MIE 1210: Computational Fluid Mechanics and Heat Transfer, Assignment 4: Developing a Finite volume Lid Driven Cavity solver using the SIMPLE algorithm in Python

Declan Bracken[1, *]

[1] *Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, ON M5R 0A3, Canada*

(Dated: December 21st, 2023)

I. INTRODUCTION

A. Navier-Stokes Non-dimensionalization

The purpose of this assignment was the construction of a 2-dimensional finite volume computational fluid dynamics solver which could solve an incompressible, steady state fluid flow within a rectangular domain. The sample problem of choice is the famous lid driven cavity problem, to which the SIMPLE (semi-implicit method for pressure linked equations) algorithm originally proposed by Spalding and Patankar is applied. The algorithm aims to solve the non-linear momentum equation described by the incompressible, steady state Navier-Stokes equation with no gravity term [1];

$$\nabla \cdot \vec{u}\vec{u} = \nabla \cdot \nu \nabla \vec{u} - \rho^{-1} \nabla p. \tag{1}$$

Where $\vec{u}$ is the vector velocity field in 2 dimensions, ∇ is a vector operator denoting divergence or gradient (depending on whether it used directly or with a dot product), ν is the kinematic viscosity of the fluid, ρ is the density of the fluid, and p is the pressure. The solution to this equation must also satisfy the continuity equation for incompressible flow;

$$\nabla \cdot \vec{u} = 0 \tag{2}$$

To solve for all 3 variables $\vec{u}$, $\vec{v}$, and p, an iterative approach using the SIMPLE algorithm will be performed. We can begin this approach by non-dimensionalizing equation (1) by introducing characteristic scales for length, velocity, and pressure;

- Velocity: $\vec{u}^* = \vec{u}/U$
- Spatial coordinates: $x^* = x/L$, $y^* = y/L$
- Pressure: $p^* = p/P$

Substituting these into the Navier-Stokes equations and choosing $P = \rho U^2$ as the characteristic pressure scale, we obtain the non-dimensional Navier-Stokes equations:

- Continuity equation:

$$\nabla \cdot \vec{u}^* = 0 \tag{3}$$

- Momentum equation:

$$\nabla \cdot \vec{u}^* \vec{u}^* = \frac{1}{Re} \nabla^2 \vec{u}^* - \nabla p^* \tag{4}$$

Here, $Re = \frac{\rho U L}{\nu}$ is the Reynolds number, representing the ratio of inertial forces to viscous forces in the flow. The choice of scales might vary based on the specific physical problem under consideration. For our problem, we'll be tackling a lid driven cavity in a square box with side lengths $l_x, l_y = 1$, and the lid operates at a velocity of 1 m/s. We could also assume a density near 1 for water, but it's irrelevant. Since the characteristic lengths and speeds are each on the scale of 1, varying the Reynolds number effectively changes the ratio of density to viscosity. For convenience, the * superscript will be dropped from the terms for the remainder if the discretization of the momentum and continuity equations during the derivation of the SIMPLE algorithm. The finite volume integration over the momentum equation can be taken as;

$$\int_A \mathbf{n} \cdot (\rho \vec{u}\vec{u})\, dA = \int_A \mathbf{n} \cdot (\Gamma \nabla \vec{u})\, dA + \int_{CV} S\, dV. \tag{5}$$

In this equation, the right hand side represents the transport of a scalar quantity through diffusive means, and a source term which encompasses the pressure gradient across the volume. The left hand side of (5) represents the transport of the state variable through convective means, IE transportation through movement of the fluid. In this case, the scalar quantity is actually a component of the velocity field in 2D; u or v. To represent this equation in it's most general form we'll rewrite it in terms of a scalar quantity ϕ, which will represent u or v;

$$\int_A \mathbf{n} \cdot (\rho \mathbf{u} \phi)\, dA = \int_A \mathbf{n} \cdot (\Gamma \nabla \phi)\, dA + \int_{CV} S_\phi\, dV. \tag{6}$$

For a 2D problem, and if we drop the source term for simplicity, this integration yields the partial differential equation:

$$(\rho u A \phi)_e - (\rho u A \phi)_w - (\rho v A \phi)_n + (\rho v A \phi)_s = \left[\Gamma A \frac{\partial \phi}{\partial x}\right]_e - \left[\Gamma A \frac{\partial \phi}{\partial x}\right]_w - \left[\Gamma A \frac{\partial \phi}{\partial y}\right]_n + \left[\Gamma A \frac{\partial \phi}{\partial y}\right]_s, \tag{7}$$

with continuity equation;

$$(\rho u A)_e - (\rho u A)_w + (\rho v A)_n - (\rho v A)_s = 0. \tag{8}$$

* declan.bracken@mail.utoronto.ca

Where the subscripts n, s, e, and w correspond to the cardinal directions of the 2d grid denoting north, south, east and west when viewing the plane. It is then useful to define convective and diffusive coefficients for the discretization separately:

$$F_e = (\rho u)_e \qquad F_w = (\rho u)_w \tag{9}$$

$$D_e = \frac{\Gamma_e}{\delta x_e} \qquad D_w = \frac{\Gamma_w}{\delta x_w} \tag{10}$$

$$F_n = (\rho v)_n \qquad F_s = (\rho v)_s \tag{11}$$

$$D_n = \frac{\Gamma_n}{\delta y_n} \qquad D_s = \frac{\Gamma_s}{\delta y_s} \tag{12}$$

Where the F terms are convective fluxes and the D terms are diffusive fluxes per unit area in any direction.

To continue with the discretization, we must decide on an advection scheme to calculate the gradients $\frac{\partial \phi}{\partial x}$ and $\frac{\partial \phi}{\partial y}$. We will select between 2 schemes: central difference and first order upwind.

B. Upwind Scheme

The central difference scheme works well for a pure diffusion problem because it weighs each cardinal direction equally in how the state variable is distributed. In the case of diffusion this is true, however, applying the same scheme to convection could cause problems. This is because with vectorized convection terms, transport is mostly only occurring in the direction of the velocity field. We use the term 'transportiveness' to describe an advection scheme's capability to represent directional transport by weighing the upstream (or upwind) coefficients more than downwind. We saw in assignment 3 how the implementation of the central difference scheme in the presence of a vector field can yield issues with boundedness, and for the lid driven cavity problem using a central difference scheme could erase important fluid dynamics phenomena such as eddy flows in the corners of the rectangular domain.

For the upwind scheme, we adopt a gradient discretization which is flow direction dependant. Take a 1D example from [1]. For a positive (rightward) flow, ϕ_e becomes ϕ_p and ϕ_w becomes ϕ_W, effectively shifting flux faces towards the upstream direction, yielding:

$$F_e\phi_P - F_w\phi_W = D_e(\phi_E - \phi_P) - D_w(\phi_P - \phi_W) \tag{13}$$

If the flow is negative (leftward) however, then the process would be reversed, and ϕ_w that would become ϕ_p, while ϕ_e becomes ϕ_E:

$$F_e\phi_E - F_w\phi_P = D_e(\phi_E - \phi_P) - D_w(\phi_P - \phi_W) \tag{14}$$

Incorporating the same process of grouping like terms will yield neighboring coefficients in the same way as it did for central difference, but this time there is flow direction dependence:

When populating a dependency matrix, it's inefficient to use if-statements to constantly check flow direction in order to adjust the coefficient, which is why we can use a *max* function to check flow direction and adjust coefficients accordingly:

Condition	a_W, a_s	a_e, a_n
$F_w > 0, F_e > 0$	$D_w + F_w$	D_e
$F_w < 0, F_e < 0$	D_w	$D_e - F_e$
$F_s > 0, F_n > 0$	$D_s + F_s$	D_n
$F_s < 0, F_n < 0$	D_s	$D_n - F_n$

Table I: Coefficients for discretized transport equations considering flow direction.

a_W	$D_w + \max(F_w, 0)$
a_E	$D_e + \max(0, -F_e)$
a_S	$D_s + \max(F_s, 0)$
a_N	$D_n + \max(0, -F_n)$

Table II: Coefficient calculations for 2D discretized transport equations.

II. DERIVATION OF SIMPLE ALGORITHM FOR A CO-LOCATED MESH

We begin the simple derivation by starting with the equations for momentum and mass balance along with the source terms as outlined in the previous section. The source term in equation (6) can be separated between x and y components;

$$S^x = (p_w - p_e)\Delta y, \tag{15a}$$

$$S^y = (p_s - p_n)\Delta x. \tag{15b}$$

To understand the SIMPLE algorithm on a colocated mesh, it's expected that the reader has a good understanding of it's application on a staggered grid where different variables such as velocity and pressure are calculated are different points on the grid. The key steps to the SIMPLE algorithm are as follows:

1. Guess $u^{(k)}$, $v^{(k)}$, and $p^{(k)}$ where $k = 0$
2. Calculate Momentum Links and Sources (A_p, A_{nb}, S_x, S_y)
3. Solve X-Momentum - Inner Iterations
4. Solve Y-Momentum - Inner Iterations
5. Calculate Face Velocities using Pressure Weighted Interpolation Method (Rhie and Chow Interpolation)
6. Calculate Pressure Links and Mass Imbalance (Ap_p,Ap_{nb}, S_p)
7. Solve Pressure Correction - Inner Iterations
8. Correct velocity and pressure
9. Check convergence
    - If not converged, return to step 2 with $k = k+1$

The SIMPLE algorithm aims to iteratively solve for the coupled velocity-pressure field given some geometry over k iterations. This derivation of the simple algorithm will closely follow the lecture series by Professor Sandip Mazumder [2]. The algorithm begins by guessing the fields in question; $u^{(k)}$, $v^{(k)}$, and $p^{(k)}$. While the guesses for the $u^{(k)}$, $v^{(k)}$ fields are not particularly important, the guess for the pressure field is. The pressure field will always be initialized to 0 to avoid using the Rhie and Chow interpolation on the first iteration of the algorithm, where the pressure is weighted into calculating face velocities in step 5. This is done because solution convergence tends to be highly sensitive to the guessed pressure field, meaning a poor guess could lead to diverging results.

The next step in the process is calculating the x and y-momentum link coefficients as outlined in table II. The central coefficient a_p can nearly be calculated as the sum of the neighbor coefficients, but it's not. Instead the central coefficient inverts the sign on the advective terms, creating a completely different quantity;

$$a_p = D_w + D_e + D_s + D_n \tag{16}$$
$$+ \max(-F_w, 0) + \max(0, F_e) \tag{17}$$
$$+ \max(-F_s, 0) + \max(0, F_n) \tag{18}$$

In order to calculate the coefficients for the link matrix, we need to calculate the face velocities between cells. During the first ($k = 0$) iteration, this is done by merely guessing the face velocities. Since typically the initial guess for the velocity field is 0, the face velocities are also initialized to 0 when the momentum equation is solved for the first time.

At the boundaries of the grid the momentum equations changed slightly. First, we consider a no-slip condition where the velocity of the fluid in contact with the wall is equal to the velocity of the wall itself. This is implemented by setting face velocities against the wall to 0 on all 4 sides of the square domain;

$$\max(F_x, 0) = 0 \tag{19}$$

Additionally, a boundary layer approximation yields the result that the momentum equation which is solved normal to the surface of the boundary is orders of magnitude smaller than the momentum equation solved tangentially to the surface of a boundary. This result is derived in [2] and will not be repeated for the sake of concision. The derivation shows that the pressure gradient normal to the wall is negligible compared to the gradient along the surface. For an east or west wall, this means $\frac{\partial p}{\partial y} \gg \frac{\partial p}{\partial x}$. From this, we can make an assumption that the pressure 'outside' the boundaries which is necessary both for solving the momentum equation and for calculating the face velocities during the Rhie and Chow approximation is equal to the pressure at the cell just inside the boundary: $P_W = P_{j=0}$, $P_E = P_{j=nx-1}$.

For the northern wall where there is a boundary moving at a constant velocity, we use the equation for the shear force at a boundary from [1]:

$$F_s = -\tau_w A_{\text{Cell}} = -\mu \frac{u_P}{\Delta y_P} A_{\text{Cell}} \tag{20}$$

Where A_{Cell} is the area of the cell face (Δx), and Δy_P is the distance from the boundary which would be equal to $\Delta y/2$. This equation can be non-dimensionalized using the Reynolds number then added to the S_X source term:

$$F_s = -2 \frac{u_{\text{lid}}}{Re \Delta x} \Delta y \tag{21}$$

To solve the X and Y momentum equations and find the velocities of all cell centers, an iterative sparse solver is used. Refer to assignment 1 of the course for more details on the types solvers which can be applied to this problem. For this assignment, multiple solvers were tested for their ability to converge given different preconditioning/solution tolerances, and it was found that in general, bi-conjugate gradient stabilized from the SciPy linear algebra toolbox was the fastest.

After solving the momentum equations, the face velocities for each cell needs to be recalculated using the new cell centers. This is done using the Rhie and Chow interpolation [3].

A. The Rhie and Chow interpolation

Rhie and Chow proposed to a temporary velocity field at the cell faces which is simply a distance weighted interpolation between the cell faces (Central Difference). Consider 4 adjacent cells spanning 1 dimension for simplicity; east-east, east, P (central cell), and west.

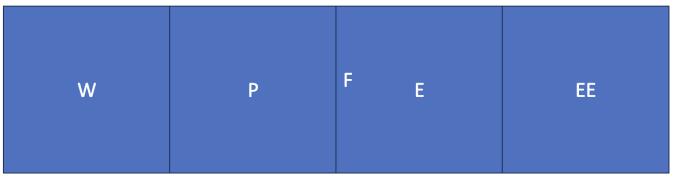

Figure 1: Four adjacent finite volume cells from left to right: West, P, East, East-East. Note the face in the center of the diagram labeled F, which is the where the velocity we're solving for is located.

Now consider we would like to calculate the face F between cells east and west. To do this, the distance weighted interpolation between cells west and east would be;

$$U_F = U_P \cdot \frac{l_P}{l_P + l_E} + U_E \cdot \frac{l_E}{l_P + l_E} \tag{22}$$
$$= U_P \cdot l_x + (1 - l_x) \cdot U_E. \tag{23}$$

Where l_P and l_E are the distances between the west and east node centers respectively to the face F, and l_x is the sum between them to simplify the equation. Rhie and Chow proposed some 'correction' term which could be added to this equation to solve for the actual face velocity, U_f;

$$U_f = \overline{U_F} + correction \tag{24}$$

Where $\overline{U_F}$ is the original distance weighted interpolation solved for in equation (23). To solve for this correction, we

can imagine a staggered grid cell around the face F, and can perform a pressure-inclusive momentum balance around it using the values at the cell centers P and E. We start by rewriting the momentum balance using our coefficients to solve for the value U_p:

$$a_p U_p + \sum a_N U_N = -\left.\frac{\partial p}{\partial x}\right|_P V_P \tag{25}$$

$$U_p = -\frac{1}{a_p}\sum a_N U_N - \left.\frac{V_P}{a_p}\frac{\partial p}{\partial x}\right|_P \tag{26}$$

Which uses the notation derived by ***, where V_p is the volume of the cell, and a_N, u_N are the coefficients and velocities at the neighboring cells. We can simplify this equation by introducing the variables;

$$d_p = \frac{V_P}{a_p} \tag{27}$$

$$\bar{U}_p = -\frac{1}{a_p}\sum a_N U_N \tag{28}$$

Which simplifies to the momentum balance to :

$$U_p = \tilde{U}_p - d_p \left.\frac{\partial p}{\partial x}\right|_P \tag{29}$$

This equation is neatly describing the cell centered velocity at point P in terms of the coefficients and velocities of it's neighbors, as well as the pressure gradient evaluated across the cell. This equation can be translated over to also calculate velocity at cell center E by simply changing the indices:

$$U_E = \tilde{U}_E - d_E \left.\frac{\partial p}{\partial x}\right|_E \tag{30}$$

Now let's consider the equations we've written, which solve for the velocities at the cell centers using the face velocities. We would like to apply this method to actually solve for the velocity at the face F, so, while imagining a staggered cell around the face F with boundaries at points P and E, we can say:

$$U_f = \tilde{U}_f - d_f \left.\frac{\partial p}{\partial x}\right|_f \tag{31}$$

In order to combine these equations, specifically equations (29), (30), and (31), and to solve for the correction term, we can write $\tilde{U_f}$ as a linear interpolation of the cell center velocity components which are also neighbor dependant:

$$\tilde{U}_f = l_x \tilde{U}_p + (1 - l_x)\tilde{U}_E \tag{32}$$

If we rearrange equations (30) and (29) to solve for the $\tilde{U}$ terms, we can then substitute them into equation (32):

$$\tilde{U}_f = l_x \left[U_p + d_p \left.\frac{\partial p}{\partial x_j}\right|_P\right] + (1 - l_x)\left[U_E + d_E \left.\frac{\partial p}{\partial x_j}\right|_E\right] \tag{33}$$

Now we have an equation for $\tilde{U_f}$ which can be substituted back into equation (31) to get the final equation for the face velocities. Before we do this, first we simplify equation (33) by grouping like terms and recognizing that the distance weighted interpolation of velocity exists in this equation:

$$\tilde{U}_f = \overline{U}_f + \left[l_x d_p \left.\frac{\partial p}{\partial x_j}\right|_P\right] + \left[(1 - l_x) d_E \left.\frac{\partial p}{\partial x_j}\right|_E\right] \tag{34}$$

$$= \overline{U}_f + d\overline{\left.\frac{\partial p}{\partial x_j}\right|_f} \tag{35}$$

Where d still denotes the volume of a finite cell over the central coefficient a_p, but it is now a linear interpolation between the 2 adjacent cells A_p and A_E. The same can be said about the pressure gradient, which is now calculated as a linear interpolation between the pressure gradients at cells P and E. If we substitute equation (35) back into equation (31), we yield an equation for the face velocity at face F in terms of the linearly interpolated velocities, along with a correction term utilizing the pressure gradient:

$$U_f = \overline{U}_f - \left(d_f \left.\frac{\partial p}{\partial x_j}\right|_f - d\overline{\left.\frac{\partial p}{\partial x_j}\right|_f}\right) \tag{36}$$

Now to discretize this equation and specify it to our coding problem, we will assume that all node points on the mesh are evenly spaced, with δx and δy representing the x and y spacing between successive nodes respectively, as well as the surface area of the node faces since the grid is 2 dimensional. We will also use the letter k to denote the outer iteration number, where at iteration 0, only a distance weighted interpolation for $(\overline{U_f})$ is used.

$$\begin{aligned}\tilde{u}_f = &\frac{1}{2}(\tilde{u}_p + \tilde{u}_E) + \frac{1}{4}\left(\frac{p_E^{(k)} - p_W^{(k)}}{A_p|_p}\right)\Delta y \\ &+ \frac{1}{4}\left(\frac{p_{EE}^{(k)} - p_p^{(k)}}{A_p|_E}\right)\Delta y \\ &- \left[\frac{1}{A_p|_p} + \frac{1}{A_p|_E}\right]\left(\frac{p_E^{(k)} - p_p^{(k)}}{2}\right)\Delta y\end{aligned} \tag{37}$$

In this equation, the values $A_p|_x$ are the central coefficients calculated from the momentum equation, which can be evaluated at node x. For a more in depth derivation of the Rhie and Chow interpolation, refer to [3] or [4].

B. Velocity and Pressure Correction

Once the face velocities have been corrected using the Rhie and Chow interpolation (or distance weighted interpolation for the first iteration), we need correct velocities

and the actual Pressure field. To do this I'll introduce the notation used in [2] where we have 2 equations for cell centered velocities; one for the current iteration k and another for the next iteration $k+1$:

$$\hat{u}_p = -\frac{1}{A_p|_p}\sum_{nb} A_{nb}\hat{u}_{nb}|_p + \frac{(p_W^{(k)} - p_E^{(k)})}{2A_p|_p}\Delta y \tag{38}$$

$$\hat{\hat{u}}_p = -\frac{1}{A_p|_p}\sum_{nb} A_{nb}\hat{\hat{u}}_{nb}|_p + \frac{(p_W^{(k+1)} - p_E^{(k+1)})}{2A_p|_p}\Delta y \tag{39}$$

In this case, the corrected velocities for the next iteration is what we want to determine, and that correction can be defined by subtracting the 2 equations:

$$u'_p = \hat{u}_p - \hat{\hat{u}}_p = -\frac{1}{A_p|_p}\sum_{nb} A_{nb}u'_{nb}|_p + \frac{(p'_W - p'_E)}{2A_p|_p}\Delta y \tag{40}$$

Where u'_p is the velocity correction which needs to be applied to $\hat{u}_p$ to correct the equation, and p' are the pressure corrections which we need to solve for. We can actually **boldly drop** the first term in equation (40) since ideally the converged solution will have u'_{nb} converge to 0. Therefore:

$$u'_p = \frac{(p'_W - p'_E)}{2A_p|_p}\Delta y \tag{41}$$

The same derivation is mirrored in the vertical direction of the grid:

$$v'_p = \frac{(p'_S - p'_N)}{2A_o|_o}\Delta x \tag{42}$$

We can follow the exact same process to correct cell face velocities by re-writing equation (38) as it's k and $k+1$ iterations, subtracting the 2, and then dropping the neighbouring u' terms:

$$u'_e = \left[\frac{1}{A_p|_E} + \frac{1}{A_p|_p}\right]\frac{(p'_p - p'_E)}{2}\Delta y \tag{43}$$

Now for the final step of SIMPLE, we need to derive the pressure corrections p' in order to not only correct the pressure distribution, but also for the correction of the cell-center and face velocities as seen in equations (41) through (43). To do this, we check the mass imbalance using the continuity equation using the new velocity terms:

$$(\hat{\hat{u}}_e - \hat{\hat{u}}_w)\Delta y + (\hat{\hat{v}}_n - \hat{\hat{v}}_s)\Delta x = 0 \tag{44}$$

We may then decompose the velocities and separate out the correction terms to introduce the net mass imbalance in the equation:

$$\begin{aligned}(u'_e - u'_w)\Delta y + (v'_n - v'_s)\Delta x = \\ -\left[(\hat{u}_e - \hat{u}_w)\Delta y + (\hat{v}_n - \hat{v}_s)\Delta x\right] = -\dot{m}_{\text{imbalance}}\end{aligned} \tag{45}$$

From this, we can then substitute in the terms derived in equation (43), using all the face velocities:

$$\left[\frac{1}{A_p|_E} + \frac{1}{A_p|_p}\right]\frac{(p'_p - p'_E)}{2}(\Delta y)^2 - \tag{46}$$

$$\left[\frac{1}{A_p|_W} + \frac{1}{A_p|_p}\right]\frac{(p'_W - p'_p)}{2}(\Delta y)^2 + \tag{47}$$

$$\left[\frac{1}{A_p|_N} + \frac{1}{A_p|_p}\right]\frac{(p'_p - p'_N)}{2}(\Delta x)^2 - \tag{48}$$

$$\left[\frac{1}{A_p|_S} + \frac{1}{A_p|_p}\right]\frac{(p'_s - p'_p)}{2}(\Delta x)^2 = -\dot{m}_{\text{imbalance}}$$

Equation (49) can be rewritten in terms of the components to solve for (p') and separated coefficients:

$$\begin{aligned}A_p^p p'_p + A_E^p p'_E + A_W^p p'_W + A_N^p p'_N + A_S^p p'_S = \\ A_p^p p'_p + \sum_{nb} A_{nb}^p p'_{nb} = S_p^p\end{aligned} \tag{49}$$

With coefficients:

$$A_E^p = -\frac{(\Delta y)^2}{2}\left[\frac{1}{A_{p|E}} + \frac{1}{A_{p|p}}\right], \tag{50a}$$

$$A_W^p = -\frac{(\Delta y)^2}{2}\left[\frac{1}{A_{p|W}} + \frac{1}{A_{p|p}}\right], \tag{50b}$$

$$A_N^p = -\frac{(\Delta x)^2}{2}\left[\frac{1}{A_{p|N}} + \frac{1}{A_{p|p}}\right], \tag{50c}$$

$$A_S^p = -\frac{(\Delta x)^2}{2}\left[\frac{1}{A_{p|S}} + \frac{1}{A_{p|p}}\right], \tag{50d}$$

$$A_p^p = -\left(A_E^p + A_W^p + A_N^p + A_S^p\right) \tag{50e}$$

This can be solved using the same sparse matrix method as the momentum equations, yielding a grid of pressure correction terms p' which can then be used in equations (41) through (43) to compute the velocity corrections u' for both faces and cell centers. These correction terms can then be used to correct the cell center and face velocities, along with the pressure grid according to an element-wise addition:

$$p_p^{(k+1)} = p_p^{(k)} + \omega_p p'_p, \tag{51a}$$

$$u_p^{(k+1)} = \hat{u}_p + \omega_u u'_p, \tag{51b}$$

$$v_p^{(k+1)} = \hat{v}_p + \omega_v v'_p. \tag{51c}$$

The terms ω_p, ω_u, and ω_v are called under-relaxation factors, and typically range between the numbers 0 and 1. The purpose of the under-relaxation factor is to only add a partial component of the correction term for each u, v, or P at a time. This is because it's easy for the SIMPLE algorithm to diverge given corrections which are too large.

Once the correction to the relevant fields have been made, we can check whether or not the values have converged by calculating the L2 norm between the $k+1$ and the k iteration, and defining convergence as being below a specific threshold. If not all the fields have converged,

then we may use them at the initial inputs and start a new iteration of SIMPLE by recalculating the momentum link coefficients and repeating the process until convergence.

III. RESULTS AND DISCUSSION

The lid Driven Cavity Problem was solved using a 257x257 point mesh for a cavity with side lengths $l_x, l_y = 1$ and a Reynolds number of $Re = 100$. A Bi-conjugate Gradient Descent sparse solver retrieved from the SciPy Linear Algebra python library was used for the inner iterations when solving the momentum equations and the pressure correction terms. A partial LU factorization preconditioner was used for all sparse solvers, significantly speeding up the convergence time. For each of the inner iterations, a solver tolerance of 10^{-7} and pre-conditioner tolerance of 10^{-5} were implemented. Under-relaxation factors $alpha_{uv}$ and $alpha_p$ were set to 0.3 and 0.001 respectively after experimenting with different factors on a lower resolution grid. A tolerance of 10^{-4} was applied to the convergence condition of all 3 scalar fields u, v, and P. The maximum number of iterations to reach convergence was set to 35000. It took 13 hours and 43 minutes to complete computation.

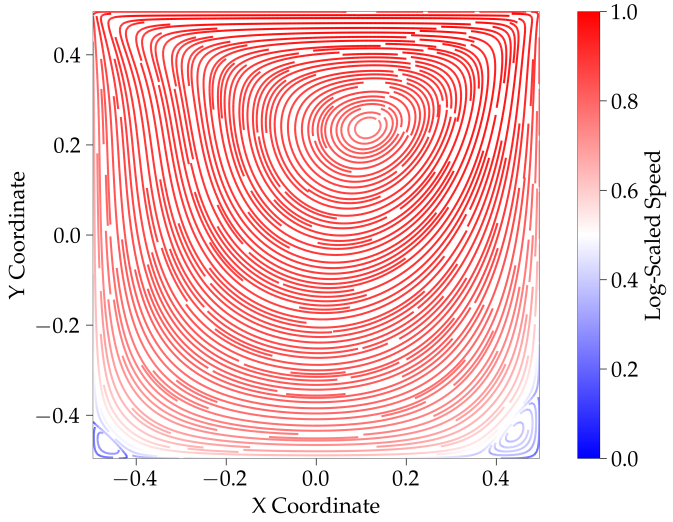

Figure 2: Streamline plot illustrating fluid flow at steady state. Velocity values range from scales 1 at the top of the grid to $1e^{-7}$ near the bottom corners, so to illustrate the change in velocities a log scaled speed colorization is used.

In figure 2 the arrows of the stream plot have been removed to reduce clutter. We notice several phenomena occurring in this stream plot. The first is the major negative-vorticity (clockwise) rotation of the fluid around an epicenter near the upper-East quadrant of the grid. The fluid above this epicenter tends to travel rightward along the North boundary with the lid, and values at the north wall approach very close to the lid velocity, which is equal to 1. On the east side of the grid, the flow which now comes in contact with the east wall is forced downwards, which is corroborated by the V or y-directional velocity plot in 5 showing the negative vertical velocity component, which reaches it's maximum at the top-right corner of the domain. As the flow reaches the bottom right corner of the domain, it splits, with most of the flow continuing by travelling west-ward around the epicenter, while some of the flow gets caught in a small eddy current in the corner.

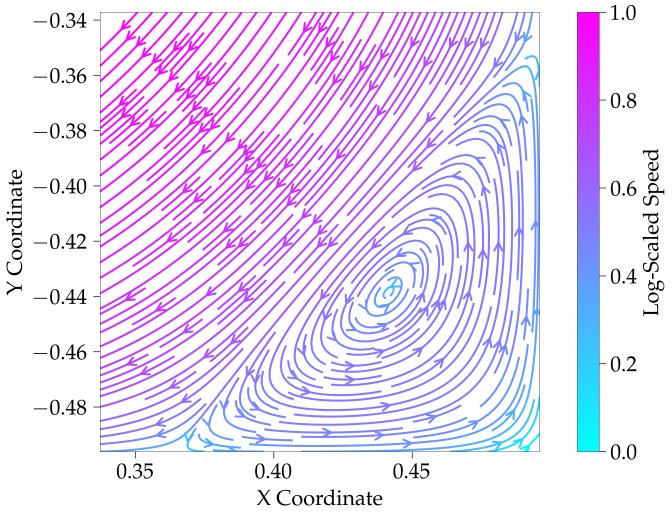

Figure 3: Streamline plot illustrating the Eddy vortex in the South-East corner of the domain. Velocity values have been log-scaled to max out at 1. Window size is $\frac{1}{36}$ of the domain by area with 47 squared cells.

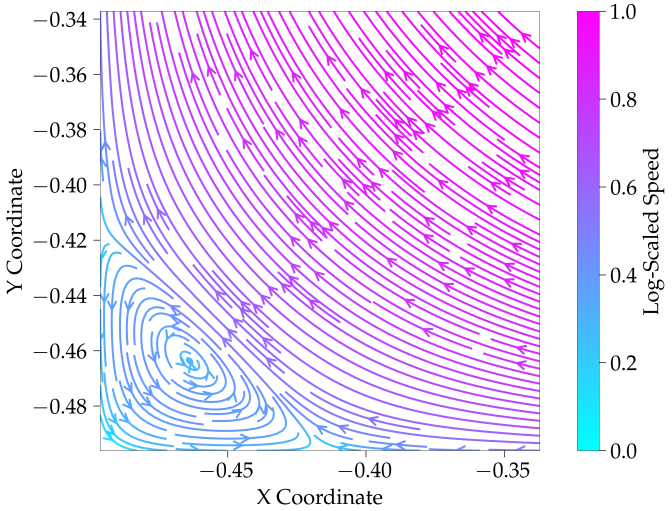

Figure 4: South-West eddy vortex streamlines. Window size is $\frac{1}{36}$ of the domain by area with 47 squared cells.

As can be seen in figures 3, after the flow separates in the south-east corner it begins turning counter clockwise through the corner, with positive vorticity acting counter to the major stream. That said the velocity values in this corner, as well as in the south-east corner seen figure 4, are nearly 0, with the speeds in the figures log-scaled for visualization purposes. The presence of these 2 eddy's in either corner of the domain show great promise that the upwind scheme has been correctly implemented, as use of a simple

distance weighted velocity interpolation would diffuse the eddies out of the simulation.

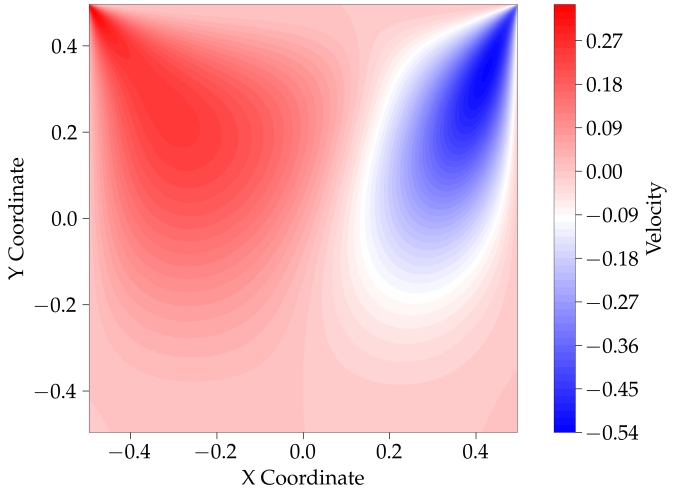

Figure 5: Y-directional velocity contour plot with 80 levels (v-profile). Positive values denotes an upwards or northern flow of the fluid, while negative represents downwards.

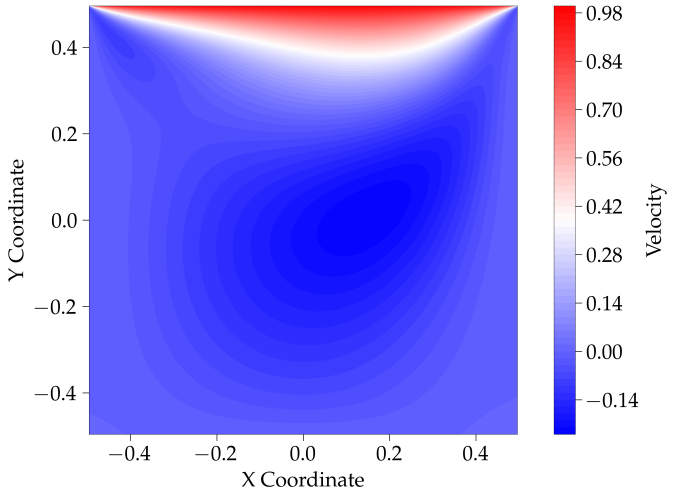

Figure 6: X-directional velocity contour with 80 levels (u-profile). Positive values denote a rightwards or eastern flow of the fluid while negative, values represent a leftward flow.

After being pushed across the bottom of the domain, the fluid is pulled and accelerated upwards, as can be once in figure 5 where the largest y-directional velocities are focused near the top left corner.

Examining figure 6, we also notice another phenomenon, which is the building of a laminar boundary layer along the north wall. The top of the figure shows a bright red region denoting the large x-directional velocities, which looks like a layer of fluid who's thickness is increasing linearly as it crosses the top of the domain. This increase must of course stop as the fluid crashes back into the eastern wall, but it's interesting to see how the boundary layer, with an initially large velocity gradient at the north-west corner of the domain (which can be observed by the highly discrete color change) begins to smooth out as the boundary collects more fluid moving eastward.

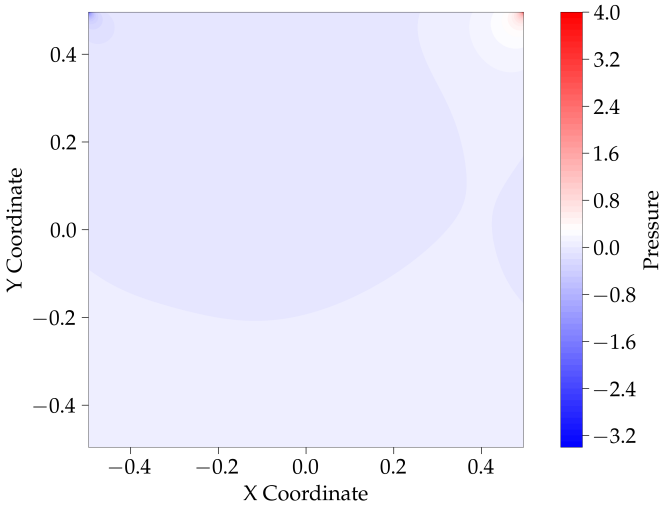

Figure 7: Pressure contour plot. Absolute values are somewhat negligible given the lack of reference or gauge pressure.

The velocity profiles can be largely explained using the pressure contour plot seen in figure 7. There are no specific boundary conditions given for the pressure in this problem, meaning that as long as a gradient is solved for, the absolute values of the pressure distribution can change, and only their relative difference will truly matter to the resulting velocity profiles. We can see that the pressure in the domain is mostly near 0, except for a large concentration of positive and negative pressure at the upper east and west corner respectively. The north-east corner with it's high pressure means that the fluid is actively being driven into that area, which is corroborated by figure 6, which in turn creates the force that drives the fluid downwards seen as the large blue profile in figure 5.

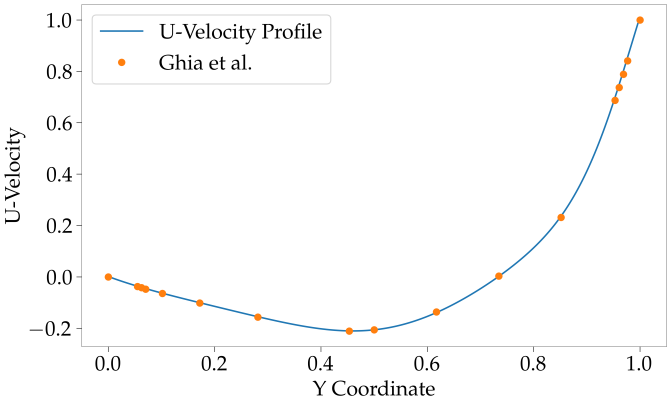

Figure 8: X-directional velocity profile (U-Velocity) taken as the cross section from a vertical line which separates the center of the domain. Orange circles represent values obtained from [5] for a 129 point squared mesh.

On the opposite end, the large negative pressure in the north-west corner indicates that the fluid is being actively

pulled away from that point in the domain, which makes sense given the u velocity profile. In order to satisfy the continuity equation, fluid is then pulled from from below into the low pressure zone, creating the high upwards velocity seen as the large red profile in figure 5 and completing the vortex.

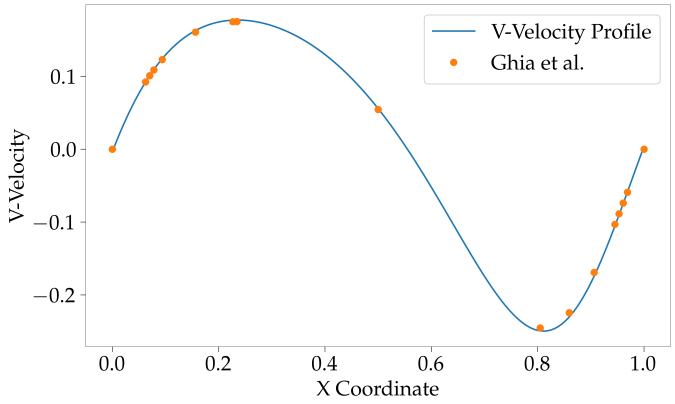

Figure 9: Y-directional velocity profile (V-Velocity) taken as the cross section from a horizontal line which separates the center of the domain.

To validate the results and quantitatively test their reproducibility against the results of other who've used the same algorithm, the velocity profile of the domain is taken for 2 cross-sections; A horizontal line which cuts through the center of the grid at $y = 0$, and a vertical line which cuts through the center at $x = 0$. Along the vertical line, the U-velocities or x-directional velocities are measure, while the V-velocities or y-directional velocities are measured across the horizontal line. This mimics the research performed by Ghia et. al. in [5]. The points obtained from Ghia et. al. are plotted as orange circles, while the profiles from this paper are plotted as the continuous blue lines. We can see from the graphs that the profiles are nearly identical, with some slightly off values in the figure 9. Of course, some dissimilarity is to be expected given that the same convergence tolerances were not used, nor were the same grid sizes or linear solver as [5], which also uses a 129 point mesh rather than a 257 point mesh..

IV. CONCLUSIONS

The culminating assignment for MIE 1210: Computational Fluid Mechanics and Heat Transfer was indeed a difficult one. Small issues with sign convention, indexing, or even more annoyingly, implementation of under-relaxation factors could lead to code-breaking bugs. In the ends, however, the results achieved in this final assignment were highly satisfying. The cross-referenced results between this assignment and that of Ghia et al. in [5] shows unequivocally that the SIMPLE algorithm and the finite volume method was correctly implemented in a 2D domain. There are further applications to the code however. Having correctly specified the number of x and y dimensional points, as-well as their spacing, throughout the code, my script is easily capable of solving the lid driven cavity for non-square grids:

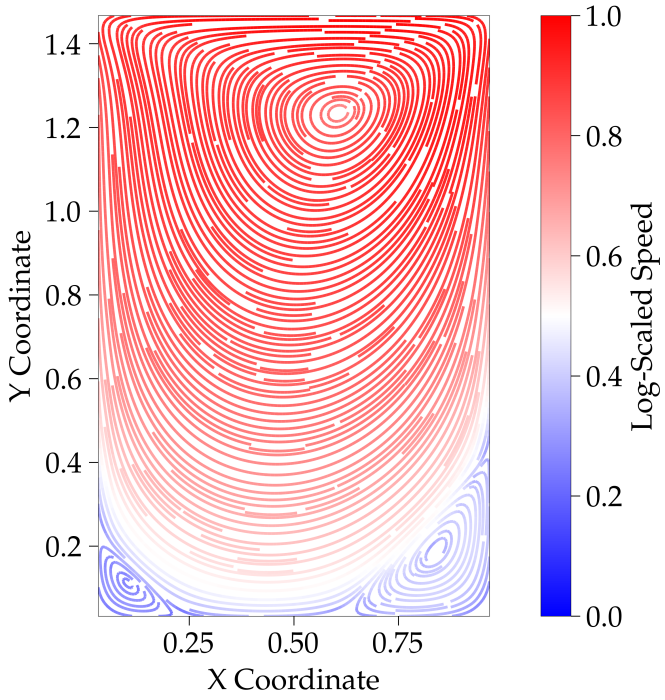

Figure 10: Streamlines for a rectangular grid of dimensions $l_x = 1$, $l_y = 1.5$ with node size $n_x = 30$, $n_y = 45$.

The simulation in figure 10 shows the steady state solution to a 35x45 rectangular grid with domain lengths $l_x = 1$, $l_y = 1.5$. Examining this interesting extension we can note that first; my code works with non-equal cell spacing or area, which is a useful method of checking for symmetry bugs and troubleshooting. Second, we notice that the general shape of the solution is highly similar to that of the square cavity, but now the presence of the eddy currents penetrates significantly deeper into the domain. I speculate that this is due to the longer distance which the flow has to separate from the wall, and the comparatively lower vertical pressure gradient which doesn't force as much of the fluid deep into the corners of the domain. For the sake of concision I won't bore the reader with the additional velocity and pressure contour plots, they look very similar in their shape the square domain, except the u-velocity's boundary layer envelope is shortened, and the vertical velocity's high density regions have been extended to greater lengths down the domain.

The problem with the code is that the boundary conditions have been hard-coded into the domain. Additionally the code is slow, with a convergence time which scales quadratically on the order of **. In order to improve convergence speed I attempted to implement a variable under-relaxation factor: a relaxation factor which could increase or decrease based off of feedback from the computed L2 norms every several iterations. The idea is derived from an adaptive learning rate in machine learning, where if certain values, or adjustments are repeated in the same way multiple times in a row, the relaxation factor could increase in order to speed up the correction steps. On the other hand,

if using a relaxation factor which is too high, instability can occur. If the recent L2 norms began oscillating or stagnating, the relaxation for that variable would decrease by a specified factor in order to reduce the correction step size.

This approach was introduced very late, after all the results for this assignment were completed, and so it didn't offer much use in terms of convergence speed up, but it's an interesting concept which I'd like to test further.

The other thing I would change about the code architecture is how the boundaries are hard-coded into the python functions. This creates no generalizability in the shape of the domain (for instance using stepped boundaries or inlets/outlets) apart from varying the rectangular dimensions. In the future, I'd like to organize how the boundary conditions are set based off a dictionary of potential functions which could be applied to boundary cells.

V. APPENDIX - CONVERGENCE ANALYSIS

To satisfy the fact that our grid has converged, I'm going to compare the u and v velocity profiles across the center lines for multiple grid sizes. Specifically I'm going to show the results for a 40 point mesh and 80 point mesh grid, then compare them to figures 8 and 9 to prove that the 257^2 grid is not significantly more resolved, and has essentially converged.

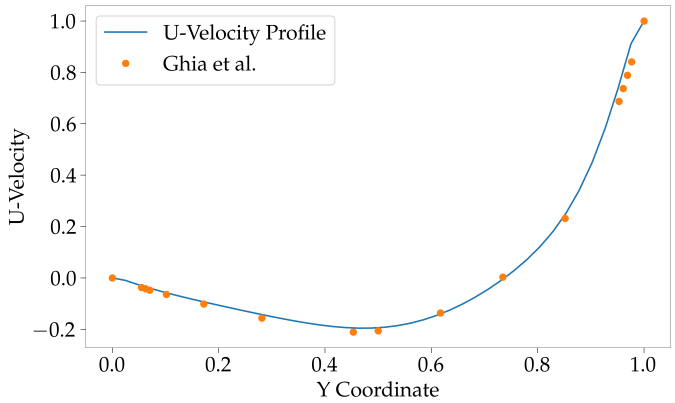

Figure 11: X-directional velocity profile (U-Velocity) taken as the cross section from a vertical line which separates the center of the domain for an 80^2 point mesh.

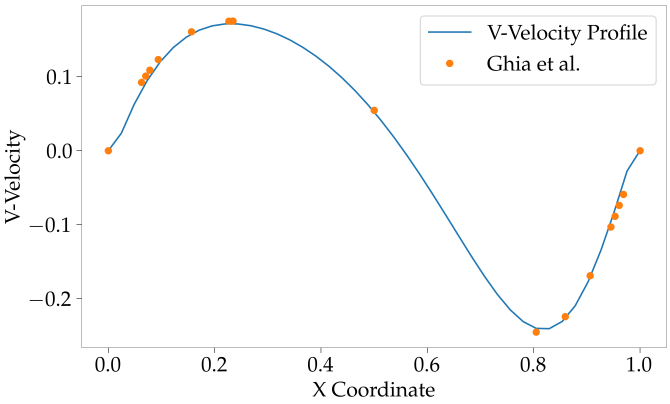

Figure 12: Y-directional velocity profile (V-Velocity) taken as the cross section from a horizontal line which separates the center of the domain for an 40^2 point mesh.

As we can see, the 40 squared point mesh actually produces some decent agreement with the results from Ghia et al. in [5], that said, there are definitive issues with resolution and the values do not match exactly. Moving on to the 80^2 mesh, the results are significantly improved, with the blue lines passing essentially through the exact center of the orange circles, especially in 14. These graphs look nearly identical to that of 8 and 9, demonstrating that a 257 point mesh is likely unnecessary for a Reynolds number this low. One might imagine, however, that the values in the corners, where the eddies are being generated is significantly different in these smaller meshes.

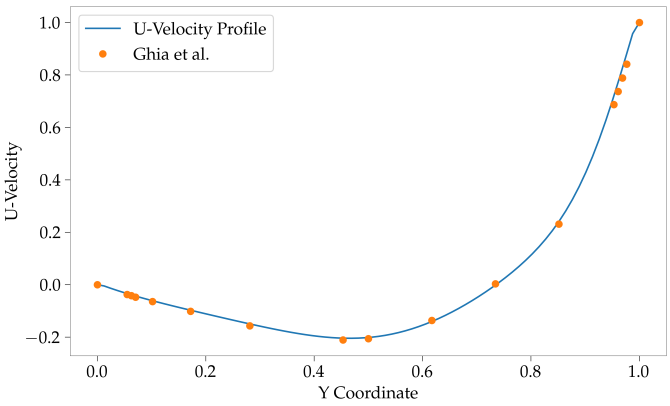

Figure 13: X-directional velocity profile (U-Velocity) taken as the cross section from a vertical line which separates the center of the domain for an 80^2 point mesh.

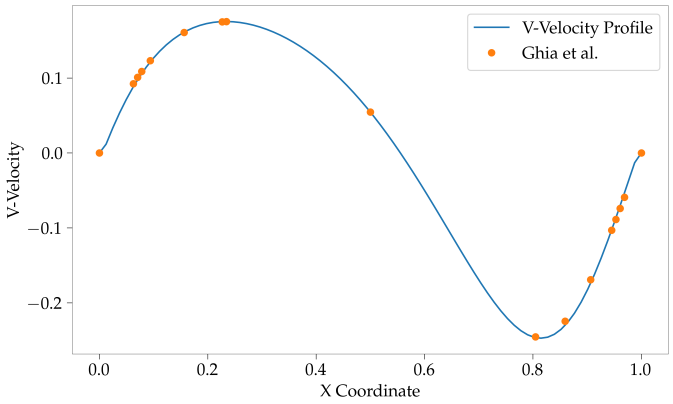

Figure 14: Y-directional velocity profile (V-Velocity) taken as the cross section from a horizontal line which separates the center of the domain for an 80^2 point mesh.

In the reference paper [5], only a 129^2 mesh grid was used to simulate the lid-driven cavity problem for Reynolds number of 100. One must imagine that given their results, as well as the distinct similarities in the center line velocity profiles exhibited in figures 8, 9, 11, 12, 13, and 14, that a 257 squared grid is fully converged to a complete solution.

[1] H. K. Versteeg and W. Malalasekera, *An Introduction to Computational Fluid Dynamics: THE FINITE VOLUME METHOD*, 2nd ed. Pearson Education Limited, 2007.

[2] D. S. Mazumder. Simple algorithm on co-located mesh. Youtube. [Online]. Available: https://www.youtube.com/watch?v=yWUc2D_WTMY&list=PLVuuXJfoPgT4gJcBAAFPW7uMwjFKB9aqT&index=8

[3] C. M. Rhie and W. L. CHOW, "Numerical study of the turbulent flow past an airfoil with trailing edge separation," vol. 21, 1983.

[4] F. M. 101. [cfd] rhie chow interpolation (part 3): Deriving the correction! Youtube. [Online]. Available: https://www.youtube.com/watch?v=PmEUiUB8ETk&ab_channel=FluidMechanics101

[5] U. Ghia, K. Ghia, and C. Shin, "High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method," *Journal of Computational Physics*, vol. 48, no. 3, pp. 387–411, 1982. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0021999182900584