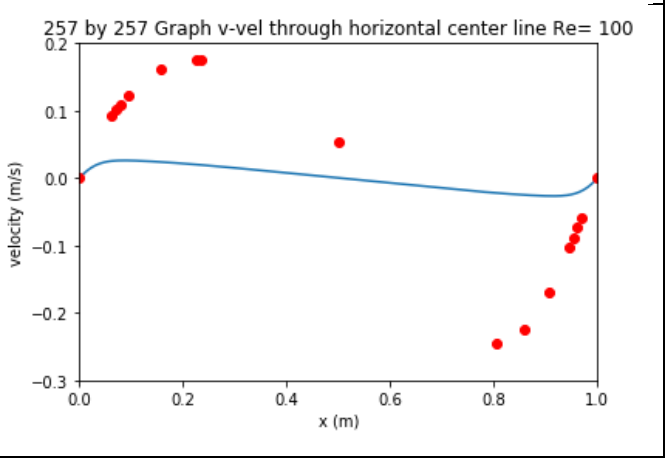
Size	Velocity through Center	Compared with Ghia
65x65		
129x129		
257x257		

Fig. 3.4.4

As shown above, the results for 65 by 65 matches with Ghia's Paper the most. This is expected as 65 by 65 mesh was the only size that went through complete convergence. The small difference in the points are due to difference in mesh size (65 by 65 and 129 by 129) and difference in method used, more explained in the conclusion.

4. Conclusion

For this project, finite volume method was used with SIMPLE algorithm. In finite volume method, it assumes that the overall mass, momentum and other parameters are conserved over a volume. It is seen to be a uniform weight function over the cell and zero everywhere else. Its accuracy is sensitive to the quality of the mesh and may need refining of the mesh volume or cells. As seen in this project, for 257*257 mesh or 66049 nodes have taken great amount of time and would not be practical for industrial geometries and boundary conditions, as there are much more nodes in 3D. Therefore, coarse finite volume method should be used for efficiency.

The paper (U. Ghia, 1982) uses finite difference method, which covers every term generated from the coordinate transformation. It uses the weighted residual method with the Dirac Delta as weight function at the node point and zero everywhere else. Therefore, in general, the finite difference solution is always more accurate than the finite volume solution since for finite volume method, overall conservation is conserved, which means the accuracy of the flow variables are sacrificed: for finite volume, convective and pressure terms involve interpolation to faces while the diffusion terms involve first derivative approximations, while the finite difference uses second derivative approximations.

On the other hand, the integral conservation equations when expressed in conservation form in finite volume method can be converted from volume integrals to surface integrals using Gauss Divergence Theorem. This control volume analysis can be easily interpreted and troubleshooted, which makes the finite volume more favorable than the finite difference method in industries.

To make improvements to this project, the other variations of SIMPLE can be used which have improved convergence, and therefore, saves computational effort and time. Other SIMPLE algorithms include: SIMPLEC, SIMPLER, and collected SIMPLE. For instance, as seen in the convergence study in the Appendix, it can be seen that error for pressure equation is high compared to the velocity equations. We can then infer that p’ is not good for estimating pressure. In SIMPLER method, more effective pressure equation is used. This results in about 30% larger number of calculations, but fast convergence reduces the computing time by 30-50%. (H.K.VersteegW.Malalasekera, 1995)

For another improvement, the coding style can be changed to make the computing more efficient. For instance, I have used much storage for storing the coefficients and have used iterative method. For saving storage, other coding methods such as using classes or SciPy, which has functions that solve the equations efficiently, could save computing time. However, this method allows easier modification for other problems, where the stored information of the coefficients might be needed.

5. Appendix: Grid Convergence Study

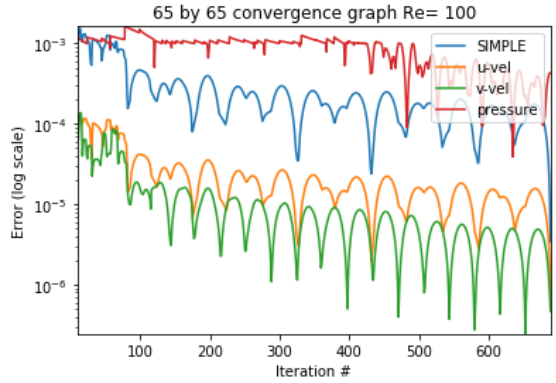
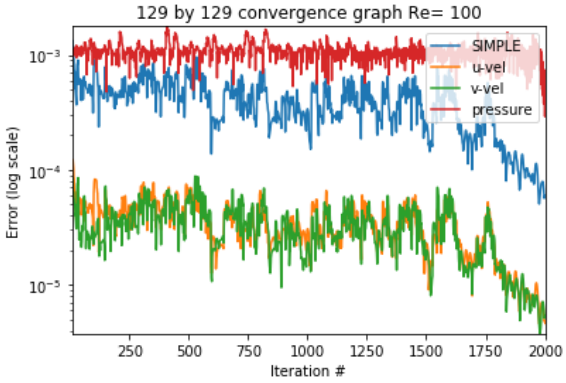
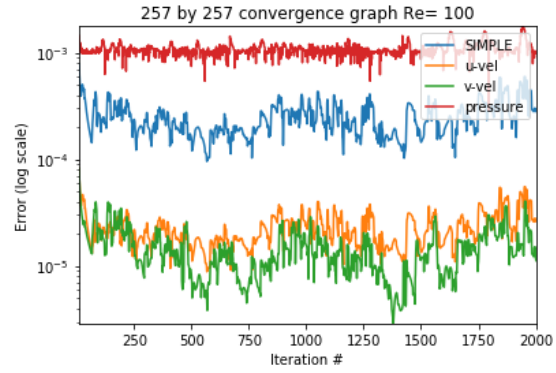
Size	Iterations	Convergence Graph
65x65	688 -U Iteration#: 2 688 -u Error: 1.76580071073e-06 688 -V Iteration#: 1 688 -V Iteration#: 2 688 -v Error: 7.65569805535e-07 688 -P Iteration # 1 688 -P Iteration # 2 688 -p Error: 0.000432772315546 688 -SIMPLE Error is: 1.1396420992e-05 Main SIMPLE Iteration#: 689 689 -U Iteration#: 1 689 -U Iteration#: 2 689 -u Error: 7.08035358938e-07 689 -V Iteration#: 1 689 -V Iteration#: 2 689 -v Error: 4.68058062041e-07 689 -P Iteration # 1 689 -P Iteration # 2 689 -p Error: 0.000432455342266 689 -SIMPLE Error is: 3.4072106331e-06	 Execution time = 350.40688133239746
129x129	2000 -u Error: 4.75862474578e-06 2000 -V Iteration#: 1 2000 -V Iteration#: 2 2000 -v Error: 5.30088450984e-06 2000 -P Iteration # 1 2000 -P Iteration # 2 2000 -p Error: 0.000452379598904 2000 -SIMPLE Error is: 5.91109110337e-05 Main SIMPLE Iteration#: 2001 2001 -U Iteration#: 1 2001 -U Iteration#: 2 2001 -u Error: 4.6315663317e-06 2001 -V Iteration#: 1 2001 -V Iteration#: 2 2001 -v Error: 5.37723687516e-06 2001 -P Iteration # 1 2001 -P Iteration # 2 2001 -p Error: 0.000440557262395 2001 -SIMPLE Error is: 5.96033389941e-05	 Execution time = 3802.998050928116
257x257	Main SIMPLE Iteration#: 2000 2000 -U Iteration#: 1 2000 -U Iteration#: 2 2000 -u Error: 2.63173380469e-05 2000 -V Iteration#: 1 2000 -V Iteration#: 2 2000 -v Error: 1.16503548857e-05 2000 -P Iteration # 1 2000 -P Iteration # 2 2000 -p Error: 0.00098358336767 2000 -SIMPLE Error is: 0.000282673036343 Main SIMPLE Iteration#: 2001 2001 -U Iteration#: 1 2001 -U Iteration#: 2 2001 -u Error: 2.61804290135e-05 2001 -V Iteration#: 1 2001 -V Iteration#: 2 2001 -v Error: 1.12567257411e-05 2001 -P Iteration # 1 2001 -P Iteration # 2 2001 -p Error: 0.000982812090275 2001 -SIMPLE Error is: 0.000281671516043	 Execution time = 17911.19946050644

Fig. 5.1

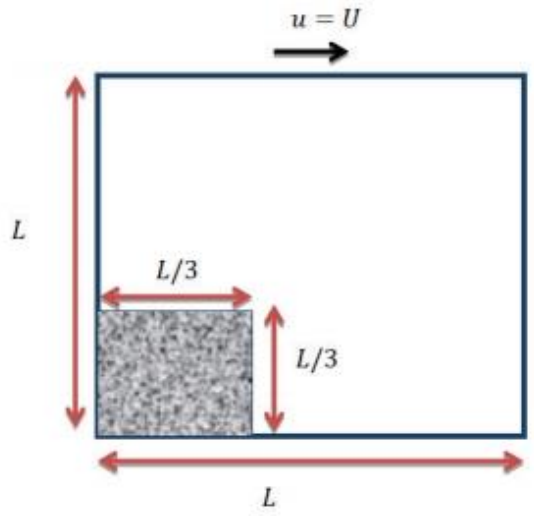
As seen in the convergence graphs above and motioned before, the error for pressure is the highest as expected. It can be seen that 65x65 mesh has converged completely and that 129x129 has begun converging but have not converged yet by iteration 2000. For 257x257, it has not started converging yet, hence the errors in the velocity graphs as shown in the previous sections. The solution is sufficiently resolved, and the results do not change significantly with mesh refinement. However, the computing time differs significantly, and more iterations and time is needed for higher number of mesh.

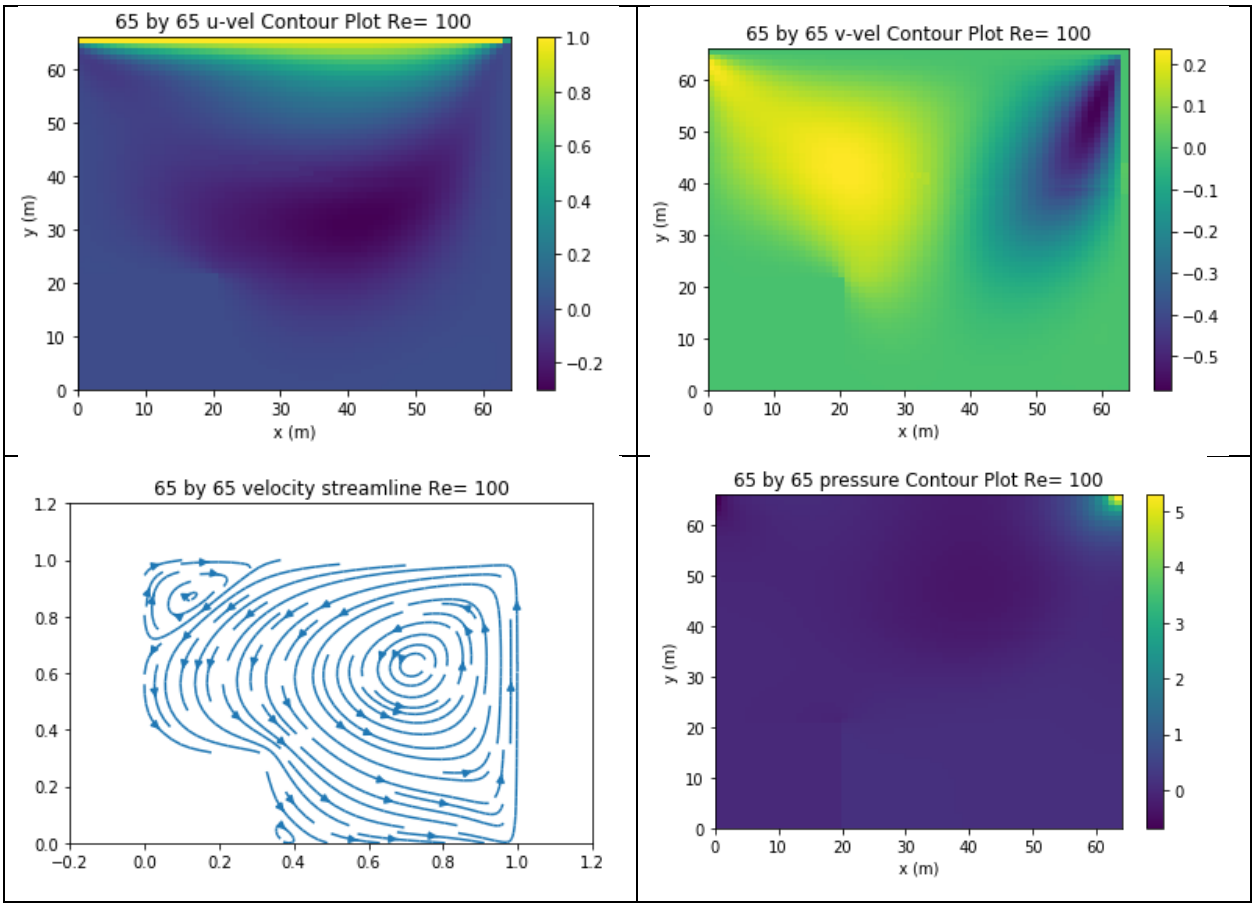
6. Bonus

6.1 Lid Driven Cavity with Step

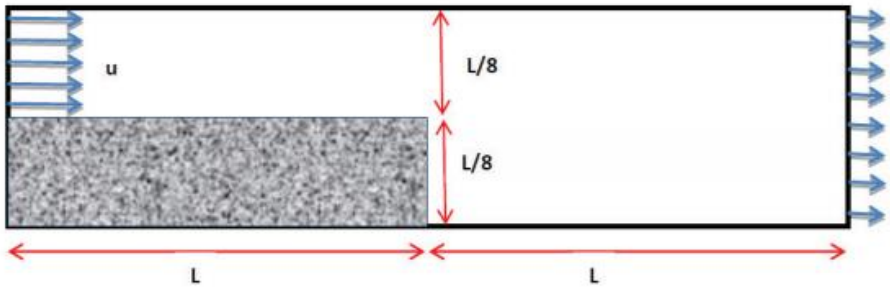
The lid driven cavity flow with a step as shown in Fig.6.1.1 is solved. This done by applying the boundary condition of the wall on the block.

The results of the run are shown below. As seen in the stream line graph, there are two secondary vortex flows on the top left corner and the bottom right of the step.





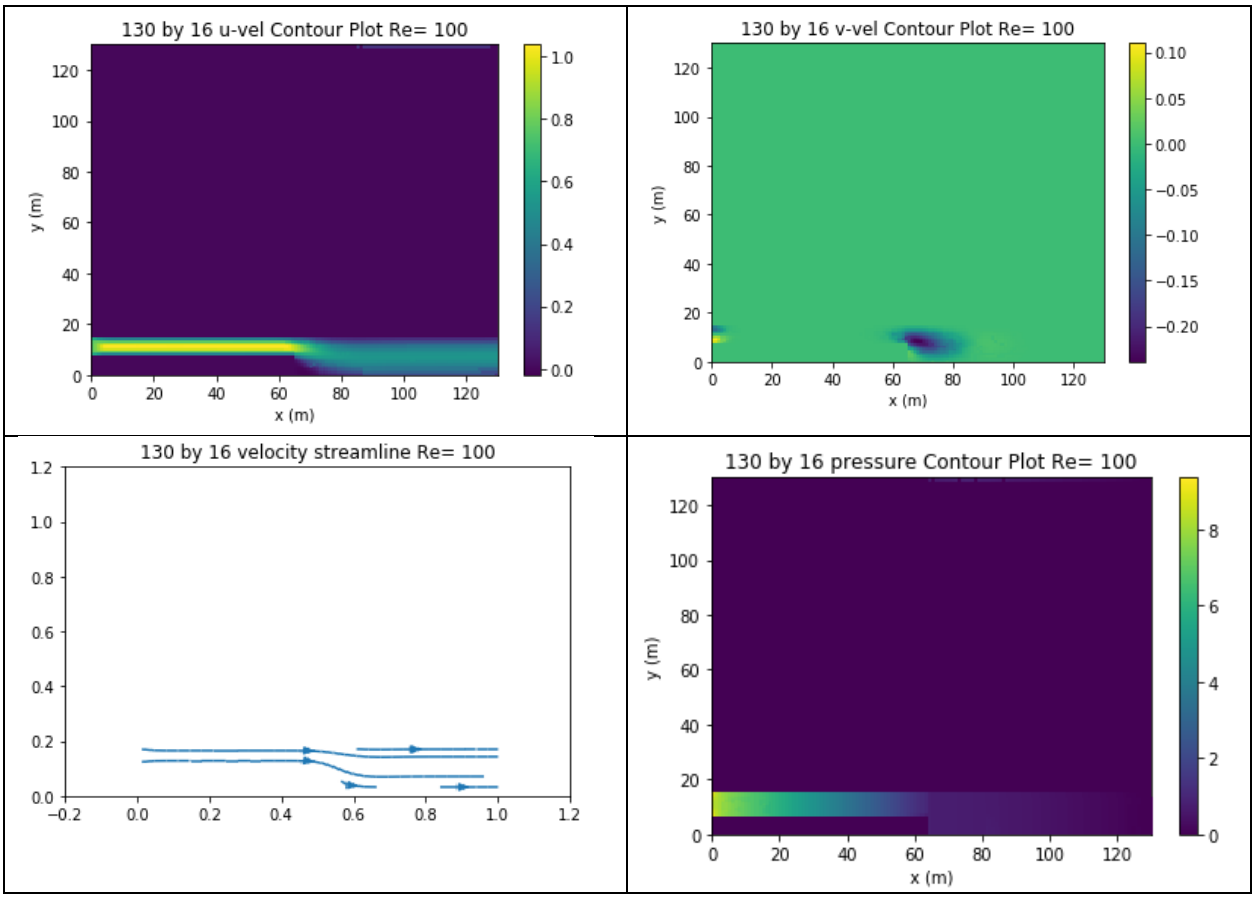
6.2 Back-Step Flow



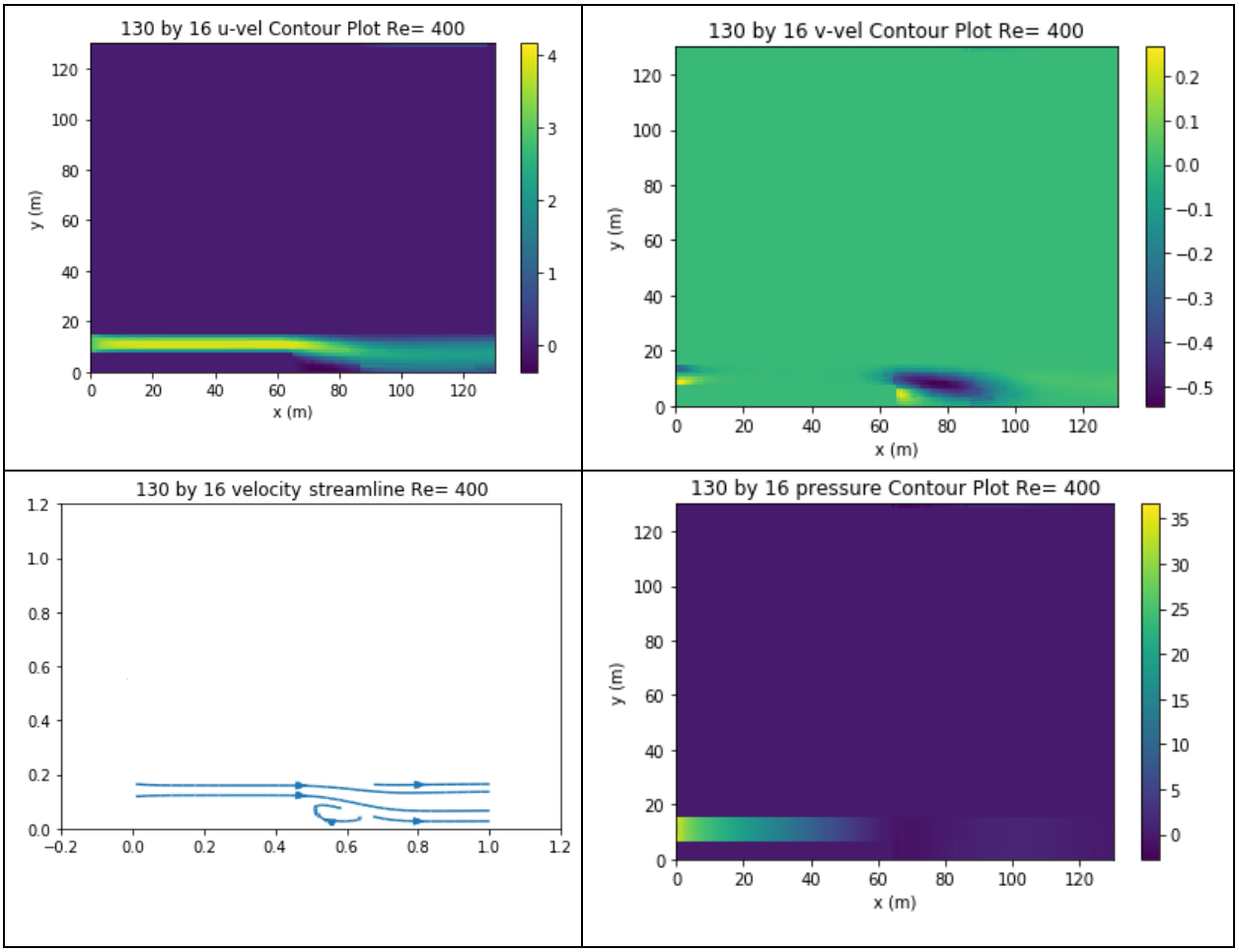
The back step flow problem shown in the figure above is solved.

For boundary conditions, the inlet velocity u=1m/s, with the same boundary conditions for the others.

For Re=100



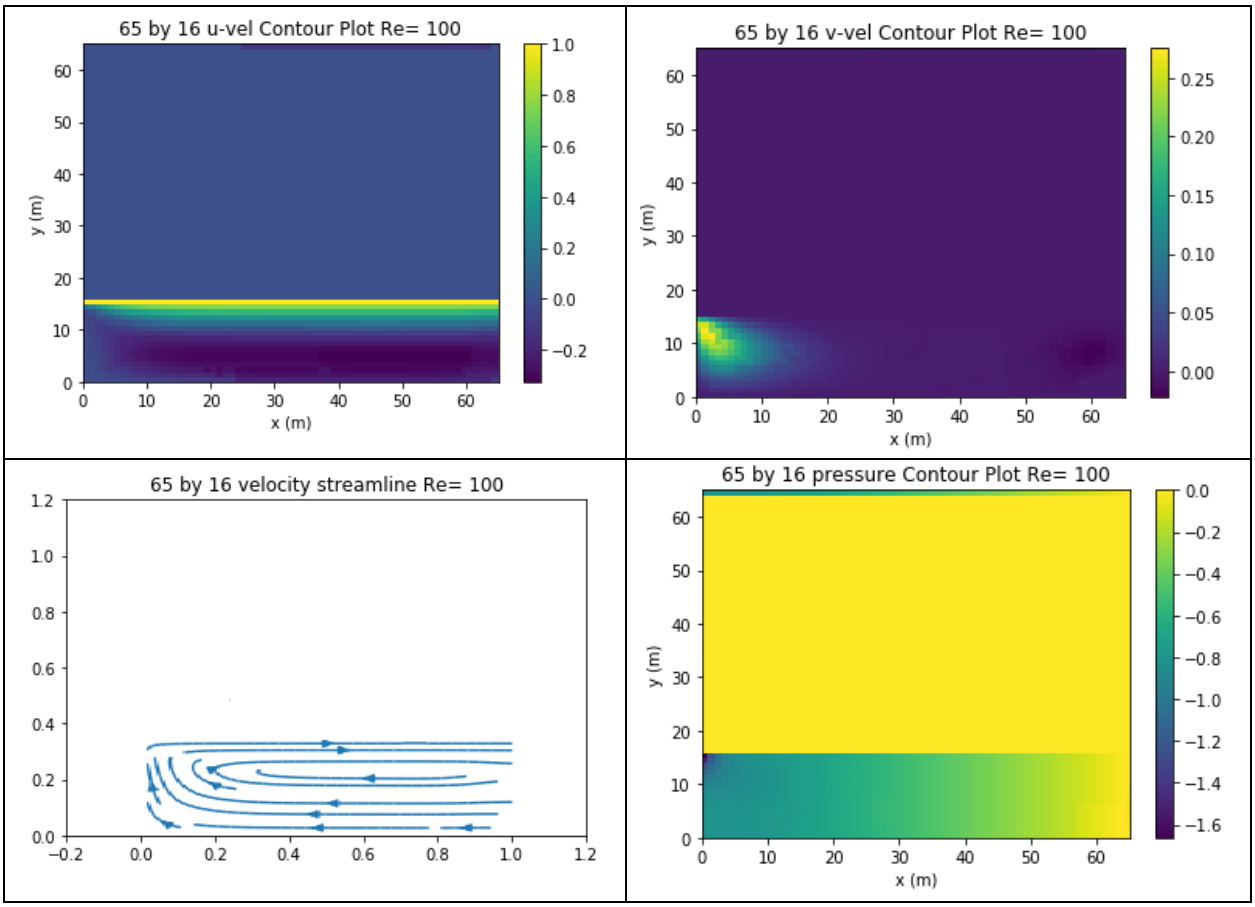
For Re=400



As seen in the above figures, with higher Reynolds number the secondary vortex flow on the bottom right of the step becomes larger.

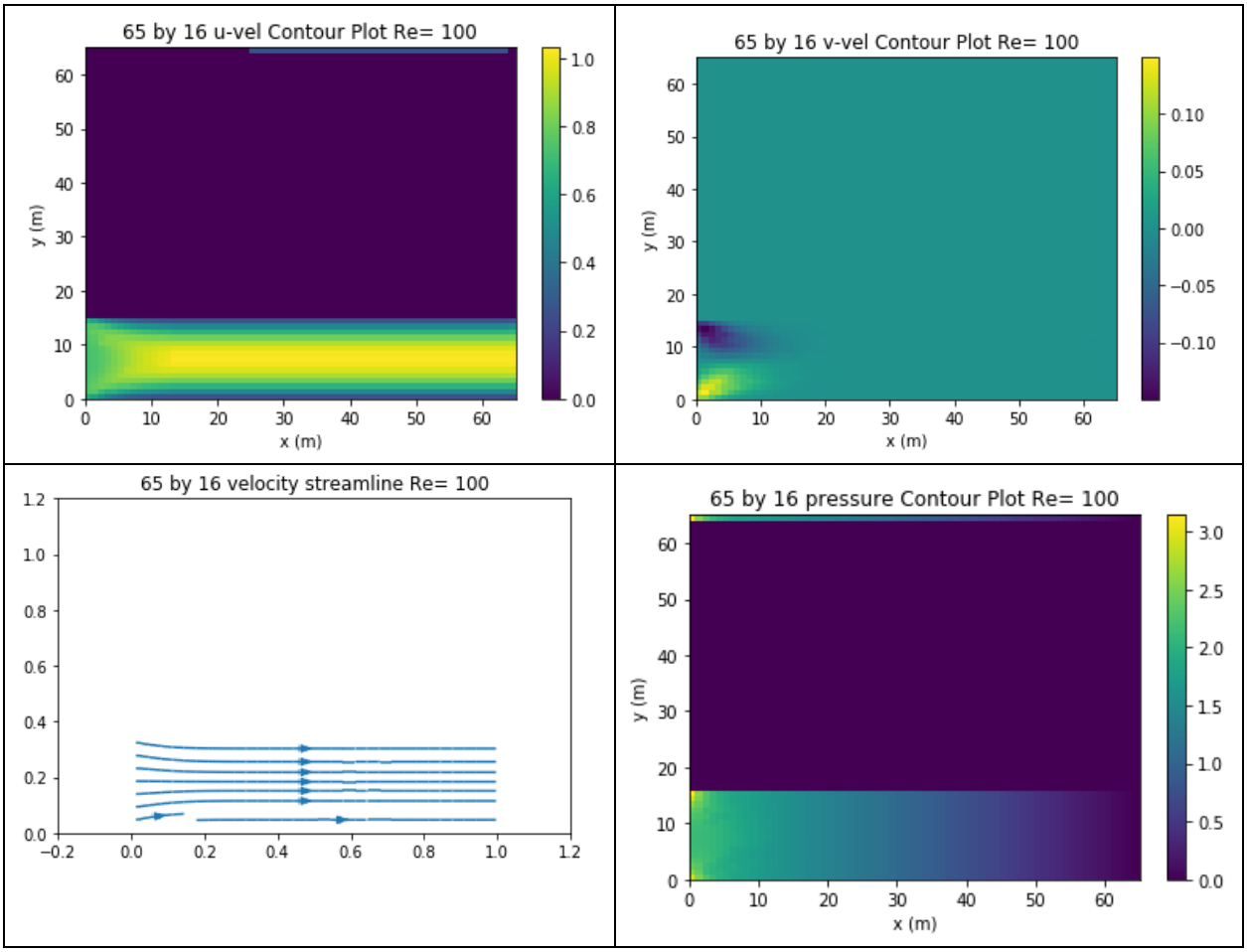
6.3 Couette Flow

Couette Flow is solved, where the top lid is moving at 1m/s with no boundary condition on the right and left.



6.4 Fully Developed Channel Flow

Fully developed Channel flow is the back-step flow without the step.



7. References

CFDOnline. (n.d.). *Rhie-Chow interpolation*. Retrieved from CFD Online Wiki: https://www.cfd-online.com/Wiki/Rhie-Chow_interpolation

H.K.Versteeg, W. (1995). *An Introduction to Computational Fluid Dynamics. The Finite Volume Method.* Longman Scientific & Technical.

U. Ghia, K. G. (1982). *High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method.* Cincinnati, Ohio: University of Cincinnati. Retrieved from https://www.sciencedirect.com/science/article/pii/0021999182900584