ME 6434- Final Project

Solution of a Lid Driven Cavity at $RE = 100$ and $RE = 1000$ Using a Staggered Grid and a Predictor Corrector Method

Kevin Hoopes

May 9, 2012

Abstract

The lid driven cavity problem on a square domain is solved for $RE = 100$ and $RE = 1000$ cases using a staggered grid and a predictor corrector solution method. Time integration is performed using the 2^{nd} order Adams-Bashforth integration scheme and the solution is advanced in time to a steady state for each case. The steady state solution at $RE = 100$ and $RE = 1000$ are found to be in good agreement with the results obtained by Ghia, Ghia, and Shin [1]. For the $RE = 1000$ case, the largest permissible time step of $\Delta t = 0.00581$ was found and used for the computations. This result is compared to the maximum permissible time step for the $RE = 100$ case.

1 Problem Definition and Setup

The lid driven cavity is a very common test case for validating new CFD codes and techniques as well as an introduction to CFD programming. Figure 1 shows the geometry and boundary conditions associated with a lid driven cavity type problem. As can be seen in the figure, the top of

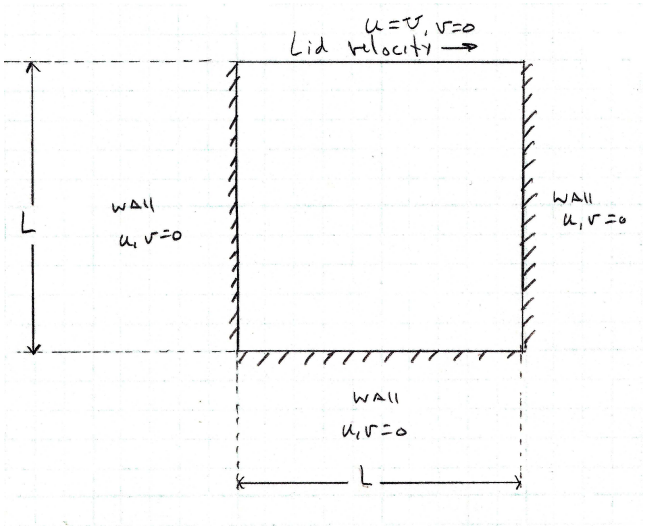


Figure 1: Geometry used in the current study. As the figure indicates, the lid of a square box, LxL has a positive velocity U, while all the other sides of the box are stationary walls.

the cavity is a moving wall with velocity U. All of the other walls are stationary. The incompressible Navier Stokes equations were used in order to model the behavior of the fluid flow. In order to solve for the resulting flow at different lid velocities, the problem is first nondimensionalized in section 2. A general overview of the solution is presented in section 3. The discritization of the Navier Stokes equations onto a uniform staggered grid is presented in section 4. Sections 6 and 7 show results for $RE = 100$ and $RE = 1000$ cases.

2 Nondimensionalization of Governing Equations

The x-momentum, y-momentum, and continuity equations are given in equations 1, 2, and 3 respectively.

$$\frac{\partial}{\partial t}(\rho u) + \frac{\partial}{\partial x}(\rho uu) + \frac{\partial}{\partial y}(\rho uv) = -\frac{\partial P}{\partial x} + \frac{\partial}{\partial x}(\mu \frac{\partial u}{\partial x}) + \frac{\partial}{\partial y}(\mu \frac{\partial u}{\partial y}) \tag{1}$$

$$\frac{\partial}{\partial t}(\rho v) + \frac{\partial}{\partial x}(\rho uv) + \frac{\partial}{\partial y}(\rho vv) = -\frac{\partial P}{\partial y} + \frac{\partial}{\partial x}(\mu\frac{\partial v}{\partial x}) + \frac{\partial}{\partial y}(\mu\frac{\partial v}{\partial y}) \tag{2}$$

$$\frac{\partial}{\partial x}(\rho u) + \frac{\partial}{\partial y}(\rho v) = 0 \tag{3}$$

In order to make these equations easier to discritize and handle through the present solution technique, they are nondimensionalized.

2.1 Nondimensional Factors

Table 1 contains the chosen dimensions used in the nondimensionalized procedure. The lid velocity was chosen as the characteristic velocity, along with the the side length of the cavity for the characteristic length. It should be noted that although a reference pressure is used in the nondimensionalization procedure, throughout the solution it is assumed that it has a value of 0.

Table 1: Dimensional factors used to nondimensionalize the incompressible Navier Stokes equations

Characteristic Dimension	**Description**
U	Lid velocity
L	Lid side length
P_{ref}	Reference pressure

Table 2 shows the nondimensionalized versions of all the variables in equations 1 through 3.

Table 2: Nondimensional groups along with their corresponding dimensional variables

Variable	**Nondimensional Relationship**
u	$u^* = u/U$
v	$v^* = u/U$
x	$x^* = x/L$
y	$y^* = y/L$
t	$t^* = tU/L$
P	$P^* = \frac{P-P_{ref}}{\rho U^2}$

2.2 Nondimensionalization of the Momentum Equation

The nondimensional relationships defined in table 2 were substituted into equation 1 which results in equation 4.

$$\rho\frac{\partial(u^*U)}{\partial(t^*L/U)} + \rho\frac{\partial(u^*Uu^*U)}{\partial x^*L} + \rho\frac{\partial(u^*Uv^*U)}{\partial y^*L} = -\frac{\partial(P^*\rho U^2 + P_{ref})}{\partial x^*L} + \frac{\partial}{\partial x^*L}(\mu\frac{\partial u^*U}{\partial x^*L}) + \frac{\partial}{\partial y^*L}(\mu\frac{\partial u^*U}{\partial y^*L}) \tag{4}$$

Next, the entire equation is multiplied by $L/(\rho U^2)$ and the resulting $\frac{\mu}{\rho UL}$ term is collapsed into the Reynolds number. The resulting equation is shown in equation 5.

$$\frac{\partial u^*}{\partial t^*} + \frac{\partial u^*u^*}{\partial x^*} + \frac{\partial u^*v^*}{\partial y^*} = -\frac{\partial P^*}{\partial x^*} + \frac{1}{RE}\left[\frac{\partial^2 u^*}{\partial(x^*)^2} + \frac{\partial^2 u^*}{\partial(y^*)^2}\right] \tag{5}$$

Finally, for convenience, the * terms are dropped and equation 6 is the final nondimensional form of the x-momentum equation.

$$\frac{\partial u}{\partial t} + \frac{\partial uu}{\partial x} + \frac{\partial uv}{\partial y} = -\frac{\partial P}{\partial x} + \frac{1}{RE}\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right] \tag{6}$$

A similar procedure is applied to the y-momentum equation resulting in equation 7.

$$\frac{\partial v}{\partial t} + \frac{\partial uv}{\partial x} + \frac{\partial vv}{\partial y} = -\frac{\partial P}{\partial y} + \frac{1}{RE}\left[\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right] \tag{7}$$

2.3 Nondimensionalization of the Continuity Equation

The nondimensional relationships defined in table 2 were substituted into equation 3 which resulted in equation 8.

$$\frac{\partial u^* U}{\partial x^* L} + \frac{\partial v^* U}{\partial y^* L} = 0 \tag{8}$$

Next, multiply the whole equation by L/U which results in equation 9.

$$\frac{\partial u^*}{\partial x^*} + \frac{\partial v^*}{\partial y^*} = 0 \tag{9}$$

Finally the * designation is dropped for convenience, resulting in the nondimensional form of the continuity equation, equation 10.

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{10}$$

3 Overview of solution technique

Now that all the equations have been nondimensionalized, we can turn our attention to actually solving this system of coupled partial differential equations. The general framework that was used to solve the equations is enumerated in the following list:

1. Predict x and y velocities at the next time step, designated $\tilde{u}$ and $\tilde{v}$, using a form of the x and y momentum equations and the 2^{nd} order Adams-Bashforth time integration technique.
2. Form a pressure correction, P', using $\tilde{u}$ and $\tilde{v}$ that will ensure that the velocity terms at the next time step will satisfy the continuity equation.
3. Solve for P' using an iterative solver.
4. Correct velocity and Pressure using $P^{'}$.
5. Return to step 2.

This technique can be characterized as a two step, explicit, pressure correction technique. First, the velocity approximated at the next time step, and then this approximation is corrected to reveal the actual velocity at the next time step. This effectively decouples the continuity and the momentum equations making the solution much simpler and more rapid.

In order to advance the solution in time, a 2^{nd} order Adams-Bashforth time integration scheme is used. In general, the scheme integrates a differential equation like the one found in 11.

$$\frac{d\phi}{dt} = H(\phi) \tag{11}$$

This equation can be integrated to equation 12.

$$\phi^{t_n+1} = \phi^{t_n} + \int_{t_n}^{t_n+1} H(\phi)dt \tag{12}$$

The 2^{nd} order Adams-Bashforth time integration scheme approximates the integral in 12 using the current and previous function values to form a linear approximation to the function. The resulting relationship is shown in equation 13. Note that in this equation, the superscripts denote time steps.

$$\phi^{t_n+1} = \phi^{t_n} + \Delta t \left[\frac{3}{2}H^n - \frac{1}{2}H^{n-1}\right] \tag{13}$$

One major advantage of this method is that it is explicit, meaning that the values at the next time step do not depend on each other, but only on the values at the current and previous time steps. This makes the actual computation much faster as there are no linear systems to solve; the velocity at a point is simply an algebraic combination of surrounding velocities and pressures. Also, even though this method is explicit, it is second order accurate in time.

In order to apply this solution method to the lid driven cavity problem, the momentum equations must be rearranged to be in the same form as equation 11. This can be seen for the X-momentum equation in equation 14.

$$\frac{\partial u}{\partial t} = -\frac{\partial P}{\partial x} + \frac{1}{RE}\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right] - \frac{\partial uu}{\partial x} - \frac{\partial uv}{\partial y} \tag{14}$$

The pressure terms are ignored presently, resulting in equation 15. The motivated reader will notice that this matches the form needed for the Adams-Bashforth method as defined in 11.

$$\frac{\partial u}{\partial t} = H_x(u,v) = \frac{1}{RE}\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right] - \frac{\partial uu}{\partial x} - \frac{\partial uv}{\partial y} \tag{15}$$

Note that in its present form, this H_x term is still a continuous equation. In the section 4.1 it will be discritized on a finite volume grid.

The pressure term is now added back to the formulation resulting in equation 16.

$$\tilde{u} = u^n + \Delta t \left[\frac{3}{2}H_x^n - \frac{1}{2}H_x^{n-1}\right] - \Delta t\frac{\partial P^n}{\partial x} \tag{16}$$

True to the general solution steps outlined above, the resulting prediction for the velocity at the next time step is given the tilde designation. This is done as we know that the predicted velocities will probably not satisfy continuity.

Ideally we would not need to use an intermediate velocity and could simply move from u^n directly to u^{n+1} as shown in equation 17. Note that this equation would only work if we were sure that our system would satisfy continuity.

$$u^{n+1} = u^n + \Delta t \left[\frac{3}{2} H_x^n - \frac{1}{2} H_x^{n-1} \right] - \Delta t \frac{\partial P^n}{\partial x} \tag{17}$$

By subtracting equation 17 from 16, we arrive at equation 18, which will be termed the X corrector equation. Basically it gives an expression for u^{n+1} in terms of $\tilde{u}$ and a pressure correction term.

$$u^{n+1} = \tilde{u} - \Delta t \frac{\partial (P^{n+1} - P^n)}{\partial x} \tag{18}$$

To make the notation easier, the relationship for this pressure correction found in equation 19 will be used.

$$P^{'} = P^{n+1} - P^n \tag{19}$$

Substituting equation 19 into equation 18 we arrive at our final correction equation, equation 20. The value of $P^{'}$ that is used in the corrector equation comes from the discritized continuity equation. The details of how to obtain this value are contained in section 4.5.

$$u^{n+1} = \tilde{u} - \Delta t \frac{\partial P^{'}}{\partial x} \tag{20}$$

Similarly the v predictor and corrector steps can be written as equations 21 and 22.

$$\tilde{v} = v^n + \Delta t \left[\frac{3}{2} H_y^n - \frac{1}{2} H_y^{n-1} \right] - \Delta t \frac{\partial P^n}{\partial y} \tag{21}$$

$$v^{n+1} = \tilde{v} - \Delta t \frac{\partial P^{'}}{\partial y} \tag{22}$$

In equation 21, H_y is composed of the convection and diffusion terms derived from the discritized y-momentum equation and $P^{'}$ is the same pressure correction seen in equation 20.

In summary, the solution procedure can be outlined as follows. We desire to advance our solution in time one time step. We first predict what the velocity would be at that next time step assuming that our equations already satisfy continuity. We then formulate a correction for these intermediate velocities using the continuity equation. When this correction is applied to both pressure and velocity, we are now at the next time step. This decouples the momentum and continuity equations, but it still makes sure that they are both satisfied for each time step.

4 Discritization of governing equations

As we saw in section 3, in order to advance our solution in time we need expressions for the discritzed momentum and continuity equations. Figure 2 displays the solution domain discritized into a staggered grid. Each circular node in the figure represents a pressure node, a horizontal arrow represents a u-velocity node, and a vertical arrow represents a v-velocity node. The small figure to the right of the grid explains the grid's staggered nature. The velocities associated with each pressure node are offset from the pressure node itself even though all of these three components would have the same i and j indicies.

This type of staggered arrangement is done in order to avoid over interpolation. As will be seen in the finite volume discritization, often values are needed at the cell faces. This staggered grid

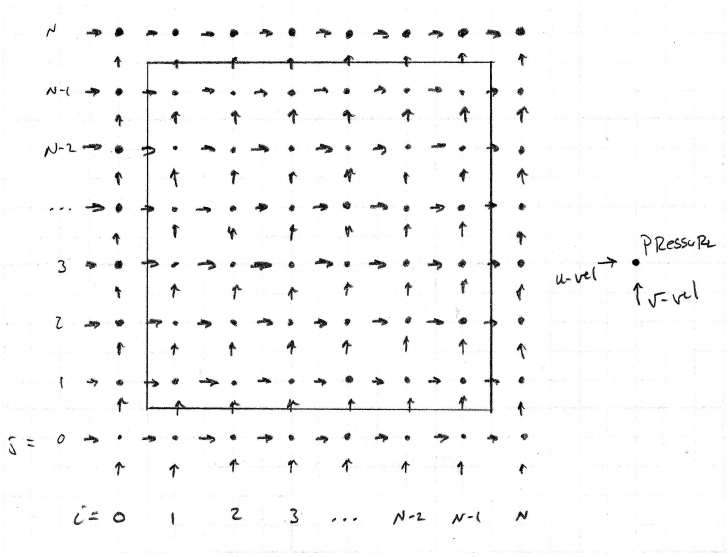


Figure 2: The staggered grid that is applied to the lid driven cavity problem. The solid lines indicate the boundaries of the solution domain. The figure at the right represents a typical P, u, v node combination.

arrangement provides a simple way to have these values easily at hand without resorting to interpolation of cell face values from far off nodes.

The u-velocity is predicted and corrected at grid points $i = 2 \rightarrow N-1$ and $j = 1 \rightarrow N-1$. Similarly, the v-velocity is predicted and corrected at grid points $i = 1 \rightarrow N-1$ and $j = 2 \rightarrow N-1$. The pressure correction is evaluated at every internal grid point, $i = 1 \rightarrow N-1$ and $j = 1 \rightarrow N-1$.

4.1 X-Momentum

In order to discritize the X-momentum equation, a control volume is drawn around a typical u-velocity node. This control volume can be seen as the dotted line in figure 3.

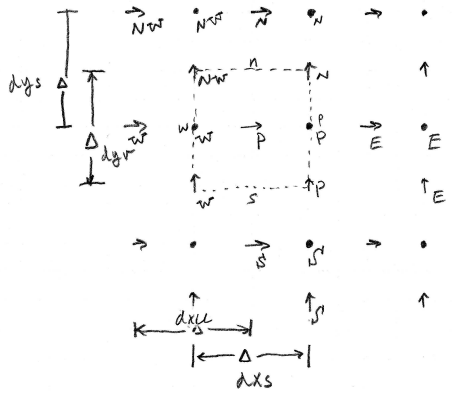


Figure 3: The dotted line shows a typical control volume used in the discritization of the X-momentum equation. The capital letters P, N, E, S, W indicate nodal values while the lower case letters n, e, s, w represent the cell faces.

The continuous form of the X-momentum equation can be written as equation 23.

$$\frac{\partial u}{\partial t} + \frac{\partial uu}{\partial x} + \frac{\partial uv}{\partial y} = -\frac{\partial P}{\partial x} + \frac{1}{RE}\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right] \tag{23}$$

Ignoring the pressure terms as stated previously and integrating over the finite volume, we arrive at equation 24.

$$\int_w^e \int_s^n \frac{\partial u}{\partial t} dxdy + \int_w^e \int_s^n \frac{\partial uu}{\partial x} dxdy + \int_w^e \int_s^n \frac{\partial uv}{\partial y} dxdy = \int_w^e \int_s^n \frac{1}{RE}\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right] dxdy \tag{24}$$

Knowing that $\Delta = dyv = dxs$ the integrals can be evaluated as equation 25.

$$\frac{\partial u}{\partial t}\Delta^2 + (uu|_e - uu|_w)\Delta + (uv|_n - uv|_s)\Delta = \frac{1}{RE}\left[\left(\frac{\partial u}{\partial x}\Big|_e - \frac{\partial u}{\partial x}\Big|_w\right)\Delta + \left(\frac{\partial u}{\partial y}\Big|_n - \frac{\partial u}{\partial y}\Big|_s\right)\Delta\right] \tag{25}$$

Finally, after some algebraic manipulation we arrive at equation 26, which is the discritized form of the momentum equation that will be needed in the u velocity predictor equation, equation 16.

$$H_x = \frac{1}{\Delta RE}\left[\left(\frac{\partial u}{\partial x}\Big|_e - \frac{\partial u}{\partial x}\Big|_w\right) + \left(\frac{\partial u}{\partial y}\Big|_n - \frac{\partial u}{\partial y}\Big|_s\right)\right] - \frac{(uu|_e - uu|_w)}{\Delta} - \frac{(uv|_n - uv|_s)}{\Delta} \tag{26}$$

Even though this equation is in a discritized form, it is still written in terms of velocities and derivatives of velocities at the faces of the x-momentum control volume. Table 3 shows how these terms can be approximated using the velocities of the surrounding nodes.

Table 3: Coefficients used in the calculation of H_x, equation 26

$u\|_n = 1/2(u_P + u_N)$	$u\|_e = 1/2(u_P + u_E)$	$u\|_s = 1/2(u_P + u_S)$	$u\|_w = 1/2(u_P + u_W)$
	$v\|_n = 1/2(v_{NW} + v_N)$	$v\|_s = 1/2(v_W + v_P)$	
$\frac{\partial u}{\partial x}\|_e = \frac{u_E - u_P}{dxu}$	$\frac{\partial u}{\partial x}\|_w = \frac{u_P - u_W}{dxu}$	$\frac{\partial u}{\partial y}\|_n = \frac{u_N - u_P}{dys}$	$\frac{\partial u}{\partial y}\|_s = \frac{u_P - u_S}{dys}$

As an example, u is needed at the north face of the x-momentum control volume. This is denoted $u|_n$ in equation 26. In order to express this term as a function of known u-velocities we consult the diagram of the x-momentum control volume as found in figure 3. $u|_n$ can be approximated as the simple average of u_N and u_P. The derivatives are evaluated similarly, for example $\frac{\partial u}{\partial x}|_w$ can be evaluated as $\frac{u_P - u_W}{dxu}$. This represents a central difference approximation to the derivative at the west face of the control volume.

4.2 X-Momentum Boundaries

As stated previously, the u-velocity is predicted and corrected at grid points $i = 2 \rightarrow N-1$ and $j = 1 \rightarrow N-1$ as shown in figure 2. At $i = 2$ and $i = N-1$ the u-velocity control volume will not interact with the bounding walls of the domain so no change in the formulation of the H_x term is needed.

On the other hand, when evaluating H_x at $j = 1$ and $j = N-1$ The walls of the domain will form the bottom and top of the control volume. This does not effect the formulation of the H_x equation,

equation 26, but it does affect how the velocities and velocity derivatives are computed at the cell faces. This essentially modifies table 3 for this bounding case.

The key to the required modification is to assume that the values outside of the domain lie on the walls of the domain. This will ensure that our boundary conditions are enforced as, for example, there are no u velocity nodes on the top wall of the domain. Without some sort of treatment, it would be difficult for this wall velocity to affect the flow field.

For example, the u-velocity will be predicted at $i = 3$, and $j = N-1$. This will result in the top wall of the control volume lying along the top wall of the domain. This is easiest to see by examining figure 3 and imagining that the top of the control volume is also the top of the domain. We will assume that u_N actually lies on the the n face of the control volume, so, we will write $u|n = u_N$ instead of the average of u_P and u_N as usual. The derivative term, $\frac{\partial u}{\partial y}|_n$ will now be formulated as usual, but we will make sure to specify the distance between u_N and u_P as $0.5dys$ instead of dys to represent that this velocity is now considered to be at the face of the control volume. The resulting expression can be written as $\frac{\partial u}{\partial y}|_n = \frac{u_N - u_P}{0.5dys}$.

Similarly along the bottom wall, when $j = 1$, $u|_s = u_S$ and $\frac{\partial u}{\partial y}|_s = \frac{u_P - u_S}{0.5dys}$.

4.3 Y-Momentum

The discritization of the y-momentum equation is very similar to that of the x-momentum equation. Figure 4 shows a typical control volume for the computation of H_y.

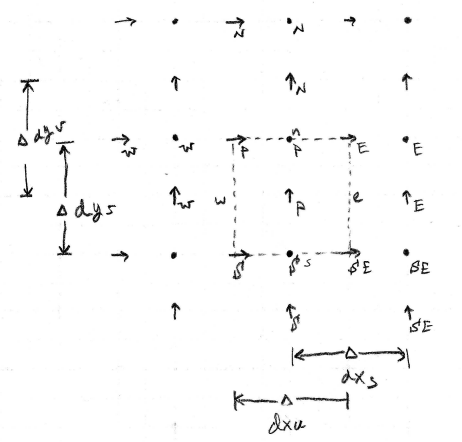


Figure 4: The dotted line shows a typical control volume used in the discritization of the y-momentum equation. The capital letters P, N, E, S, W indicate nodal values while the lower case letters n, e, s, w represent cell faces.

Notice that now the control volume is centered at a v-velocity arrow, though the naming conventions used in the x-momentum control volume are the same.

Following a similar procedure as in section 4.1, an expression for H_y is obtained and is found in equation 27.

$$H_y = \frac{1}{\Delta RE}\left[\left(\frac{\partial v}{\partial x}\Big|_e - \frac{\partial v}{\partial x}\Big|_w\right) + \left(\frac{\partial v}{\partial y}\Big|_n - \frac{\partial v}{\partial y}\Big|_s\right)\right] - \frac{(uv|_e - uv|_w)}{\Delta} - \frac{(vv|_n - vv|_s)}{\Delta} \tag{27}$$

Table 4 lists the approximations used when evaluating equation 27. These approximations are

formulated using the same logic that table 3 was formulated.

Table 4: Coefficients used in the calculation of H_y, equation 27

$v\|_n = 1/2(v_P + v_N)$	$v\|_e = 1/2(v_P + v_E)$	$v\|_v = 1/2(v_P + v_S)$	$v\|_w = 1/2(v_P + v_W)$
	$u\|_e = 1/2(v_{SE} + u_E)$	$u\|_w = 1/2(u_S + u_P)$	
$\frac{\partial v}{\partial x}\|_e = \frac{v_E - v_P}{dxu}$	$\frac{\partial v}{\partial x}\|_w = \frac{v_P - v_W}{dxu}$	$\frac{\partial v}{\partial y}\|_n = \frac{v_N - v_P}{dys}$	$\frac{\partial v}{\partial y}\|_s = \frac{v_P - v_S}{dys}$

4.4 Y-Momentum Boundaries

Similar to the x-momentum equation, special treatment is needed at some of the extrema of the H_y calculation. When $j = 2$ and $j = N - 1$ no special treatment is needed as the boundaries of the resulting control volume will not intersect the boundaries of the domain. On the other hand, when $i = 1$ and $i = N - 1$ the boundaries of the control volume will intersect the walls of the system and modifications to the approximations found in table 4 must be made.

The same general rule is used as when modifying the x-momentum equations. The nodes beyond the walls are assumed to lie on the walls themselves when formulating the approximations.

When $i = 1$ then $u|_w = u_W$ and $\frac{\partial v}{\partial x}|_w = \frac{v_P - v_W}{0.5dxu}$ Similarly when $i = N - 1$ then $u|_e = u_E$ and $\frac{\partial v}{\partial x}|_e = \frac{v_E - v_P}{0.5dxu}$

4.5 Continuity and Pressure corrections

The idea behind the pressure correction step is to form a correction such that it can be used to modify the $\tilde{u}$ and $\tilde{v}$ terms such that they will satisfy continuity.

With this in mind, first substitute u^{n+1} from equation 20 and v^{n+1} from equation 22 into the general form of the continuity equation, equation 10, as u and include v to get 28.

$$\frac{\partial}{\partial x}\left[\tilde{u} - \Delta t \frac{\partial P'}{\partial x}\right] + \frac{\partial}{\partial y}\left[\tilde{v} - \Delta t \frac{\partial P'}{\partial y}\right] = 0 \tag{28}$$

After some algebraic manipulation this can be written as equation 29, which is the general, continuous form of our pressure correction equation.

$$\frac{\partial^2 P'}{\partial x^2} + \frac{\partial^2 P'}{\partial y^2} = \frac{1}{\Delta t}\left[\frac{\partial \tilde{u}}{\partial x} + \frac{\partial \tilde{u}}{\partial y}\right] \tag{29}$$

We then discritize our domain as shown in figure 5. This is similar to the approach used in the x and y momentum equations except now the pressure node is at the center of the control volume.

Then integrate over the control volume as shown in equation 30.

$$\int_w^e \int_s^n \frac{\partial^2 P'}{\partial x^2} dxdy + \int_w^e \int_s^n \frac{\partial^2 P'}{\partial y^2} dxdy = \int_w^e \int_s^n \frac{1}{\Delta t}\left[\frac{\partial \tilde{u}}{\partial x} + \frac{\partial \tilde{u}}{\partial y}\right] dxdy \tag{30}$$

Evaluate these integrals at the cell faces to obtain equation 31.

$$\left[\frac{\partial P'}{\partial x}\Big|_e - \frac{\partial P'}{\partial x}\Big|_w\right] dys + \left[\frac{\partial P'}{\partial y}\Big|_n - \frac{\partial P'}{\partial y}\Big|_s\right] dxs = \frac{1}{\Delta t}\Big[(\tilde{u}|_e - \tilde{u}|_w)dys + (\tilde{v}|_n - \tilde{v}|_s)dxs\Big] \tag{31}$$

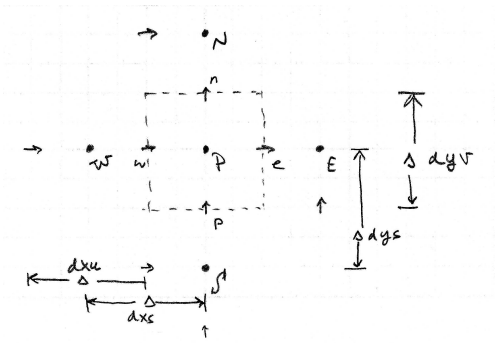


Figure 5: The dotted line shows a typical control volume used in the discritization of the pressure correction equation. The capital letters P, N, E, S, W indicate nodal values while the lower case letters n, e, s, w represent cell faces.

Table 5 shows the approximations used for the derivatives of pressure found in equation 31. Since the velocities used in this equation are simply the velocities at the cell faces, which our staggered grid already has, no approximations are needed for these velocities.

Table 5: Coefficients used in the calculation of the discrete pressure equation, equation 31

$$\left.\frac{\partial P'}{\partial x}\right|_e = \frac{P'_E - P'_P}{dxs} \qquad \left.\frac{\partial P'}{\partial x}\right|_w = \frac{P'_P - P'_W}{dxs} \qquad \left.\frac{\partial P'}{\partial y}\right|_n = \frac{P'_N - P'_P}{dys} \qquad \left.\frac{\partial P'}{\partial y}\right|_s = \frac{P'_P - P'_S}{dys}$$

$$\tilde{u}|_e = u_E \qquad \tilde{u}|_w = u_P \qquad \tilde{v}|_n = v_N \qquad \tilde{v}|_s = v_P$$

Substitute the approximations from table 5 into equation 31 and, after some algebraic manipulation and knowing our grid is uniform, i.e. $dxs = dyv = \Delta$, arrive at the discritized form of the pressure equation 32.

$$4P'_P - P'_E - P'_W - P'_N - P'_S = -\frac{\Delta}{\Delta t}\Big[\tilde{u}|_E - \tilde{u}|_P + \tilde{v}|_N - \tilde{v}|_P\Big] \tag{32}$$

Unlike the calculation of H_y and H_x the calculation of P' at each node will result in a pentadiagonal system of linear equations.

4.6 Pressure Boundaries

The boundary conditions used for the pressure correction equation are simply that the derivative of pressure normal to a wall is zero at the wall. For example, when computing the pressure at the $i = 1$ and $j = 3$ node, $\frac{\partial P'}{\partial x}|_w$ is zero. After setting this value to zero in equation 31 and some algebraic reduction, we are left with equation 33, which is a general expression for all nodes along the west wall.

$$3P'_P - P'_E - P'_N - P'_S = -\frac{\Delta}{\Delta t}\Big[\tilde{u}|_E - \tilde{u}|_P + \tilde{v}|_N - \tilde{v}|_P\Big] \tag{33}$$

This same procedure can be generalized to all the walls in the domain. Corners will also be treated in the same fashion, for example, at $i = 1$ and $j = 1$ both the south and west walls of the domain are in line with the south and west walls of the control volume. This means that both $\frac{\partial P'}{\partial x}|_w$ and

$\frac{\partial P^{'}}{\partial y}|_s$ will be zero. After setting these values to zero in equation 31 and some algebraic reduction, we are left with equation 34.

$$2P_P^{'} - P_E^{'} - P_N^{'} = -\frac{\Delta}{\Delta t}\Big[\tilde{u}|_E - \tilde{u}|_P + \tilde{v}|_N - \tilde{v}|_P\Big] \tag{34}$$

A similar procedure is used for all the boundaries and corners of the domain.

5 Description of solver

Following the general solution technique as outlined in section 3, a computer program was written to solve the lid driven cavity problem. The code was writen in functional C++ using the Microsoft Visual Studio 2010 integrated developer environment and compiler. The code uses a Gauss Seidel scheme to solve the pressure correction equation until the average $L2$ norm of the residual falls below 1E-5. The solution is integrated in time from rest until the $L2$ norm of the change in u and v velocities both falls below 1E-8. Solutions were computed at $RE = 100$ for various grid levels and time steps as contained in table 6.

Table 6: $RE = 100$ Cases computed when solving the lid driven cavity problem.

$N = 32$	Δt =3.125E-3	Δt =6.25E-3	Δt =0.0125
$N = 64$	Δt =5.0E-4	Δt =1.0E-3	Δt =3.0E-3
$N = 128$	Δt =1.25E-4	Δt =2.5E-4	Δt =5.0E-4

A solution was also computed for $RE = 1000$. For this case, the largest permissible time step was found and used to obtain convergence. The results for all of these cases are found in sections 6 and 7.

The complete source code for the computer program is found in Appendix A.

6 Results for RE=100

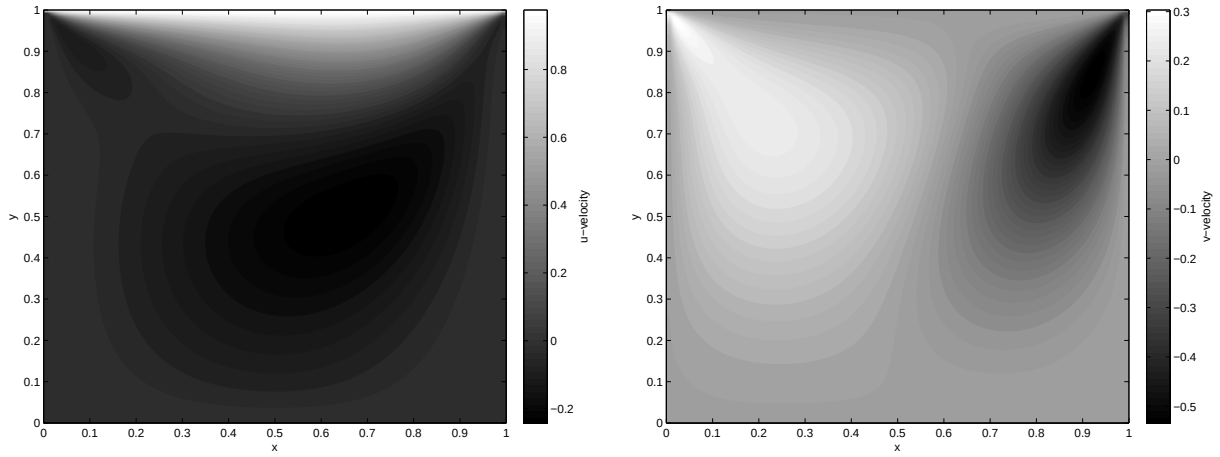
Figure 6 displays steady state u and v velocity contours solved for using the present computer program. These contours where made using the finest grid and time step from table 6

Figure 7 shows a vector plot of the steady state solution.

Figure 8 displays semi-logarithmic plots for the average $L2$ norm of the u-velocity residual plotted against non-dimensional time, t^*.

6.1 Trends in Convergence

By examining figure 8 several important trends appear. It is interesting that all the given time steps converge except the $\Delta t = 0.0125$ case for the $N = 32$ grid which diverges. In general, as the time step decreases, the nondimensional time to convergence goes down as well. To compare the effect of grid size, the Δt =5E-4 line from figure 8(b) can be compared to the Δt =5E-4 line from figure 8(c). Both of these cases converge in the same nondimensional time even though they use different grids.



(a) u-Velocity (b) v-Velocity

Figure 6: Steady state contours of velocity for the $RE = 100$ case. These contours where made using the finest grid and time step from table 6

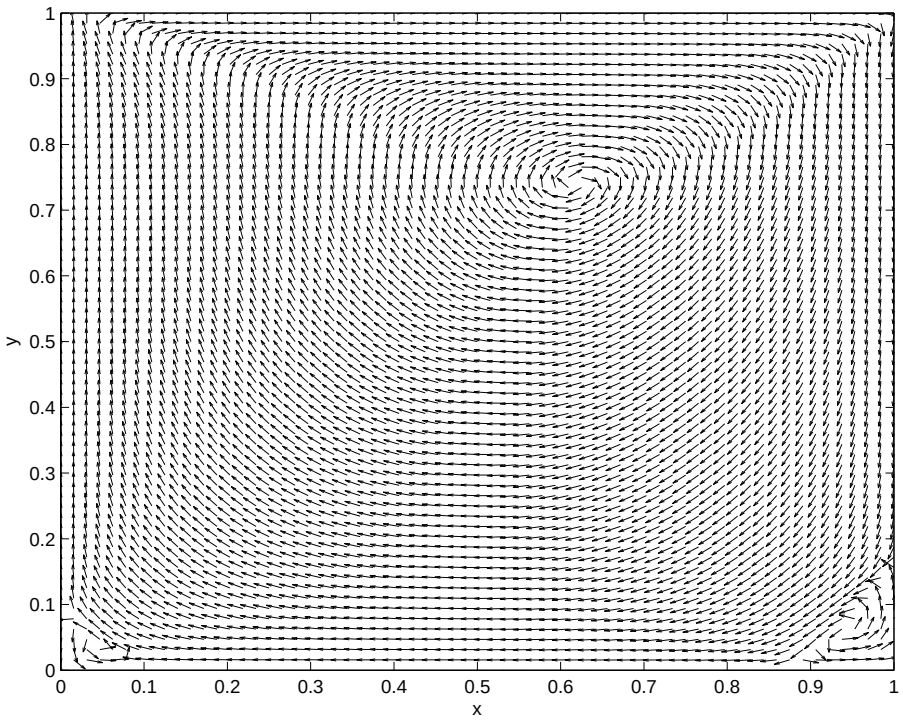
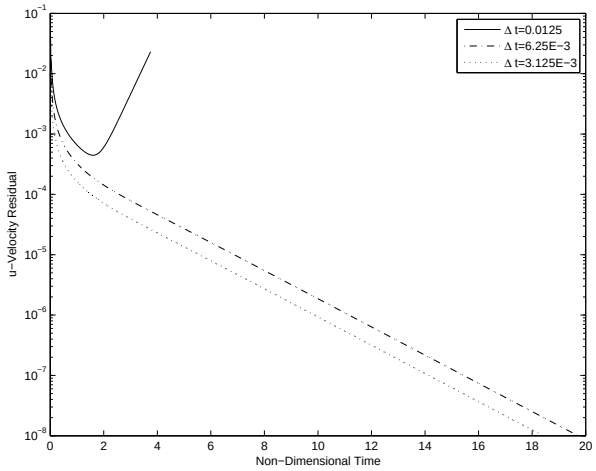
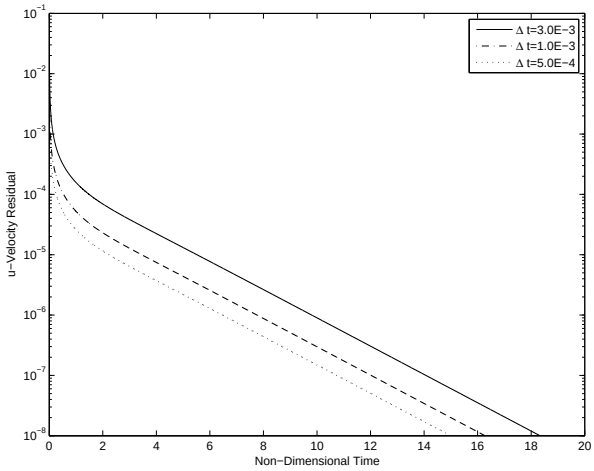


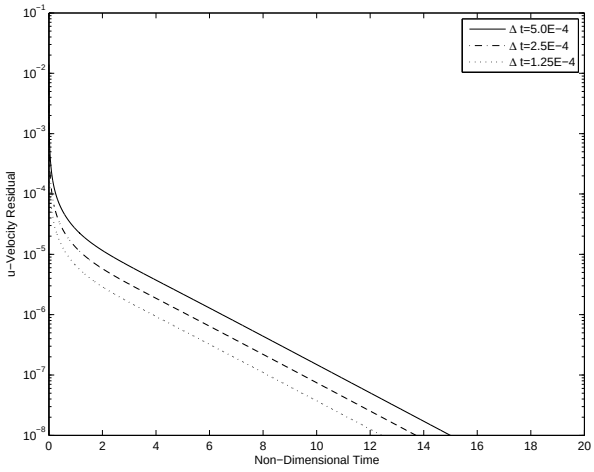
Figure 7: Vectors of velocity for the $RE = 100$ case. Note that these vectors are not plotted to scale. In reality the velocity near the bottom corner has a magnitude much lower than that in the center of the flow field. These vectors where made using the finest grid and time step from table 6 skipping every other velocity node.



(a) $N = 32$



(b) $N = 64$

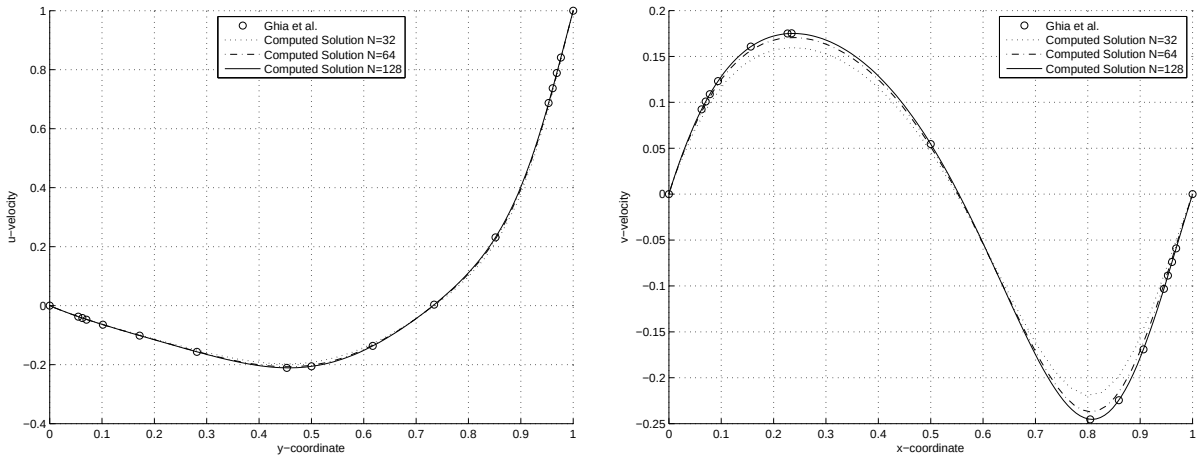


(c) $N = 128$

Figure 8: L2 norm of the u-velocity residual for the $RE = 100$ cases.

6.2 Validation

In order to validate the solution, it was compared against the solution obtain by Ghia,Ghia, and Shin[1]. Figure 9 shows the comparison of these results with the results of the present study.



(a) u-Velocity (b) v-Velocity

Figure 9: Comparison of u and v velocity to results found in [1, p 398-399]

As the grid resolution increases, the results from the present study approach those from Ghia, Ghia, and Shin. It should be noted that in their study, they also used $RE = 100$ and for all their computations they used $N = 128$.

6.3 Compute time

Table 7 shows the CPU time to convergence, the total number of time steps to convergence, as well as the total number of iterations used to solve the pressure correction equation for all the grid sizes and time steps found in table 6.

Table 7: CPU time, total time steps to convergence, and total number of pressure correction iterations performed for the $RE = 100$ cases. * this case did not converge

$N = 32$	Δt =3.125E-3	Δt =6.25E-3	Δt =0.0125
CPU time to convergence	2.97s	2.48s	*
Time steps	5,892	3,155	*
Pressure corrector iterations	19,937	16,568	*
$N = 64$	Δt =5.0E-4	Δt =1.0E-3	Δt =3.0E-3
CPU time to convergence	45.09s	29.28s	15.63s
Time steps	29,993	16,280	6,111
Pressure corrector iterations	54,737	47,639	36,124
$N = 128$	Δt =1.25E-4	Δt =2.5E-4	Δt =5.0E-4
CPU time to convergence	550.80s	348.54s	233.48s
Time steps	99,469	54,857	29,993
Pressure corrector iterations	134,039	118,441	101,987

The general trend is that as grid size increases, the CPU time to steady state increases very rapidly. The $N = 32$ cases converged on the order of 2 seconds on an Intel Core $i5-2500k$ quad core desktop

computer. The $N = 128$ cases took over 100 times as long to converge on the same computer. Also as time step decreased, CPU time increased dramatically. For the $N = 128$ case the largest time step took 233.48 while the finest time step took 550.80 to converge to steady state.

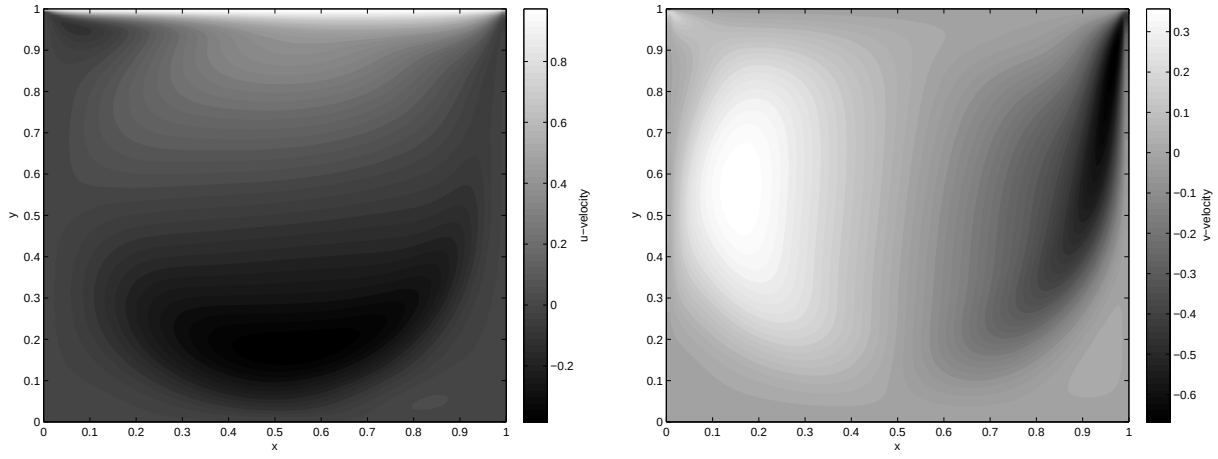
This can be explained as we examine the actual number of time steps being taken for each case. For the $N = 128$ grid and fine time step case, the solution takes $99,496$ time steps while for the coarse time step case it only takes $29,993$ steps. The finer case takes three times as many time iterations to reach a steady state. In this regard the nondimensional time is somewhat deceiving. Even though the finer case takes less nondimensional time to reach convergence, in order to do so it takes more time steps and more actual CPU time. The number of time steps are what counts when measuring CPU time as that will give insight as to how many computations were actually performed.

Another interesting comparison can be made when looking at the number of total pressure correction steps that are taken for each time step. It is interesting to note that the finer time step cases take more total pressure correction iterations. This should come as no surprise as we have already seen that they take more total time steps to converge. By comparing the total number of iterations done with the total number of time steps needed to reach convergence, we can see that even though finer time steps take more pressure iterations in total, they take fewer pressure iterations per time step.

We can conclude from this that at a finer time step the pressure correction is easier to solve. On the other hand, a finer time step means that the solution is advancing in time at a slower pace which translates to more total calculations and thus a longer CPU time to convergence. Also a time step that is too coarse can lead to divergence as was seen in the coarsest time step on the $N = 32$ grid.

7 Results for RE=1000

The solution was also solved on an $N = 128$ grid for $RE = 1000$. Figure 10 shows contours of u and v velocities for the $RE = 1000$ case.



(a) u-Velocity (b) v-Velocity

Figure 10: Contours of velocity at steady state for the $RE = 1000$ case.

Figure 11 shows vectors of velocity for the $RE = 1000$ case.

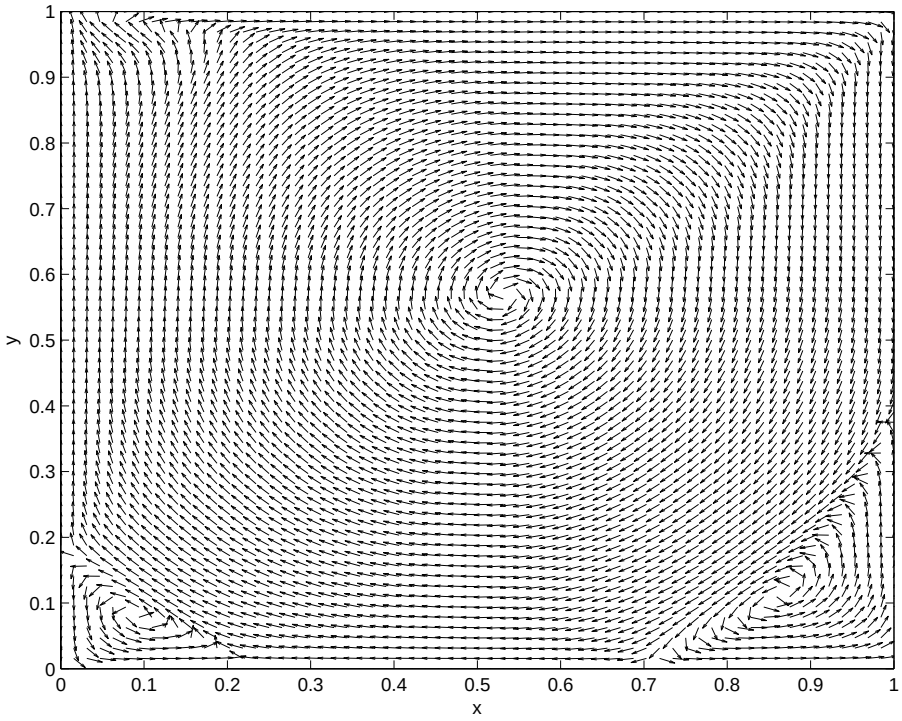
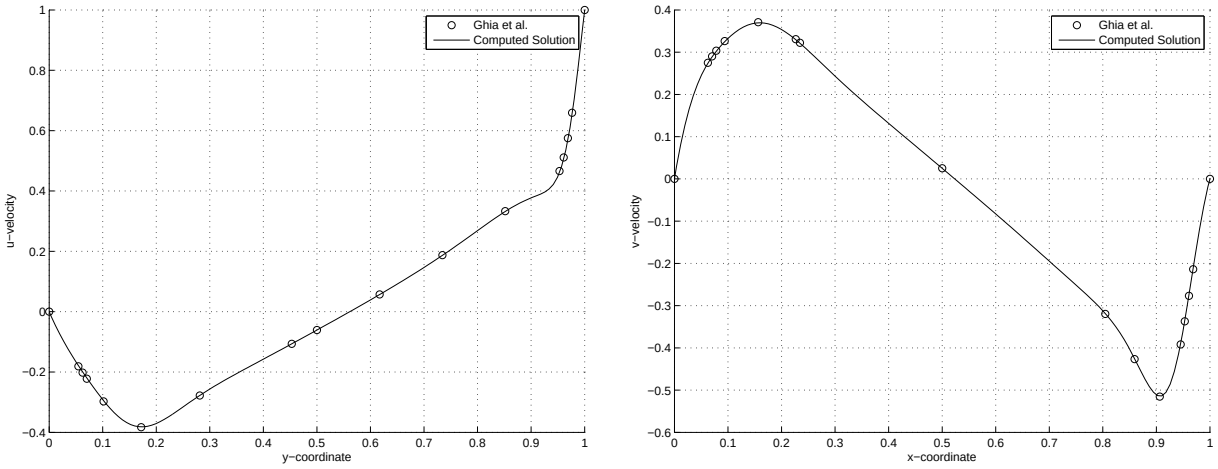


Figure 11: Vectors of velocity for the $RE = 1000$ case. Note that these vectors are not plotted to scale. In reality the velocity near the bottom corners has a magnitude much lower than that in the center of the flow field.

7.1 Validation

In order to validate the solution, it was once again compared to the values obtained by Ghia, Ghia, and Shin [1]. Figure 12 shows a graphical comparison of the two sets of data.



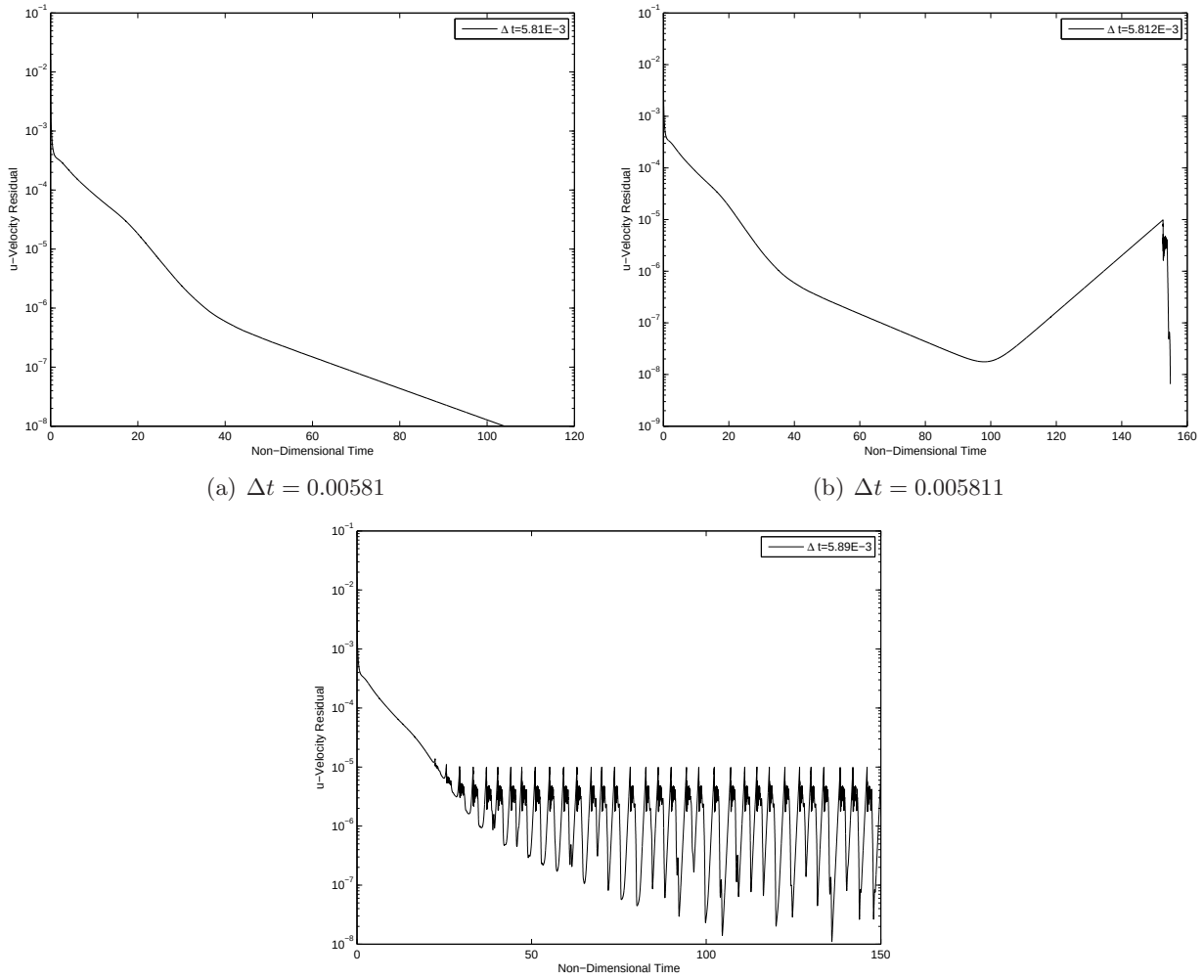
(a) u-velocity (b) v-velocity

Figure 12: Comparison of u and v velocity to results found in [1, p 398-399]

As can be seen in the figure, the computed solution matches the comparison data very well.

7.2 Maximum Permissible Time Step

A simple iterative technique was used in order to find the maximum permissible time step for the $N = 128$ and $RE = 1000$ case. The time step was incrementally increased until a divergence was detected. It was found that the maximum allowable time step was $\Delta t = 0.00581$. Figure 13 shows the L2 norm of the residual as a function of nondimensional time for three different cases. One for a time step slightly higher than the maximum, and one well above the maximum.



(a) $\Delta t = 0.00581$

(b) $\Delta t = 0.005811$

(c) $\Delta t = 0.0059$

Figure 13: L2 norm of the u-velocity residual for the $RE = 1000$ case at three different time step levels.

As can be seen in figure 13(a), the solution for $\Delta t = 0.00581$ smoothly converges to steady state. Once the time step is increased only slightly, the solution becomes unstable. This can be seen for $\Delta t = 0.005811$ in figure 13(b). At $\Delta t = 0.0059$, as shown figure 13(c) the solution becomes wildly unstable.

7.3 Comparison to $RE = 100$

A similar procedure was carried out for the $RE = 100$ case and a maximum permissible time step was found to be $\Delta t = 0.000731$. It is interesting that the maximum allowable time step increases with increasing RE. The fundamental difference between the two cases is that in the $RE = 100$ case, the viscous forces on the fluid are much greater than the inertial forces on the fluid. The opposite is true for the $RE = 1000$ case. This is because the Reynold's number is the ratio of inertial to viscous forces.

With this in mind, we can examine the stability further by computing the CFL number and the Von Neumann stability number, r, using equations 35 and 36.

$$CFL = \frac{u\Delta t}{\Delta x} \tag{35}$$

$$r = \frac{\mu\Delta t}{\Delta x^2} \tag{36}$$

In equation 35 a u of 1, consistent with our nondimensionalization, was used. In equation 36 a μ is derived from the Reynolds number at each case. In both equations, the maximum permissible time step found previously was used as Δt. Table 8 shows the results of computing both the CFL number and the Von Neumann stability number, r, for both RE cases as well as the nondimensional time to convergence.

Table 8: CFL, Von Nueman stability conditions, and non dimensional time to convergence for $RE = 100$ and $RE = 1000$ cases using their computed maximum permissible timestep.

RE	CFL	r	**Non dimensional time to convergence**
100	0.0977	0.1250	15.78
1000	0.7437	0.0952	104.02

It is clear from the table that at $RE = 100$, r is much closer to its critical value, 1/2, than the CFL number is to its critical value, 1. The opposite is true for the $RE = 1000$ case. Since the $RE = 100$ case is more viscous dominated it is more controlled by the Von Nueman stability condition. As the $RE = 1000$ is more convection dominated, it is more controlled by the CFL number. The reason that we are not seeing either the CFL number or the Von Nueman stability number approach its critical value is that, in reality, both cases are dominated by both conditions.

Another interesting comparison is the different nondimensional time that each case takes to reach convergence. From the table, the $RE = 1000$ case takes roughly 6 times as long to converge as the $RE = 100$ case. This is due to the decreased level of viscous damping in the $RE = 1000$ case. This can be seen visually by comparing the velocity vectors from the $RE = 1000$ case, figure 11, with velocity vectors from the $RE = 100$ case, figure 7. In the $RE = 1000$ case, the flow structures are much larger and more complicated which will cause the solution to converge slower.

8 Potential Speed Improvements

The code in the present study was developed for instructional purposes only. It was not optimized in any way. Apart from obvious code and compiler optimization or potential parallelization, some algorithmic changes could be made to potentially decrease the time to solution.

One thing that could be done to greatly increase the speed of the pressure corrector calculation

would be to implement a multigrid framework. Hoopes, in his study of the application of the multigrid framework to the 2D Poisson equation, found a speed improvement of almost two times with only one level of multigrid[2, p 2]. If multiple levels of multigrid were applied, even more speedup would be anticipated.

Other changes that could be implemented would be to use an implicit time integration scheme. This could possibly decrease total time to solution as it would allow a larger time step to be taken. Care would have to be taken to ensure that the extra computation involved in an implicit time integration scheme would not offset the gains produced by increasing the time step.

Another improvement would be replacing the Guass Sidel solver that was used for the pressure corrector equation with a different, possibly faster, method. The Alternating Direction Implicit method could be implemented, it uses a tridiagonal solver which, if properly optimized, offers potential speed benefits.

9 Conclusion

The lid driven cavity problem on a square domain was solved for $RE = 100$ and $RE = 1000$ cases using a staggered grid and an predictor corrector solution method. The steady state solution at $RE = 100$ and $RE = 1000$ was compared to the results obtained by Ghia, Ghia, and Shin[1] and the results were found to be in good agreement. For the $RE = 1000$ case, the largest permissible time step of $\Delta t = 0.00581$ was found. Some conclusions were drawn about the stability of the solution by comparing the maximum permissible time step for the $RE = 100$ case and the $RE = 1000$ case. It was found that the lower RE value is more strictly controlled by viscous terms and thus the Von Neumann stability condition while the high RE case is more dominated by the CFL condition.

References

[1] Ghia, Ghia, and Shin, *High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method*. Journal of Computational Physics, Vol. 48, pp. 387-411, 1982.

[2] Hoopes, Kevin, *CFD HW 6: Multigrid framework applied to the 2D Poisson equation*. 2012.

Appendix A - Code

```
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <math.h>

using std::vector;
using namespace std;
int dimension;
double delta;
double RE;
double delta_t;
int write_interval;
int report_interval;


int print_2d (vector<vector<double> > array2D, int dimension, string filename) {
        ofstream outfile;
        outfile.open(filename);

        for (int j=0; j<dimension; j++){

                for (int i=0; i<dimension; i++) {
                        outfile << array2D[i][j] << " ";
                }
                outfile << endl;

        }
        outfile.close();

        return 1;
}

int clear_array(vector<vector<double> > &array2d) {
        for (int i = 0; i < dimension; i++) {
                for(int j=0; j < dimension; j++) {
                        array2d[i][j] = 0;
                }
        }
        return 1;
}


double calcHi_x(vector<vector<double> > &u, vector<vector<double> > &v, int i, int j
    ) {


        double u_e,u_w,u_n,v_n,u_s,v_s;
        double dudx_e,dudx_w,dudy_n,dudy_s;

        double Hi;

        //set volume side lengths
        double dyv,dxs;

        dyv = delta;
```

```
dxs = delta;

//set neighborhood
int P_i,P_j,N_i,N_j,E_i,E_j,S_i,S_j,W_i,W_j;
P_i = i;
P_j = j;
N_i = i;
N_j = j+1;
E_i = i+1;
E_j = j;
S_i = i;
S_j = j-1;
W_i = i-1;
W_j = j;

//set extendened neighborood
int NW_i,NW_j;
NW_i = i-1;
NW_j = j+1;

//set default values to calculate Hi and Hi_old

//face vel
u_n = 0.5*(u[N_i][N_j]+u[P_i][P_j]);
u_e = 0.5*(u[E_i][E_j]+u[P_i][P_j]);
u_s = 0.5*(u[S_i][S_j]+u[P_i][P_j]);
u_w = 0.5*(u[W_i][W_j]+u[P_i][P_j]);


v_n = 0.5*(v[NW_i][NW_j]+v[N_i][N_j]);
v_s = 0.5*(v[W_i][W_j]+v[P_i][P_j]);

//face deriv
dudy_n = 1.0/delta*(u[N_i][N_j]-u[P_i][P_j]);
dudx_e = 1.0/delta*(u[E_i][E_j]-u[P_i][P_j]);
dudy_s = 1.0/delta*(u[P_i][P_j]-u[S_i][S_j]);
dudx_w = 1.0/delta*(u[P_i][P_j]-u[W_i][W_j]);



//set boundary specific values

//left
if(i == 2) {
        //cout << u[W_i][W_j] << endl;

}

//right
if (i == dimension-2) {
}

//bottom
if (j == 1) {
        u_s = u[S_i][S_j];
        u_s = 0;
        dudy_s = 1.0/(0.5*delta)*(u[P_i][P_j]-u[S_i][S_j]);

}
```

```
        //top
        if(j == dimension-2){
                u_n = 1;
                dudy_n = 1.0/(0.5*delta)*(u_n-u[P_i][P_j]);
        }

        //calculate Hi
        Hi = 1.0/RE*((dudx_e - dudx_w)*dyv + (dudy_n - dudy_s)*dxs) - (u_e*u_e - u_w
            *u_w)*dyv - (u_n*v_n - u_s*v_s)*dxs;

        return Hi;


}

double calcHi_y(vector<vector<double> > &u, vector<vector<double> > &v, int i, int j
    ) {


        double v_n,v_e,v_s,v_w,u_e,u_w;
        double dvdx_e,dvdx_w,dvdy_n,dvdy_s;

        double Hi;

        //set volume side lengths
        double dyv,dxs;

        dyv = delta;
        dxs = delta;

        //set neighborhood
        int P_i,P_j,N_i,N_j,E_i,E_j,S_i,S_j,W_i,W_j;
        P_i = i;
        P_j = j;
        N_i = i;
        N_j = j+1;
        E_i = i+1;
        E_j = j;
        S_i = i;
        S_j = j-1;
        W_i = i-1;
        W_j = j;

        //set extendened neighborood
        int SE_i,SE_j;
        SE_i = i+1;
        SE_j = j-1;

        //set default values to calculate Hi and Hi_old

        //face vel
        v_n = 0.5*(v[N_i][N_j]+v[P_i][P_j]);
        v_e = 0.5*(v[E_i][E_j]+v[P_i][P_j]);
        v_s = 0.5*(v[S_i][S_j]+v[P_i][P_j]);
        v_w = 0.5*(v[W_i][W_j]+v[P_i][P_j]);


        u_e = 0.5*(u[SE_i][SE_j]+u[E_i][E_j]);
```

```
        u_w = 0.5*(u[S_i][S_j]+u[P_i][P_j]);

        //face deriv
        dvdy_n = 1.0/delta*(v[N_i][N_j]-v[P_i][P_j]);
        dvdx_e = 1.0/delta*(v[E_i][E_j]-v[P_i][P_j]);
        dvdy_s = 1.0/delta*(v[P_i][P_j]-v[S_i][S_j]);
        dvdx_w = 1.0/delta*(v[P_i][P_j]-v[W_i][W_j]);



        //set boundary specific values

        //left
        if(i == 1) {
                v_w = v[W_i][W_j];

                dvdx_w = 1.0/(0.5*delta)*(v[P_i][P_j]-v[W_i][W_j]);

        }

        //right
        if (i == dimension-2) {
                v_e = v[E_i][E_j];

                dvdx_e = 1.0/(0.5*delta)*(v_e - v[P_i][P_j]);

        }

        //bottom
        if (j == 2) {

        }

        //top
        if(j == dimension-2){
        }

        //calculate Hi
        Hi = 1.0/RE*((dvdx_e - dvdx_w)*dyv + (dvdy_n - dvdy_s)*dxs) - (u_e*v_e - u_w
            *v_w)*dyv - (v_n*v_n - v_s*v_s)*dxs;

        return Hi;


}

double calcPgradx(vector<vector<double> > &P, int i,int j) {

        double GP;

        GP = 1.0/delta*(P[i][j] - P[i-1][j]);

        return GP;

}

double calcPgrady(vector<vector<double> > &P, int i,int j) {

        double GP;
```

```
        GP = 1.0/delta*(P[i][j]-P[i][j-1]);

        return GP;

}
int main(int argc, const char* argv[]) {

        if ( argc != 6) {
                cout << "You had " << argc-1 << " Arguments " << "You needed " <<
                    6-1 << endl;
                return 0;
        }
        dimension = atoi(argv[1]);
        delta_t = atof(argv[2]);
        RE = atof(argv[3]);
        report_interval = atoi(argv[4]);
        write_interval = atoi(argv[5]);


        //dimension = 33;
        //delta_t = .003125;
        //write_interval = 1000;
        //RE = 100.0;

        cout << endl << "Lid driven cavity solver - Kevin Hoopes May 2012" << endl
            << endl;
        cout << "Dimension:" << dimension << endl;
        cout << "Delta t:" << delta_t << endl;
        cout << "RE:" << RE << endl;
        cout << "Report interval:" << report_interval << endl;
        cout << "Write interval:" << write_interval << endl;
        cout << endl << "Begining time itegration to steady state" << endl;

        //adjust dimension
        dimension = dimension + 2 ;
        //cout << dimension;

        if(write_interval == 0) {
                write_interval = 1e9;
        }

        if(report_interval == 0) {
                report_interval = 1e9;
        }



        //start L2 writer
        ofstream L2file;
        stringstream L2filename;
        L2filename << "results/L2error-" << dimension - 2 << "-" << delta_t << "-"
            << RE << ".txt";
        L2file.open(L2filename.str());


        delta = 1.0/(dimension-2);

        //create arrays to hold our information
```

```
        //u-vel
        vector<vector<double> > u;
        u.resize(dimension);
        for (int i = 0; i < dimension; i++) {
                u[i].resize(dimension);
        }

        vector<vector<double> > ustar;
        ustar.resize(dimension);
        for (int i = 0; i < dimension; i++) {
                ustar[i].resize(dimension);
        }

        vector<vector<double> > u_old;
        u_old.resize(dimension);
        for (int i = 0; i < dimension; i++) {
                u_old[i].resize(dimension);
        }

        //v-vel
        vector<vector<double> > v;
        v.resize(dimension);
        for (int i = 0; i < dimension; i++) {
                v[i].resize(dimension);
        }

        vector<vector<double> > vstar;
        vstar.resize(dimension);
        for (int i = 0; i < dimension; i++) {
                vstar[i].resize(dimension);
        }

        vector<vector<double> > v_old;
        v_old.resize(dimension);
        for (int i = 0; i < dimension; i++) {
                v_old[i].resize(dimension);
        }

        //Pressure
        vector<vector<double> > P;
        P.resize(dimension);
        for (int i = 0; i < dimension; i++) {
                P[i].resize(dimension);
        }

        vector<vector<double> > Pprime;
        Pprime.resize(dimension);
        for (int i = 0; i < dimension; i++) {
                Pprime[i].resize(dimension);
        }


//set initial conditions
clear_array(u);

//set top velocity
for(int i=0; i<dimension; i++){
        for(int j=0; j<dimension;j++){
```

```
                if(j == dimension - 1 && i !=0 && i!=dimension-1) {
                        u[i][j] = 1;
                }

        }
}

clear_array(u_old);
clear_array(ustar);

clear_array(v);
clear_array(v_old);
clear_array(vstar);

clear_array(P);
clear_array(Pprime);



//start time integration loop
for (int m=1; m < 2e11; m++ ) {
        //clear all arrays, set all values to zero for intermediate things.
        clear_array(ustar);
        clear_array(vstar);
        clear_array(Pprime);
        //predict u and v

                //predict u
                for (int i=2; i <= dimension-2; i++){
                        for (int j=1; j <= dimension-2; j++){

                                double Hi,Hi_old;
                                double GP;

                                //setup variables for ustar computation
                                Hi = calcHi_x(u,v,i,j);

                                //calculate Hi_old could also just remember
                                    it
                                Hi_old = calcHi_x(u_old,v_old,i,j);

                                //calculate GP
                                GP = calcPgradx(P,i,j);

                                //predict u
                                if (m == 1) {
                                        ustar[i][j] = u[i][j] + delta_t/pow(
                                            delta,2.0)*Hi - delta_t*GP;
                                } else {
                                        ustar[i][j] = u[i][j] + delta_t/pow(
                                            delta,2.0)*(1.5*Hi - 0.5*Hi_old)
                                            - delta_t*GP;
                                }

                        }
                }
```

```
//predict v
for (int i=1; i <= dimension-2; i++){
        for (int j=2; j <= dimension-2; j++){

                double Hi,Hi_old;
                double GP;

                //setup variables for ustar computation
                Hi = calcHi_y(u,v,i,j);

                //calculate Hi_old could also just remember
                    it
                Hi_old = calcHi_y(u_old,v_old,i,j);

                //calculate GP
                GP = calcPgrady(P,i,j);

                //predict v
                if (m ==1) {
                        vstar[i][j] = v[i][j] + delta_t/pow(
                            delta,2.0)*Hi - delta_t*GP;
                } else {
                        vstar[i][j] = v[i][j] + delta_t/pow(
                            delta,2.0)*(1.5*Hi - 0.5*Hi_old)
                            - delta_t*GP;
                }
        }
}

//print_2d(vstar,dimension,"vstar_jacob_out.txt");

//solve for a pressure correction until L2 error norm < 1e-5
int pressure_iterations = 0;
for (int n=0; n< 1000000; n++) {

        for (int j=1; j <= dimension -2; j++){
                for (int i=1; i <= dimension -2; i++) {

                        double Pprime_N,Pprime_E,Pprime_S,
                            Pprime_W;

                        double AP,AN,AE,AS,AW;
                        double ustar_e,ustar_w;
                        double vstar_n,vstar_s;

                        //set neighborhood
                        int P_i,P_j,N_i,N_j,E_i,E_j,S_i,S_j,
                            W_i,W_j;
                        P_i = i;
                        P_j = j;
                        N_i = i;
                        N_j = j+1;
                        E_i = i+1;
                        E_j = j;
                        S_i = i;
                        S_j = j-1;
                        W_i = i-1;
                        W_j = j;
```

```
                    //set default values;

                    Pprime_N = Pprime[N_i][N_j];
                    Pprime_E = Pprime[E_i][E_j];
                    Pprime_S = Pprime[S_i][S_j];
                    Pprime_W = Pprime[W_i][W_j];

                    AP = 4.0;
                    AN = 1.0;
                    AE = 1.0;
                    AS = 1.0;
                    AW = 1.0;

                    ustar_e = ustar[i+1][j];
                    ustar_w = ustar[i][j];

                    vstar_n = vstar[i][j+1];
                    vstar_s = vstar[i][j];



                    if (i == 1) {
                            AP = AP-1;
                            AW = 0;

                    }

                    if (i == dimension - 2) {
                            AP = AP-1;
                            AE = 0;

                    }
                    if (j == 1) {
                            AP = AP-1;
                            AS = 0;

                    }
                    if (j == dimension - 2) {
                            AP = AP-1;
                            AN = 0;
                    }


                    //now assemble our equation
                    double LHS = AN*Pprime_N + AE*
                        Pprime_E + AS*Pprime_S + AW*
                        Pprime_W;
                    double RHS = -delta/delta_t*((
                        ustar_e-ustar_w) + (vstar_n-
                        vstar_s));

                    Pprime[P_i][P_j] = 1.0/AP*(LHS + RHS
                        );

            }
    }

    //calculate rho the error
```

```
double Prho_sum = 0;
for (int j=1; j <= dimension -2; j++){
        for (int i=1; i <= dimension - 2; i++) {

                double Pprime_P, Pprime_N, Pprime_E,
                    Pprime_S, Pprime_W;

                double AP, AN, AE, AS, AW;
                double ustar_e, ustar_w;
                double vstar_n, vstar_s;

                //set neighborhood
                int P_i, P_j, N_i, N_j, E_i, E_j, S_i, S_j,
                    W_i, W_j;
                P_i = i;
                P_j = j;
                N_i = i;
                N_j = j+1;
                E_i = i+1;
                E_j = j;
                S_i = i;
                S_j = j-1;
                W_i = i-1;
                W_j = j;

                //set default values;

                Pprime_P = Pprime[P_i][P_j];
                Pprime_N = Pprime[N_i][N_j];
                Pprime_E = Pprime[E_i][E_j];
                Pprime_S = Pprime[S_i][S_j];
                Pprime_W = Pprime[W_i][W_j];

                AP = 4.0;
                AN = 1.0;
                AE = 1.0;
                AS = 1.0;
                AW = 1.0;

                ustar_e = ustar[i+1][j];
                ustar_w = ustar[i][j];

                vstar_n = vstar[i][j+1];
                vstar_s = vstar[i][j];



                if (i == 1) {
                        AP = AP - 1;
                        AW = 0;
                }
                if (i == dimension - 2) {
                        AP = AP - 1;
                        AE = 0;

                }
                if (j == 1) {
                        AP = AP - 1;
                        AS = 0;
```

```
                                }
                                if (j == dimension - 2) {
                                        AP = AP - 1;
                                        AN = 0;
                                }

                                double LHS,RHS;

                                LHS = AP*Pprime_P - AE*Pprime_E - AW
                                    *Pprime_W - AN*Pprime_N - AS*
                                    Pprime_S;
                                RHS = -delta/delta_t*((ustar_e-
                                    ustar_w) + (vstar_n-vstar_s));
                                Prho_sum =  Prho_sum + pow((LHS -
                                    RHS),2.0);


                        }
                }

                double L2_norm = 0;

                L2_norm = pow(1.0/pow((dimension-2),2.0)*Prho_sum
                    ,0.5);
                //cout << L2_norm << endl;
                if (L2_norm < 1e-5) {
                        //cout << "Timestep " << m << " : " << m*
                            delta_t << " Pprime converged in " << n
                            << " iterations. with L2 nrom of " <<
                            L2_norm << endl;
                        break;
                }

                //cout << L2_norm << endl;

                if (L2_norm > 10e3) {
                        cout << "Divergence detected on iter " << m
                            << "!" << endl;
                        return 0;
                }

                pressure_iterations++;
        }

        //correct u and v


        //update u_old
        for (int i=0; i<dimension; i++){
                for (int j=0; j<dimension; j++){

                        u_old[i][j] = u[i][j];

                }
        }

        //update v_old
        for (int i=0; i<dimension; i++){
```

```
        for (int j=0; j<dimension; j++){

                v_old[i][j] = v[i][j];

        }
}


//correct u
double rho_u = 0;
double rho_v = 0;
for (int i=2; i <= dimension - 2; i++){
        for (int j=1; j <= dimension - 2; j++){

                u[i][j] = ustar[i][j] - delta_t*calcPgradx(
                    Pprime,i,j);
                rho_u = rho_u + pow(u[i][j] - u_old[i][j
                    ],2.0);

        }
}


//correct v
for (int i=1; i <= dimension-2; i++){
        for (int j=2; j <= dimension-2; j++){

                v[i][j] = vstar[i][j] - delta_t*calcPgrady(
                    Pprime,i,j);
                rho_v = rho_v + pow(v[i][j] - v_old[i][j
                    ],2.0);
        }
}



//check to make sure continuity holds
int i = 10;
int j = 10;

if (m % 1000 == 0) {
//      double cont = (u[i][j]*delta+v[i][j]*delta) - (u[i
    +1][j]*delta+v[i][j+1]*delta);
//      cout << "Continutiy check: " << cont << endl;
}

//update P

for (int j=1; j < dimension -1; j++){
        for (int i=1; i < dimension -1; i++) {

                P[i][j] = P[i][j] + Pprime[i][j];

        }
}
```

```
//check stop criteria
double L2_norm_u,L2_norm_v;
L2_norm_u = pow(1.0/pow((dimension-2),2.0)*rho_u,0.5);
L2_norm_v = pow(1.0/pow((dimension-2),2.0)*rho_v,0.5);

L2file << m << " " << L2_norm_u << " " << L2_norm_v << " "
   << pressure_iterations << endl;


if (L2_norm_u < 1e-8 && L2_norm_v < 1e-8) {
        cout << "Solution converged to steady state in " <<
            m << " timesteps, congratulations." << endl;

        stringstream u_end_filename;
        u_end_filename << "results/uend-" << dimension - 2
            << "-" << delta_t << "-" << RE << ".txt";
        print_2d(u,dimension,u_end_filename.str());

        stringstream v_end_filename;
        v_end_filename << "results/vend-" << dimension - 2
            << "-" << delta_t << "-" << RE << ".txt";
        print_2d(v,dimension,v_end_filename.str());

        //stringstream P_end_filename;
        //P_end_filename << "Pend-" << dimension << "-" <<
            delta_t << "-" << RE << ".txt";
        //print_2d(P,dimension,P_end_filename.str());

        L2file.close();

        break;
}


if (m % report_interval == 0){
        cout << "timestep:" << m << " Pressure converged in
            " << pressure_iterations << " iterations, u
            solution L2: " << L2_norm_u << endl;

}

if (m % write_interval == 0){
        //cout << "Wrote iter: " << m << " Pressure
            converged in " << pressure_iterations << "
            iterations, Solution L2: " << L2_norm_u << endl;

        stringstream ufilename;
        ufilename << "results/u-" << dimension << "-" <<
            delta_t << "-" << RE << "_" << m << ".txt";
        print_2d(u,dimension,ufilename.str());

        stringstream vfilename;
        vfilename << "results/v-" << dimension << "-" <<
            delta_t << "-" << RE << "_" << m << ".txt";
        print_2d(u,dimension,vfilename.str());

}
```

```
                }


        return 0;
}
```