FINAL REPORT

A PREPRINT

Francisco E. Spaulding-Astudillo
Department of Earth, Planetary, and Space Sciences
University of California, Los Angeles
Los Angeles, CA 90024
fspauldinga@ucla.edu

November 28, 2022

1 Description of the overall approach to numerical solve for the incompressible flow.

I use the projection method to numerically solve for the incompressible flow.

$$Ru^F = Su^n + \frac{\Delta t}{2}(3A^n - A^{n-1}) + \frac{\nu \Delta t}{2}(bc_L^{n+1} + bc_L^n) \tag{1}$$

$$DR^{-1}GP^{n+1} = \frac{1}{\Delta t}Du^F + \frac{1}{\Delta t}bc_D^{n+1} \tag{2}$$

$$u^{n+1} = u^F - \Delta t R^{-1} G P^{n+1} \tag{3}$$

In Equation (1), an intermediate velocity is first computed at $n+1$ from the momentum equation. In Equation (2) (i.e. pressure-poisson equation), incompressability ($\nabla \cdot u^F = 0$) at $n+1$ is enforced using the intermediate velocity. By definition, the LHS is equal to the error that arises from taking the divergence numerically. The following step then uses this error to correct the prediction. Therefore, in Equation (3), the velocity at $n+1$ is computed via a correction of u_F by the pressure field that satisfies incompressability. Here, it is worth noting that the operator, A, is the negative of the advective term: $A = -\nabla \cdot (\vec{u}\vec{u})$. Implicit methods are the best for linear terms from a numerical stability standpoint, so I use the Crank-Nicholson method for the viscous diffusion term. Similarly, explicit methods are the best for non-linear terms, so I use 2^{nd} order Adams-Bashforth for the advection term. bc_D^{n+1} is known because no-flow-through is enforced at each wall at every time step (i.e. $u = 0$ at left and right walls and $v = 0$ at bottom and top walls). In practice, I use a conjugate gradient solver to find the solutions to Equations (1) and (2): u^F and P^{n+1}, respectively.

$$R = I - \frac{\Delta t \nu}{2}L \tag{4}$$

$$R^{-1} = (I - \frac{\Delta t \nu}{2}L)^{-1} = I + \frac{\Delta t \nu}{2}L + (\frac{\Delta t \nu}{2}L)^2 + ... \tag{5}$$

$$S = I + \frac{\Delta t \nu}{2}L \tag{6}$$

To ensure at least second order accuracy in time, I retain two or more terms in R^{-1}. The Laplacian operator, L, is absorbed into the R, R^{-1}, and S operators, which act on matrices of size velocity.

2 Verification of each operator used in the code.

2.0.1 Divergence ($D : n_q \rightarrow n_p$)

The divergence operator takes in an array of size velocity and outputs an array of size pressure. The walls (4), corners(4), and interior(1) are all treated separately. This operator is not evaluated at the bottom-left corner because the pressure value there is pinned. The walls have one boundary condition each, so it follows that the corners have two boundary

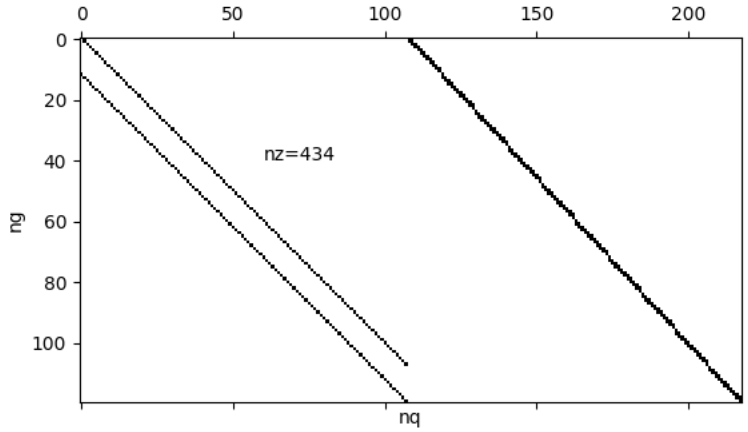


Figure 1: Sparsity pattern of divergence operator. Note that Lx=1, Ly=1.5, nx=10, ny=12.

conditions, one inherited from each adjacent wall. The boundary conditions are commented out and evaluated later in a separate boundary condition array of size velocity. In Figure 1, the sparsity pattern of the divergence operator is plotted. It agrees well with the result from lecture, which used the same grid specifications. To verify the divergence operator, I created a separate boundary condition array in which the commented-out parts from the divergence operator were evaluated.

$$\vec{u}(x, y) = sin(x)\hat{x} + sin(y)\hat{y} \tag{7}$$

Using the function definition above to compute the exact value of $\nabla \cdot \vec{u}(x, y)$ over the domain and the divergence operator in conjunction with the boundary condition operator to compute the analytical value over the domain, I determined the truncation error as a function of the domain size (and therefore the spatial step) and plotted the result in Figure 2. As expected, the 2nd-order accurate divergence operator has a slope of 2 on the log-log plot.

2.0.2 Gradient ($G : n_p \rightarrow n_q$)

The gradient operator takes in an array of size pressure and outputs an array of size velocity. The x-dir and y-dir are treated separately. There is one boundary condition in the bottom-left corner, where the pressure is pinned. The pinned pressure is commented out and evaluated later in a separate boundary condition array of size pressure. In Figure 3, the sparsity pattern of the gradient operator is plotted. It agrees well with the result from lecture, which used the same grid specifications. To verify the gradient operator, I created a separate boundary condition array in which the commented-out parts from the gradient operator were evaluated.

$$f(x, y) = sin(x) + sin(y) \tag{8}$$

Using the function definition above to compute the exact value of $\nabla f(x, y)$ over the domain and the gradient operator in conjunction with the boundary condition operator to compute the analytical value over the domain, I determined the truncation error as a function of the domain size (and therefore the spatial step) and plotted the result in Figure 4. As expected, the 2nd-order accurate gradient operator has a slope of 2 on the log-log plot.

2.0.3 Laplacian ($L : n_q \rightarrow n_q$)

The Laplacian operator takes in an array of size velocity and outputs an array of size velocity. The x-dir and y-dir are treated separately. For each direction, the walls (4), corners(4), and interior(1) are all treated differently. The walls have one boundary condition each, so it follows that the corners have two boundary conditions, one inherited from each adjacent wall. The boundary conditions are commented out and evaluated later in a separate boundary condition array of size velocity. In Figure 5, the sparsity pattern of the laplacian operator is plotted. It agrees well with the graphs obtained by other classmates, who used the same grid specifications upon request. To verify the laplacian operator, I created

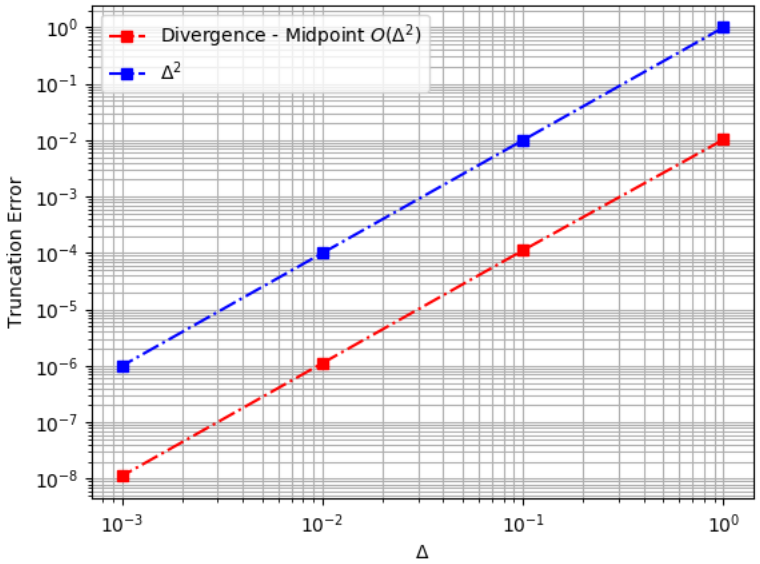


Figure 2: Truncation error of divergence operator. Here, nx=10 and ny=10.

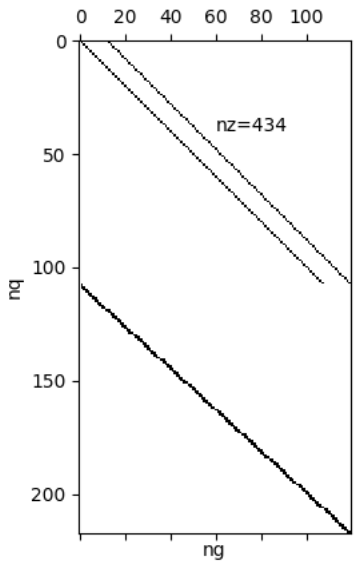


Figure 3: Sparsity pattern of gradient operator. Note that Lx=1, Ly=1.5, nx=10, ny=12.

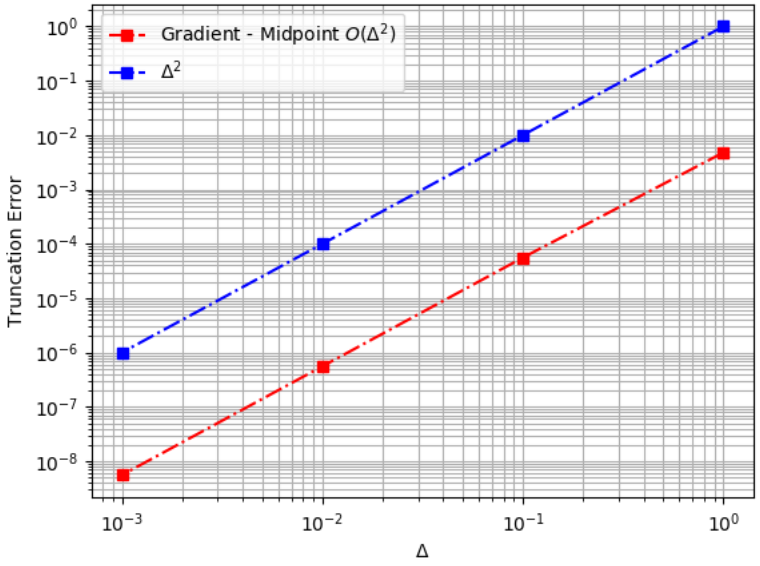


Figure 4: Truncation error of gradient operator. Here, nx=10 and ny=10.

a separate boundary condition array in which the commented-out parts from the divergence operator were evaluated. After a discussion with classmates, I decided to create an additional subroutine that returns the boundary condition value given 'u' or 'v' coordinate indices. This helped clean up my code and standardize the boundary condition look-up operation.

$$\vec{u}(x, y) = sin(x)\hat{x} + sin(y)\hat{y} \tag{9}$$

Using the function definition above to compute the exact value of $\nabla^2\vec{u}(x, y)$ over the domain and the laplacian operator in conjunction with the boundary condition operator to compute the analytical value over the domain, I determined the truncation error as a function of the domain size (and therefore the spatial step) and plotted the result in Figure 6. Unexpectedly, the laplacian operator has a slope of 3 on the log-log plot between $\Delta = [1, 0.1, 0.01]$, indicating 3rd-order accuracy. However, it does exhibit slope of 2 on the log-log plot between $\Delta = [0.01, 0.001]$.

2.0.4 Nonlinear advection term ($N : n_q \rightarrow n_p$)

The nonlinear operator takes in an array of size velocity and outputs an array of size velocity. The x-dir and y-dir are treated separately. For each direction, the walls (4), corners(4), and interior(1) are all treated differently. For the x-dir, there is 1 boundary condition at the left and right walls, 2 boundary conditions at the bottom and top walls, and so it follows that the corners have 3 boundary conditions, inherited from each adjacent wall. For the y-dir, there is 1 boundary condition at the bottom and top walls, 2 boundary conditions at the left and right walls, and so it follows that the corners have 3 boundary conditions, inherited from each adjacent wall. Each respective direction required 2nd-order midpoint interpolation to produce values for the nonlinear term. This interpolation equation is given below: it can be obtained by using a three-point stencil to Taylor expand about f_j, adding the $f_{j+1/2}$ and $f_{j-1/2}$ expressions, and solving for f_j.

$$f_j = \frac{f_{j+1/2} + f_{j-1/2}}{2} - \frac{\Delta^2}{8} f_j^{(2)} + O(\Delta^4) \tag{10}$$

Here, f_j is the interpolated value and $f_{j+1/2}$ and $f_{j-1/2}$ are part of the staggered velocity grid. Unlike the other three operators, the nonlinear operator does not have a separate boundary condition function. Instead, I implemented a boundary-condition look-up function into the operator that gives an exact value at the specified 'u' or 'v' coordinate. In Figure 7, the sparsity pattern of the nonlinear operator is plotted. It agrees well with the graphs obtained by other classmates, who used the same grid specifications upon request.

$$\vec{u}(x, y) = sin(x)\hat{x} + sin(y)\hat{y} \tag{11}$$

Using the function definition above to compute the exact value of $\nabla \cdot (\vec{u}\vec{u})$ over the domain and the nonlinear operator to compute the analytical value over the domain, I determined the truncation error as a function of the domain size (and

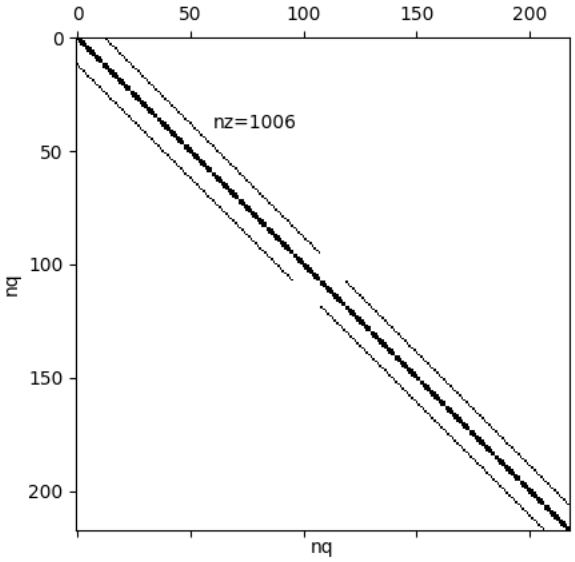


Figure 5: Sparsity pattern of Laplacian operator. Note that Lx=1, Ly=1.5, nx=10, ny=12.

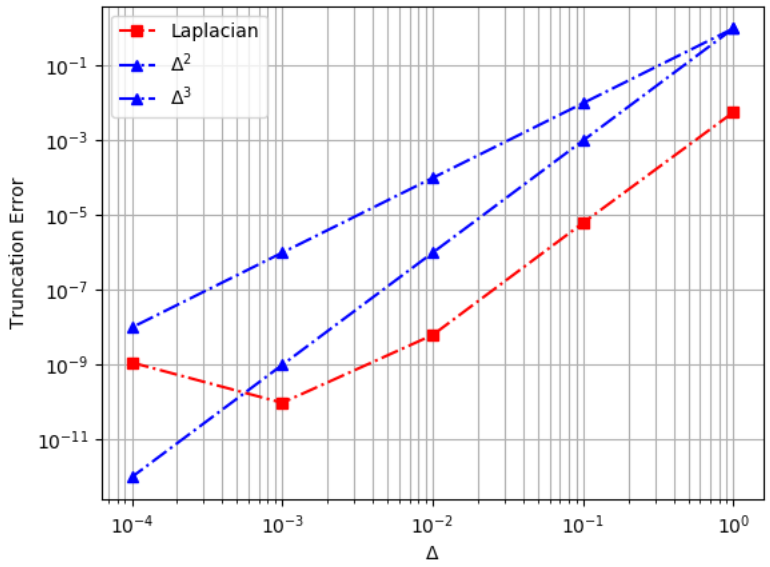


Figure 6: Truncation error of Laplacian operator. Here, nx=10 and ny=10.

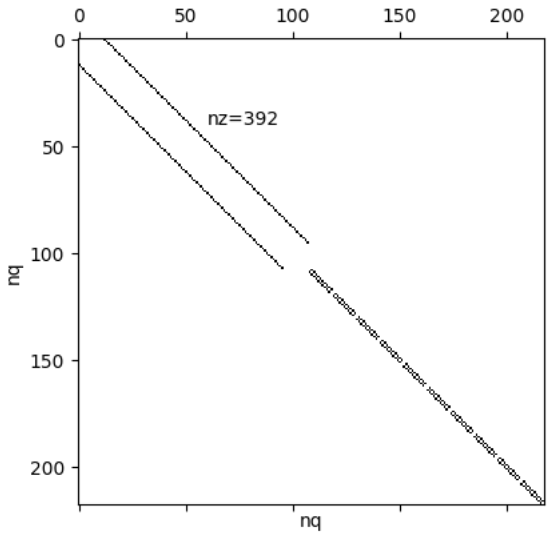


Figure 7: Sparsity pattern of nonlinear operator. Note that Lx=1, Ly=1.5, nx=10, ny=12.

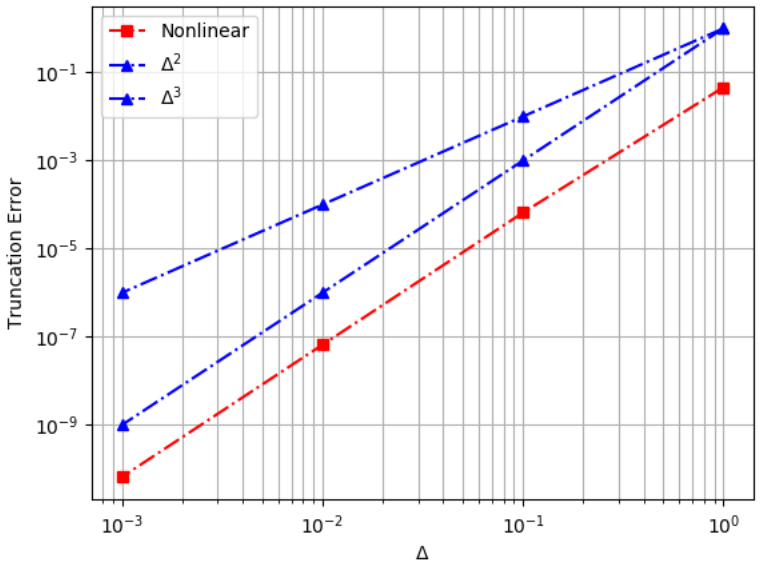


Figure 8: Truncation error of nonlinear operator. Here, nx=10 and ny=10.

therefore the spatial step) and plotted the result in Figure 8. Unexpectedly, the nonlinear operator has a slope of 3 on the log-log plot, indicating 3rd-order accuracy.

$$N_x = \sin(x)[2\cos(x) + \cos(y)] \tag{12}$$
$$N_y = \sin(y)[\cos(x) + 2\cos(y)] \tag{13}$$

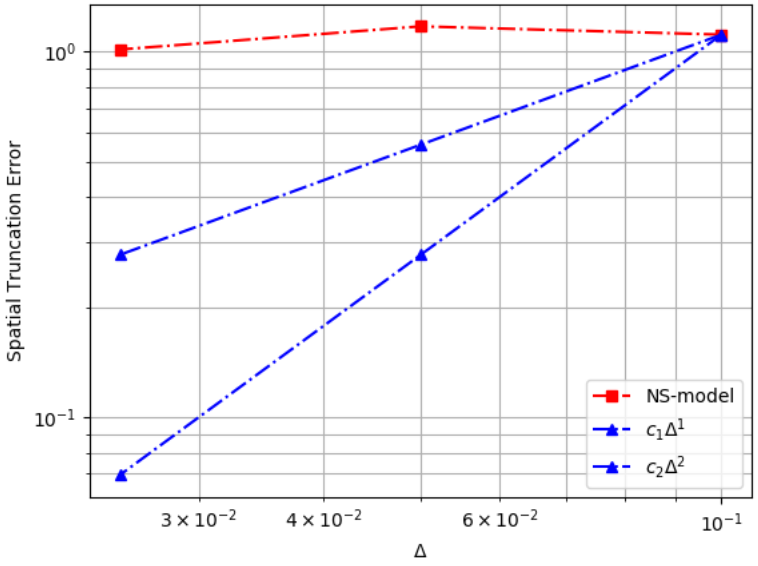
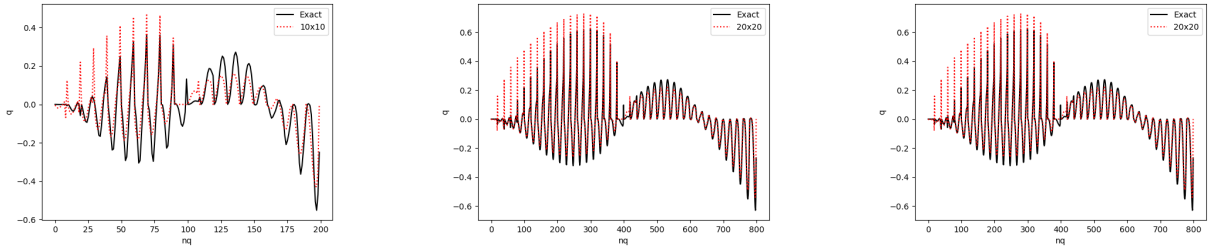


Figure 9: Shows spatial convergence of numerical solution [10x10,20x20,40x40] towards exact solution [80x80] at t=10s.



(a) 10x10 (b) 20x20 (c) 40x40

Figure 10: Plots $\vec{u}$ as an array of size n_q and compares the numerical (dotted red) and exact (black) solutions at t=10s.

3 Spatial and Temporal Convergence Check

3.1 Spatial Convergence Check

The overall process of carrying out the spatial convergence check is straightforward. I use a grid dimension of [Lx,Ly]=[1,1] and incrementally increase the resolution [nx,ny]. Using the same Δt for a given Re, I run the model to a specified t and compare the truncation error between the "exact" (highest resolution) and numerical (lower resolution) solutions. In doing so, I will be able to determine that the solution spatially converges at the right rate. For this test, I ran simulations at [20x20,40x40,80x80] to t=10s. I considered the 80x80 simulation to be the exact solution. To compute the truncation error for the 20x20 simulation, I had to create an exact solution with the same number of grid points as the 20x20 simulation. To do this, I selected every fourth grid point from the 80x80 simulation. Similarly, I selected every second grid point from the 80x80 simulation when computing the truncation error of the 40x40 simulation. The result of this analysis is shown in Figures 9 and 10. Figure 10 shows that the periodicity of the numerical solution is similar to that in the exact solution. However, Figure 9 confirms that the numerical solutions are not converging at the right rate. I have a few thoughts about why this might be the case. In computing an exact solution that can be compared against lower resolution experiments, I am selecting every nth grid point. This means that the some information from the higher-resolution simulations is lost. Perhaps an interpolation of the exact solution onto the lower resolution (nxn) grid would show the correct spatial convergence. Note that I ran my model only to t=10s before computing the truncation

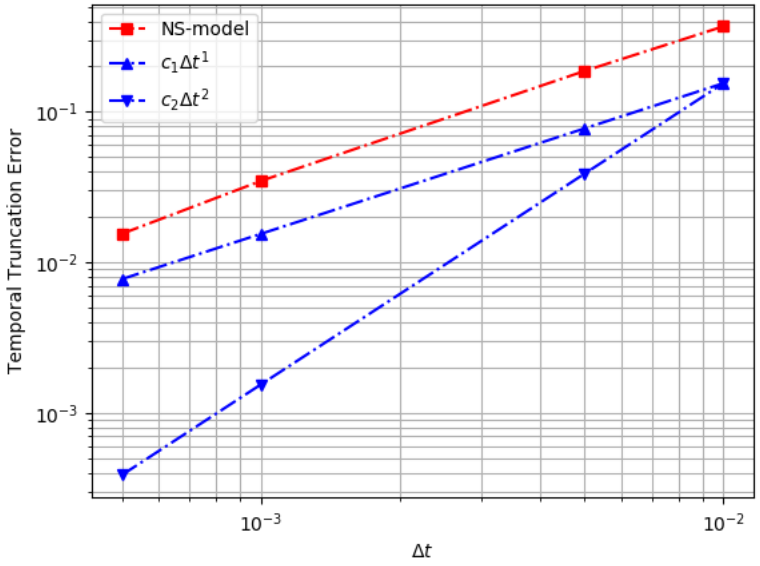


Figure 11: Shows temporal convergence of numerical solution dt=[0.0005,0.001,0.005,0.01] towards exact solution dt=[0.0001] at t=0.1s and Re=400.

error. This could be an issue. The similarity of my results to Ghia et al. is encouraging, so I think the primary problem is my method of computing the spatial convergence.

3.2 Temporal Convergence Check

To test for temporal convergence, I run my model for lid-driven cavity flow for different Δt and show that the model converges on the same solution as $\Delta t \to 0$. I run the model to a specified t and compare the truncation error between the "exact" (smallest Δt) and numerical (larger Δt) solutions. From lecture, I expect that the spatial error will be much larger than the temporal error. To counter this, I can increase the CFL condition so that I can isolate the temporal convergence.

For this test, I ran 80x80 simulations at dt=[0.0005,0.001,0.005,0.01] to t=0.1s at Re=400. I considered the $dt = 10^{-4}$ simulation to be the exact solution and computed the truncation error for each dt. The result of this analysis is shown in Figure 11. While I expected to find 2nd-order temporal convergence, Figure 11 shows that I obtained 1st-order temporal convergence.

4 Testing of the Conjugate Gradient Solver.

The goal of the conjugate gradient method is to solve the linear equation

$$Ax = b \tag{14}$$

If the matrix A is symmetric and positive-definite (i.e. all eigenvalues are positive), then the solution to Equation 14 is a minimum of $f(x) = \frac{1}{2}x^T Ax - b^T x + C$. [4]

$$f'(x) = Ax - b \tag{15}$$

The conjugate gradient method is therefore a process of searching for the x that minimizes the function $f(x)$, i.e. $f'(x) = 0$. The initial guess for x is our first search direction. The search process utilizes the concept of residuals, where the residuals have the property of being orthogonal to the previous search direction as well as previous residuals. [4] The residual is defined as:

$$r = Ax - b \tag{16}$$

In this case, I want the residual to be as close to zero as possible because then I have found the x at which $f'(x) = 0$, giving me our minimum f(x) and therefore the solution to $Ax = b$. That is where our prescribed error tolerance becomes important. I don't need r to exactly equal zero, just $r << 1$.

4.1 Test 1: Well-defined problem with known solution

I used psuedocode for the conjugate gradient solver from Appendix B2 of Shewchuk [4] to develop my code. Following a well-defined problem with a known, step-by-step solution [5] for debugging purposes, I tested my conjugate gradient solver.

$$Ax = \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \tag{17}$$

Using the suggested initial guess of

$$x_0 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \tag{18}$$

I expect the following solution.

$$x = \begin{bmatrix} \frac{1}{11} \\ \frac{7}{11} \end{bmatrix} \tag{19}$$

Using imax=5 and prescribed tolerance $\epsilon = 10^{-4}$, the solver converged on the solution in n=2 iterations. To be safe, I also checked that my solution was not sensitive to the initial guess for x. Using the initial guess of

$$x_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{20}$$

I found the same result as before. Finally, I created a grid $[nx, ny] = [10, 10]$ with $u = 1$ and $v = 2$. Using the same matrix operation as in Equation (17), I passed the array q of size velocity into the conjugate gradient solver and received a solution array $x_1[u[i, j]] = \frac{1}{11}$ and $x_2[v[i, j]] = \frac{7}{11}$ within n=2 iterations. This confirmed that the conjugate gradient solver could be scaled up to $[nx, ny]$ points.

4.2 Test 2: Laplacian Operator on a large grid

The mathematical definition of the laplacian of a velocity field is

$$M\vec{u} = \vec{L} \tag{21}$$

where M is the Laplacian operator, $\vec{u}$ is the velocity, and $\vec{L}$ is the laplacian of the velocity. Suppose that I have a velocity field, $\vec{u}$, such that

$$\vec{u}(x, y) = sin(x)\hat{x} + sin(y)\hat{y} \tag{22}$$

It is straightforward to work out what $\vec{L}$ should be.

$$\vec{L} = -sin(x)\hat{x} - sin(y)\hat{y} \tag{23}$$

To test the conjugate gradient solver, I assume that M and $\vec{L}$ are both known and $\vec{u}$ is unknown. I perform the test on a grid of $[nx, ny] = [100, 100]$. In practice, since the laplacian operator, M, has the boundary conditions separated, then to obtain $\vec{u}$, I must actually solve the following relation.

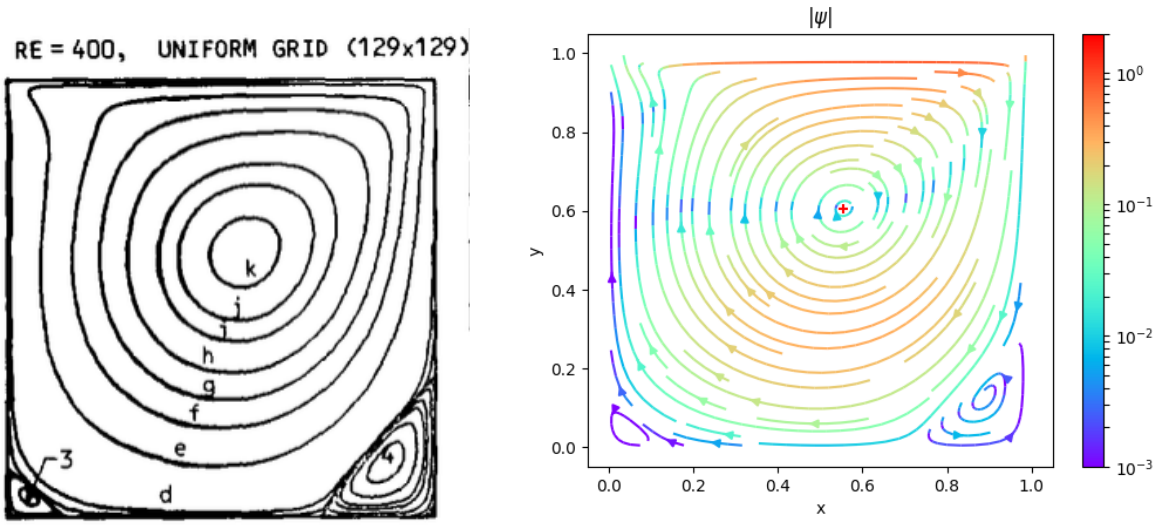
$$M\vec{u} = \vec{L} - \vec{M}_{bc} \tag{24}$$

Using imax=500 and prescribed tolerance $\epsilon = 10^{-4}$, the solver converged on the solution in n=240 iterations. The error norm of the velocity field (when compared to exact velocity field) was 2×10^{-4}, which is in agreement with the prescribed tolerance.

5 Validation of Lid-Driven Cavity Flow

6 Compare the streamline patterns.

Python has a built-in function that plots the streamlines of a velocity field. However, it does not have the capability to plot specific values of the streamfunction, so I will approach the comparison between my results and Ghia et al. using a semi-quantitative approach. In their Table 3, Ghia et al. describe a spread in streamline contour values of $[d, k] = [-10^{-4}, -0.11]$ in the primary circulation in the cavity for Re=400. Comparing their result in Figure 12a to my result in Figure 12b, I note a spread in values between the edge and center of the primary circulation of $O(10^{-3})$ to $O(10^{-1})$. For the most part, this agrees with their values. In my results, however, the streamline values do not monotonously increase nearing the center of the primary circulation. Similarly, Ghia et al. describe a spread

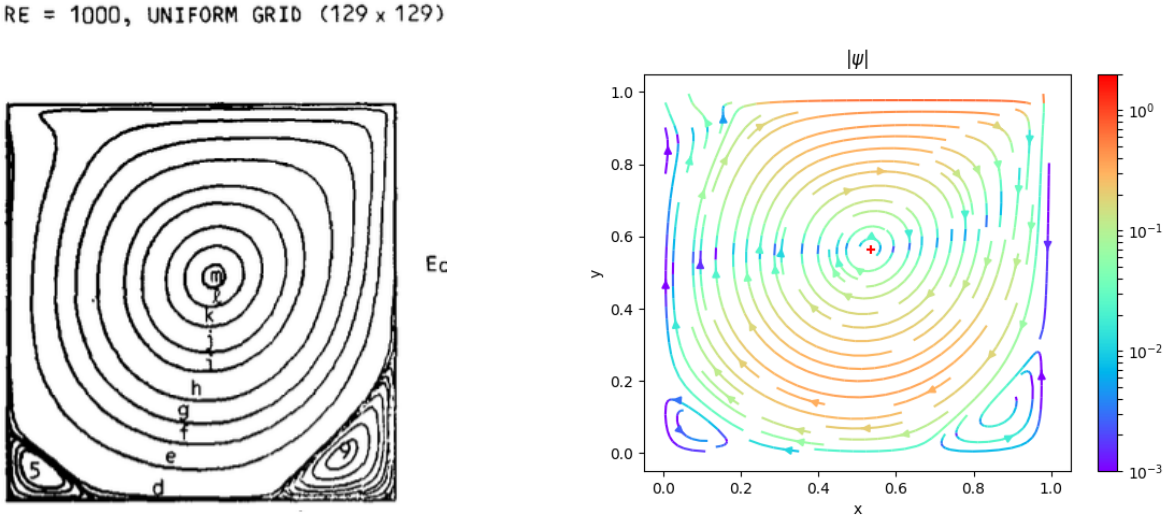


(a) (b) 100x100 at t=49.5s

Figure 12: Shows streamlines for Re=400 from (a) Ghia et al. and (b) my model. Red plus-signs denote the primary vortex center.

in streamline contour values of $[d, m] = [-10^{-4}, -0.1175]$ in the primary circulation in the cavity for Re=1000. Comparing their result in Figure 13a to my result in Figure 13b, I note a spread in values between the edge and center of the primary circulation of $O(10^{-2})$ to $O(10^{-1})$. Therefore, there is less spread in my streamline values. Again, the streamline values do not monotonously increase nearing the center of the primary circulation. One reason this may be occurring is that the simulation has not reached steady-state, so neither have the values at the center of the circulation.

In Figure 14, I compare the bottom-left secondary vortex for Re=1000 from Ghia et al. and my model, respectively. In their Table 3, Ghia et al. describe a spread in streamline contour values of $[2, 5] = [10^{-6}, 10^{-4}]$ in the bottom-left secondary vortex for Re=1000. Comparing their result in Figure 14a to my result in Figure 14b, I note a spread in



(a) (b) 100x100 at t=60s

Figure 13: Shows streamlines for Re=1000 from (a) Ghia et al. and (b) my model. Red plus-signs denote the primary vortex center.

values between the edge and center of the vortex of $O(10^{-3} - 10^{-4})$ to $O(10^{-2})$. There is a larger spread in the values reported by Ghia et al. and their values tend to be smaller overall.

In Figure 15, I compare the bottom-right secondary vortex for Re=1000 from Ghia et al. and my model, respectively. In their Table 3, Ghia et al. describe a spread in streamline contour values of $[2, 9] = [10^{-6}, 1.5 \times 10^{-3}]$ in the bottom-right secondary vortex for Re=1000. Comparing their result in Figure 15a to my result in Figure 15b, I note a spread in values between the edge and center of the vortex of $O(10^{-3} - 10^{-4})$ to $O(10^{-2})$. There is a larger spread in the values reported by Ghia et al. and their values tend to be smaller overall.

7 Compare the vorticity contours.

The structure of the vorticity field is, for the most part, consistent with the results from Ghia et al. for both Re=400 and Re=1000. The structure is most quantitatively consistent for Re=400. There are some differences between between the vorticity values around the primary vortex center. For Re=400, the vorticity contours surrounding the primary vortex center are 3-4 s^{-1} in Ghia et al. (Figure 16a). In my model, the vorticity contours surrounding the primary vortex center are 2-3 s^{-1}. For Re=1000, the vorticity contours surrounding the primary vortex center are also 3-4 s^{-1} in Ghia et al. (Figure 16a). In my model, the vorticity contours surrounding the primary vortex center are 2-3 s^{-1}. I calculate the vorticity in the primary vortex center and compare to the value found by Ghia et al. In their Table 5, Ghia et al. report a vorticity of 2.29 s^{-1} for Re=400 and 2.04 s^{-1} for Re=1000. I find that the vorticity in the primary vortex center is 2.29 s^{-1} for Re=400 and 2.03 s^{-1} for Re=1000. Therefore, there is good agreement between my model and Ghia et al. about the vorticity in the primary vortex center. The slight underestimate for Re=400 may be attributed to the allotted CPU time (t=49.5 s) and/or lower resolution. The coordinate of the primary vortex center are similar in both cases, but not perfect. Ghia et al. report coordinates of [x,y]=[0.55,0.61] and [x,y]=[0.53,0.56] for Re=400 and Re=1000, respectively. I find coordinates of [x,y]=[0.56,0.60] and [x,y]=[0.54,0.57] for Re=400 and Re=1000, respectively.

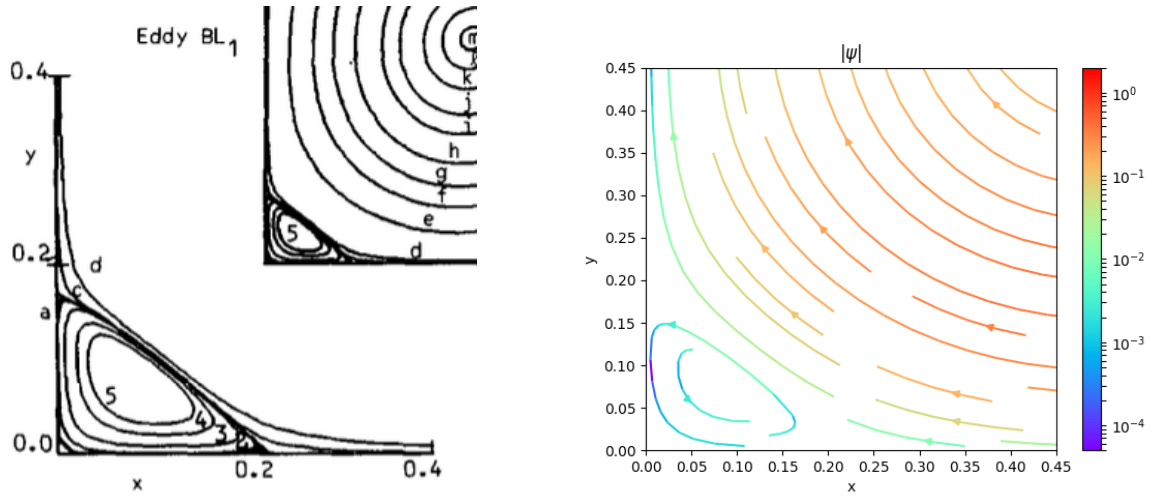
7.1 Description of vorticity operator.

To compute the vorticity, I developed an operator that takes in an array of size velocity (n_q) and returns an array of size vorticity (n_ω). Note that size vorticity is equivalent to size pressure (i.e. $n_\omega = n_p$), though the vorticity is evaluated in the top-right corner of each cell. The z-th component of the vorticity is defined as

$$\omega_z = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \tag{25}$$

and evaluated numerically as

$$\omega_z = \frac{-q[v_{i,j}] + q[v_{i+1,j}]}{dx} - \frac{-q[u_{i,j}] + q[u_{i,j+1}]}{dy} \tag{26}$$



(a) (b) 100x100 at t=60s

Figure 14: Shows streamlines of bottom-left secondary vortex for Re=1000 from (a) Ghia et al. and (b) my model. Red plus-signs denote the primary vortex center.

Therefore, the computed vorticity is first-order accurate. That's fine because I'm only using the operator diagnostically. The walls (4), corners(4), and interior(1) are treated separately. The vorticity along the top wall requires two boundary conditions ($v_{i,j} = 0$ and $v_{i+1,j} = 0$) and a ghost cell where $u_{i,j+1} = 2u_{wall} - u_{i,j}$. The vorticity along the right wall requires two boundary conditions ($u_{i,j} = 0$ and $u_{i,j+1} = 0$) and a ghost cell where $v_{i+1,j} = -v_{i,j}$. Note that the vorticity operator does not require boundary conditions or ghost values adjacent to the left or bottom walls. That is because the vorticity grid is not strictly defined on the left or bottom walls, as designed here. At the top-right corner, the vorticity is determined exclusively from boundary conditions ($u_{i,j} = 0$ and $v_{i,j} = 0$) and ghost cells ($u_{i,j+1} = 2u_{wall} - u_{i,j} = 2u_{wall}$ and $v_{i+1,j} = -v_{i,j} = 0$).

7.2 Method to locate the primary vortex center.

To locate the primary vortex center, Ghia et al. recommend locating the minimum value of the streamfunction in the domain interior. If time permits, I will implement this method. As a first step, however, I decided to find the minimum velocity magnitude in the domain interior. The coordinates corresponding the velocity magnitudes (pressure coordinates) do not overlap with the vorticity coordinates so I found the nearest vorticity coordinates to the velocity magnitude coordinates and designated this the primary vortex center. For reference, the primary vortex center is then plotted as a red plus-sign. I used this method, to start, because the python *streamplot* routine does not make the computed values of ψ available outside the routine, so it was not possible to find the minimum numerically from *streamplot*.

7.3 Transpose of the vorticity.

If I write the equation of the cross product indicial notation,

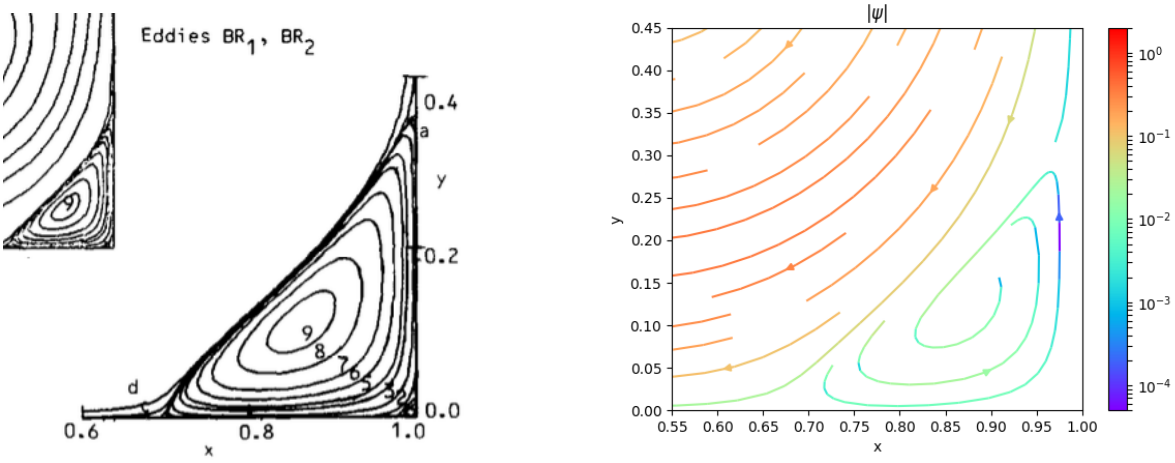
$$\nabla \times \vec{u} = \varepsilon_{ijk} \partial_j u_k \tag{27}$$

it is straightfoward to see that taking the transpose of the result (i.e. switching j and k indices) will result in a change in the sign of the Levi-Cevita symbol. This change in sign is due to even-permutations of ijk that become odd when ikj is evaluated and vice versa. As a result, the sign of the vorticity must be reversed when plotting its transpose. This makes sign of the vorticity consistent between my results and those of Ghia et al. over the domain.

8 Compare the velocity profiles.

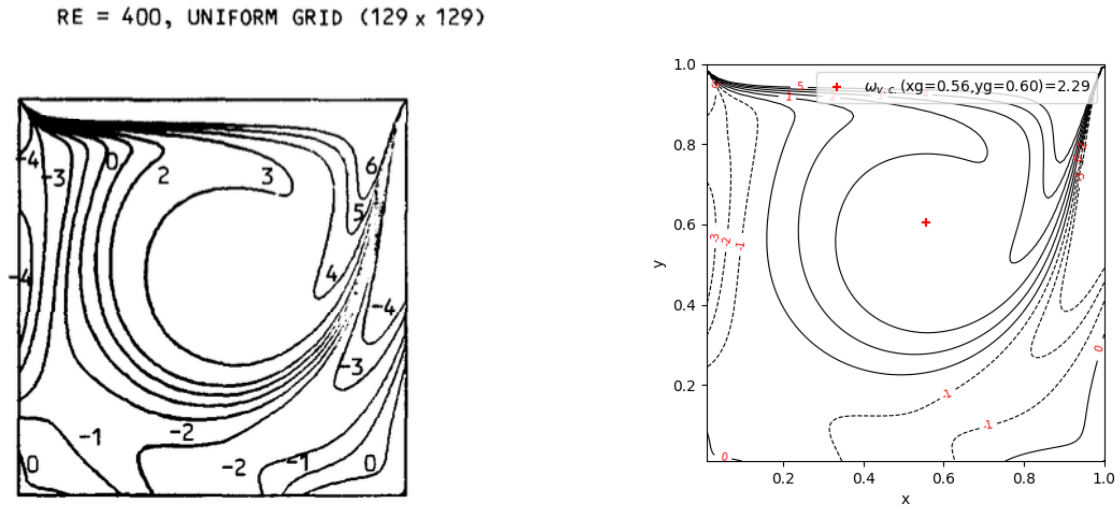
8.1 Profiles through the geometric center.

To compare the results from my lid-driven cavity flow model, I used Tables 1 and 2 from Ghia et al. to reconstruct the u- and v-velocity profiles through the geometric center and plotted the result of this analysis in Figures 18a and



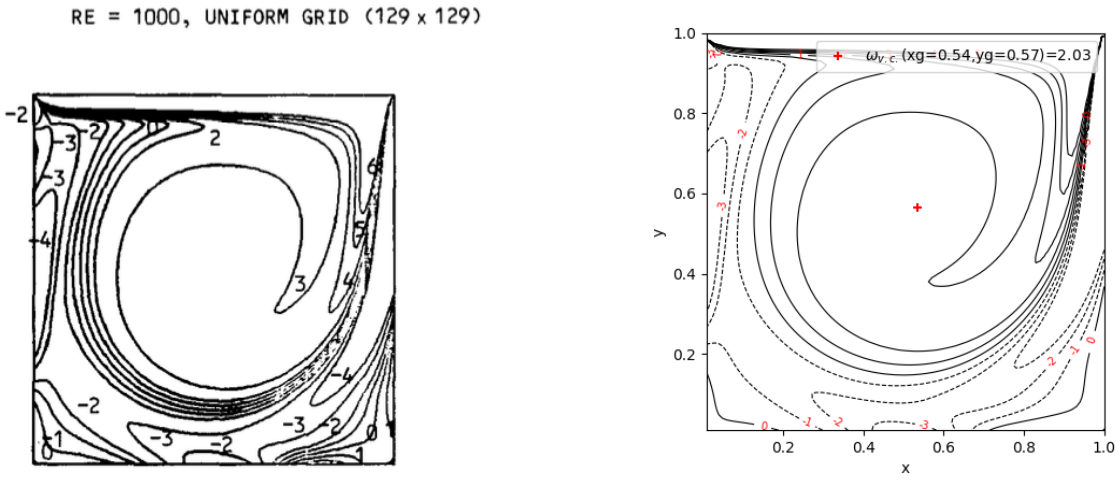
(a) (b) 100x100 at t=60s

Figure 15: Shows streamlines of bottom-right secondary vortex for Re=1000 from (a) Ghia et al. and (b) my model. Red plus-signs denote the primary vortex center.



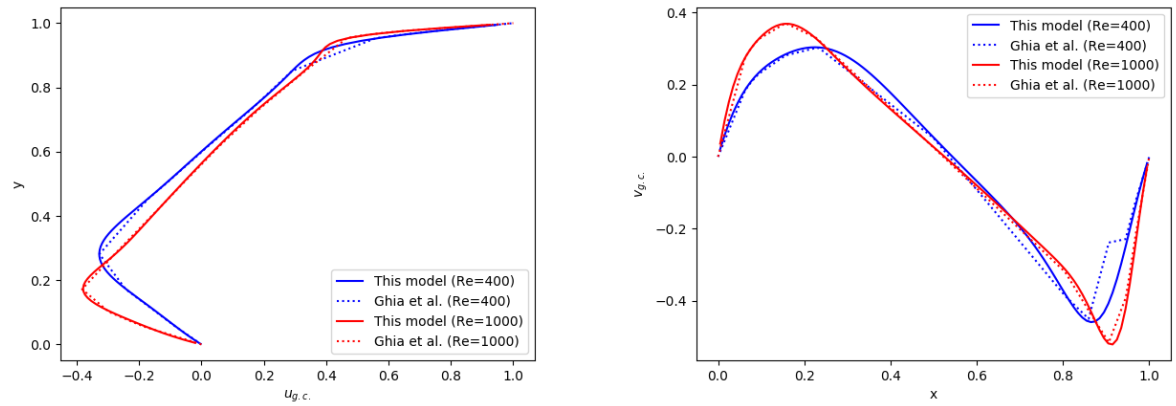
(a) (b) 100x100 at t=49.5s

Figure 16: Shows vorticity contours for Re=400 from (a) Ghia et al. and (b) my model. Red plus-signs denote the primary vortex center.



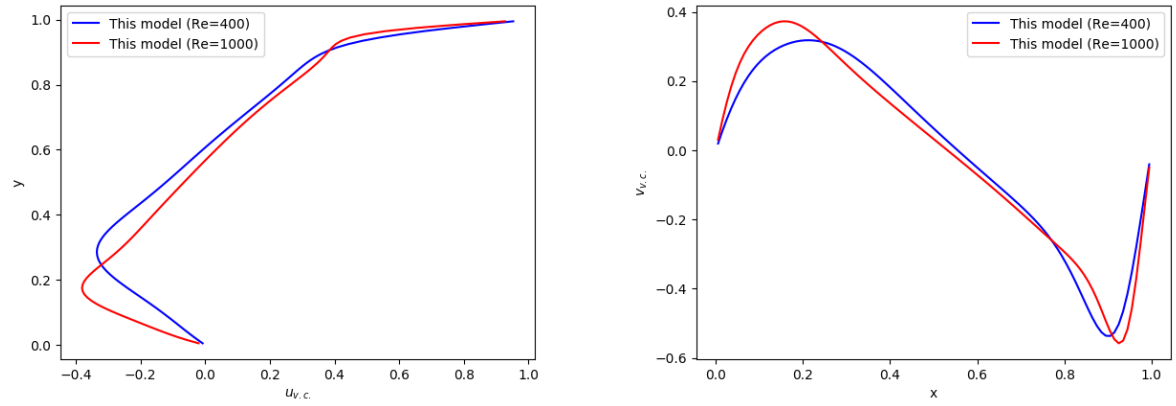
(a) (b) 100x100 at t=60s

Figure 17: Shows vorticity contours for Re=1000 from (a) Ghia et al. and (b) my model. Red plus-signs denote the primary vortex center.



(a) (b)

Figure 18: Shows (a) u-velocity and (b) v-velocity across geometric centerline for 100x100. Re=400 run for t=49.5s and Re=1000 run for t=60s.



(a) (b)

Figure 19: Shows (a) u-velocity and (b) v-velocity across primary vortex centerline for 100x100. Re=400 run for t=49.5s and Re=1000 run for t=60s.

18b. For Re=400, my model underestimates the negative u-velocities between y=0.2-0.4 and underestimates the positive u-velocities around y=0.9. Similarly, my model overestimates the positive v-velocities between x=0.2-0.4 and overestimates the negative v-velocities around x=0.9. The same general observations hold for Re=1000, though there is more agreement between the profiles. Since Re=400 was run for t=49.5s and Re=1000 was run for t=60s, it is likely this is due to differences in CPU runtime. The 'C'-shape of the u-velocity profile and the sinusoidal shape of the v-velocity profile are qualitatively similar, but there a degree of mismatch between my model and Ghia et al. This disparity may be explained in part by the difference in grid resolution and CPU time. Ghia et al. used a resolution of 129x129, which is superior to the grid that I used (100x100). My Re=400 was run for t=49.5s and Re=1000 was run for t=60s, while Ghia et al. ran their simulations for about t=1.5 minutes.

8.2 Profiles through the primary vortex center.

There was no data in Ghia et al. to reconstruct the u- and v-velocity profiles through the primary vortex center, so I plotted my results in Figures 19a and 19b and visually compared them to Figures 20 and 21. For increasing Re, Ghia et al. found that the kink around y=0.9 ascends and the kink around y=0.3 descends in Figure 20. This agrees with my Figure 19a. For increasing Re, Ghia et al. also found that the kink around x=0.9 descends and the kink around x=0.3

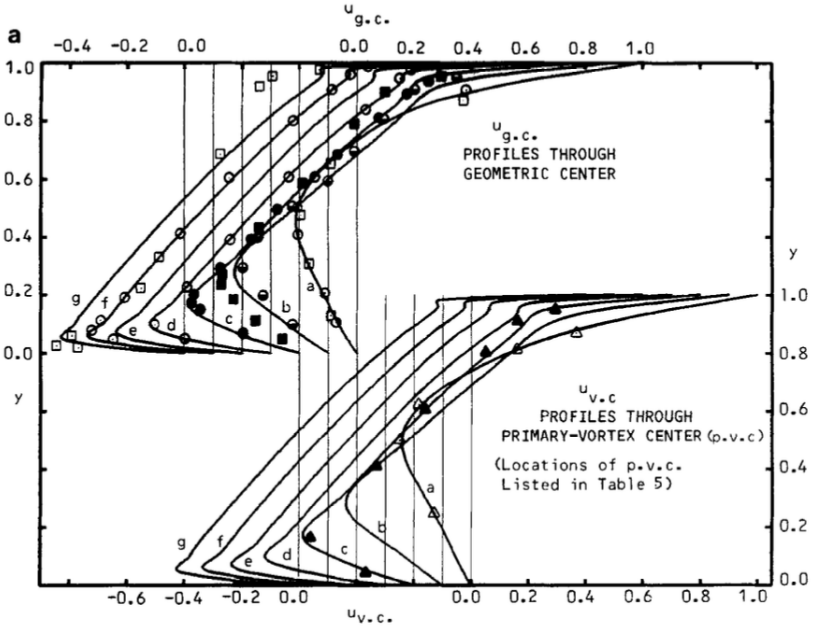


Figure 20: From Ghia et al. showing u-velocity profiles through geometric and primary-vortex centers. Re=400 is the curve denoted 'b' and Re=1000 is the curve denoted 'c'.

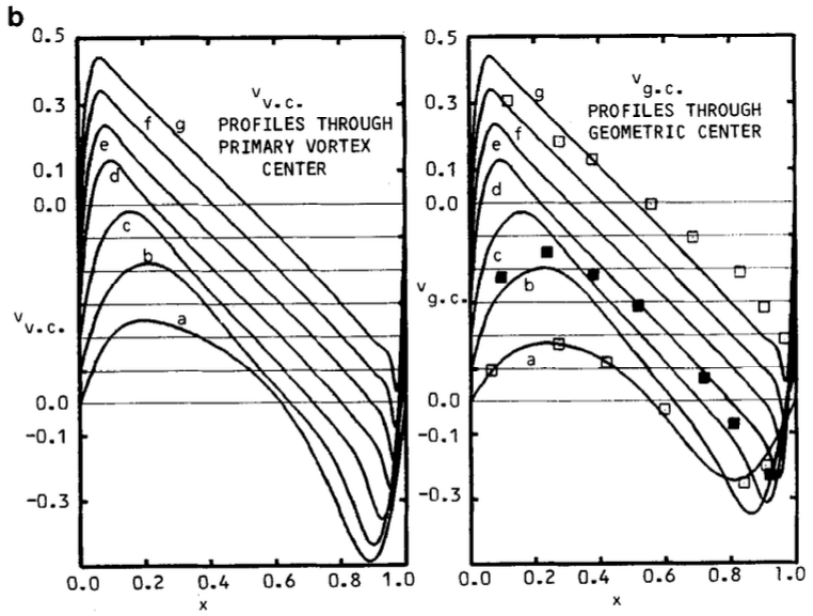


Figure 21: From Ghia et al. showing v-velocity profiles through geometric and primary-vortex centers. Re=400 is the curve denoted 'b' and Re=1000 is the curve denoted 'c'.

ascends in Figure 21. This agrees with my Figure 19b. There is agreement in the overall structure of the u-velocity and v-velocity profiles at the primary vortex structure, but it is hard to say for certain whether this is true quantitatively without plotting their data.

9 Thermal Convection in an Incompressible Fluid

In this section, I describe how I introduced thermal convection into the incompressible flow. Inspired by Sa and Kwak (1997) [3], I designed my convection experiment to be a square domain that is heated from below and cooled from above. The left and right walls have a zero heat flux boundary condition, so the temperatures there are temporally-varying. The bottom and top walls are maintained at fixed temperatures of equal magnitude but opposite sign. The thermal forcing at the bottom and top walls are expected to drive an overturning circulation. In my second experiment, I included the effect of the moving lid to see how this affects the circulation of the convectively-unstable, incompressible fluid.

9.1 Momentum Equation under the Boussinesq approximation

For a constant viscosity fluid, the incompressible Navier-Stokes momentum equation may be written as

$$\frac{\partial u}{\partial t} = -\nabla \cdot (uu) - \frac{\nabla p}{\rho} + \nu \nabla^2 u + f_b \tag{28}$$

where $\nabla \cdot (uu)$ is the conservative (divergence) form on the nonlinear term, $\frac{\nabla p}{\rho}$ is the pressure gradient term, $\nu \nabla^2 u$ is the viscous diffusion term, and f_b is the body force term. To introduce thermal convection to the momentum equation, I will apply the Boussinesq approximation. Under this approximation, variations in density are assumed to be negligible except under the influence of gravity. Defining $P = \frac{p}{\rho_0}$ and including the body force term (see subsection (9.2)), I can re-write the above equation as

$$\frac{\partial u}{\partial t} = -\nabla \cdot (uu) - \nabla P + \nu \nabla^2 u - g\beta_T (T - T_0)\hat{y} \tag{29}$$

Note that the Boussinesq approximation does not require us to re-examine the pressure-poisson equation during the projection method because the pressure field is still satisfying incompressibility within the fluid, which is now assumed to be of a constant, background density. By casting the density as a function of temperature through an equation of state, variations in the background temperature profile produce buoyancy forces that can now influence the flow. In this way, the temperature equation becomes coupled to the momentum equation.

9.2 Derivation of the body force term

Archimedes principle states that the buoyant force on an object immersed in a fluid is equal to the weight of the displaced fluid. Therefore, the buoyant force is

$$F_b = \rho_0 g \tag{30}$$

where I assume ρ_0 is the constant, background density of the fluid. Similarly, the gravitational force acting on an immersed fluid parcel of density ρ is

$$F_g = -\rho g \tag{31}$$

I can write the net force on the immersed parcel as

$$F_g' = F_g + F_b = g(\rho - \rho_0) \tag{32}$$

which may be thought of as a reduced gravity. Under the Boussinesq approximation, density variations are neglected except when multiplied by gravity. So the appropriate term in the momentum equation (when divided through by ρ_0) is

$$f_b = \frac{F_g'}{\rho_0} = g\frac{\rho - \rho_0}{\rho_0} \tag{33}$$

If I assume a linear dependence of density on temperature such that

$$\rho = \rho_0[1 - \beta_T(T - T_0)] \tag{34}$$

where β_T is the coefficient of thermal expansion, then Equation (33) becomes

$$f_b = -g\beta_T(T - T_0) \tag{35}$$

Consequently, the momentum equation becomes coupled to the temperature equation through the body force term.

9.3 Operator for the body force

This operator inputs an array of size temperature (n_T) and outputs an array of size velocity (n_q). Since the temperature is defined at the cell centers, I use 2nd-order interpolation to determine the body force acting in the y-dir where the v-velocities are defined. There are no body forces acting in the x-dir.

$$q[u[i,j]] = 0 \tag{36}$$

$$q[v[i,j]] = -g\beta_T(T_v - T_0) \tag{37}$$

where $T_v = (T_{i,j} + T_{i,j+1})/2$.

9.4 Conservation of Internal Energy

Conservation of Internal Energy for incompressible flow may be written as [2]

$$\rho \frac{De}{Dt} = \mu D : D - \nabla \cdot q \tag{38}$$

where e is internal energy per unit mass $[J/kg]$, μ is the dynamic viscosity, D is the rate of strain tensor, and q is the heat flux $[Wm^{-2}]$. The first term on the RHS represents production of heat by internal friction, which I will assume to be negligible.

$$\rho \frac{De}{Dt} = -\nabla \cdot q \tag{39}$$

Using Fourier's law, I can relate the heat flux to the local gradient in temperature.

$$q = -k\nabla T \tag{40}$$

where k is the thermal conductivity $[Wm^{-1}K^{-1}]$, which I assume to be fixed. Using the definition of the internal energy

$$e = c_v T \tag{41}$$

where c_v is the specific heat at constant volume $[Jkg^{-1}K^{-1}]$, I can rewrite Equation (39) as

$$\frac{\partial T}{\partial t} = -\nabla \cdot (\vec{u}T) + \alpha \nabla^2 T \tag{42}$$

where $\alpha = \frac{k}{c_v \rho}$ is the thermal diffusivity $[m^2 s^{-1}]$. Note that the nonlinear term is discretized in conservative (divergence) form. I will refer to Equation (42) as the temperature equation.

9.5 Solving the Temperature Equation

This subsection will describe the process of integrating the temperature equation forward in time.

$$\frac{\partial T}{\partial t} = -\nabla \cdot (\vec{u}T) + \alpha \nabla^2 T \tag{43}$$

As before, I will use 2nd-order Adams-Bashforth on the non-linear term ($A = -\nabla \cdot (\vec{u}T)$) and the Crank-Nicholson method on the linear term ($B = \nabla^2 T$).

$$T^{n+1} = T^n + \frac{\Delta t}{2}[3A^n - A^{n-1}] + \frac{\alpha \Delta t}{2}[B^{n+1} + B^n] \tag{44}$$

Absorbing B into a larger matrix operation, I can re-write this equation as

$$[I - \frac{\alpha \Delta t}{2}\nabla^2]T^{n+1} = [I + \frac{\alpha \Delta t}{2}\nabla^2]T^n + \frac{\Delta t}{2}[3A^n - A^{n-1}] + \frac{\alpha \Delta t}{2}[bc_{L,T}^{n+1} + bc_{L,T}^n] \tag{45}$$

Defining $R_T = [I - \frac{\alpha \Delta t}{2}\nabla^2]$ and $S_T = [I + \frac{\alpha \Delta t}{2}\nabla^2]$, this simplifies to

$$R_T T^{n+1} = S_T T^n + \frac{\Delta t}{2}[3A^n - A^{n-1}] + \frac{\alpha \Delta t}{2}[bc_{L,T}^{n+1} + bc_{L,T}^n] \tag{46}$$

Given boundary conditions, T^{n+1} can be found using a conjugate gradient solver.

9.5.1 Initial and Boundary Conditions

Following Sa and Kwak (1997) [3], I will specify an initial, background temperature of $T_0 = 0$. I will set the temperature of the warmer, bottom boundary and the cooler, top boundary such that $T_b = -T_t = 1$. In addition, I will specify an initial, background temperature of $T_0 = 0$ within the domain.

The boundary conditions at the left and right walls are no-flux. This means that the gradient of temperature must go to zero at the wall. In doing so, the left and right walls are insulated and buoyancy gain/loss occurs only at the bottom/top boundaries. Using a three-point stencil,

$$f_{j+1/2} = f_j + \frac{1}{2^1}\Delta f_j^{(1)} + \frac{1}{2^2}\frac{\Delta^2}{2} f_j^{(2)} + O(\Delta^3) \tag{47}$$

$$f_{j+3/2} = f_j + \frac{3}{2^1}\Delta f_j^{(1)} + \frac{3^2}{2^2}\frac{\Delta^2}{2} f_j^{(2)} + O(\Delta^3) \tag{48}$$

I find the 2nd order one-sided midpoint difference at the left wall by computing $f_{j+3/2} - 9f_{j+1/2}$.

$$f_j^{(1)} = \frac{-8f_j + 9f_{j+1/2} - f_{j+3/2}}{3\Delta} + O(\Delta^2) \tag{49}$$

To enforce no-flux at the left and right boundaries, I simply require that $f_j^{(1)} = 0$. Therefore, I must solve for the temperature at the left wall at each time step that satisfies this constraint. Specifically,

$$f_j = \frac{9f_{j+1/2} - f_{j+3/2}}{8} \tag{50}$$

For the right wall, this constraint is

$$f_j = \frac{-f_{j-3/2} + 9f_{j-1/2}}{8} \tag{51}$$

To summarize, $f_j^{(1)}$ is the gradient of temperature at the walls, and f_j is the temperature at the walls. Note that the subscript j refers to the x-coordinate in this example.

9.5.2 Gradient operator for Temperature Equation

Temperature and pressure are both defined at the same coordinates, so the gradient operator for the temperature equation is almost identical to the one for pressure. The main difference is that temperature is not pinned anywhere in the domain. This operator requires no boundary conditions.

9.5.3 Divergence operator for Temperature Equation

Temperature and pressure are both defined at the same coordinates, so the divergence operator for the temperature equation is almost identical to the one for pressure. The main difference is that the input to the divergence operator is now evaluated in the bottom-left corner. This operator requires boundary conditions, but no ghost cells.

9.5.4 $bc_{D,T}^n$

This routine is where the heat flux boundary conditions are applied. At the left and right walls, I simply define the flux to be zero. At the bottom and top walls,respectively, I use 2nd order one-sided midpoint difference to determine the heat flux.

$$f_j^{(1)} = \frac{-8f_j + 9f_{j+1/2} - f_{j+3/2}}{3\Delta} + O(\Delta^2) \tag{52}$$

$$f_j^{(1)} = \frac{f_{j-3/2} - 9f_{j-1/2} + 8f_j}{3\Delta} + O(\Delta^2) \tag{53}$$

Here, f_j is the temperature at the wall with the midpoint values corresponding to the actual temperature grid. This routine takes in an array of size temperature and outputs an array of size temperature. In the time-integration step, I make the approximation that $bc_{D,T}^{n+1} = bc_{D,T}^n$.

9.5.5 $bc_{L,T}^n$ and $bc_{L,T}^{n+1}$

Since I am computing the laplacian of the temperature field using a divergence and gradient operator, the boundary condition for the laplacian comes from the divergence operator because no boundary conditions are used in the gradient operator. Stated mathematically, $bc_{L,T}^n = bc_{D,T}^n$. Since the temperature boundary conditions are constant in time, this means that $bc_{L,T}^{n+1} = bc_{L,T}^n$.

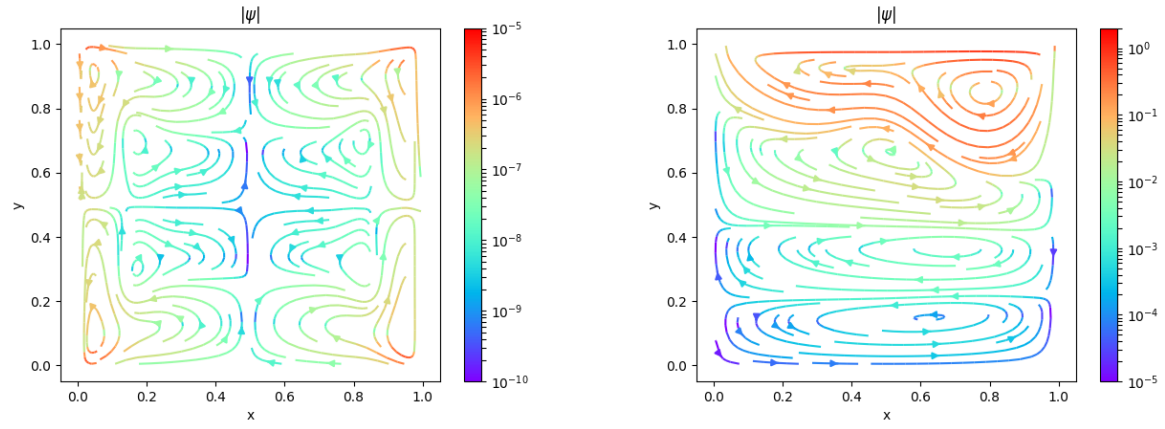
9.5.6 Nonlinear operator for Temperature Equation

The nonlinear operator may be written in indicial notation as

$$-A = \nabla \cdot (\vec{u}T) = \partial_i(u_i T) \tag{54}$$

Expanded, this reads

$$\partial_i(u_i T) = \partial_x(uT) + \partial_y(vT) \tag{55}$$



(a) $u_{wall} = 0$ (b) $u_{wall} = 1$

Figure 22: Shows streamlines of Re=400, 100x100 simulation with thermal convection at t=60s with and without a moving lid.

I will discretize the above equation using a 2nd-order accurate scheme.

$$\frac{\partial uT}{\partial x} = \frac{u^r T^r - u^l T^l}{\Delta x} \tag{56}$$

$$\frac{\partial vT}{\partial y} = \frac{v^t T^t - v^b T^b}{\Delta y} \tag{57}$$

where the superscripts r, l, t, b denote right, left, top, and bottom, respectively. Since the nonlinear operator will be evaluated at temperature coordinates, I use 2nd-order interpolation of temperature at the velocity coordinates.

$$T^r = \frac{T_{i,j} + T_{i+1,j}}{2} \tag{58}$$

$$T^l = \frac{T_{i-1,j} + T_{i,j}}{2} \tag{59}$$

$$T^t = \frac{T_{i,j} + T_{i,j+1}}{2} \tag{60}$$

$$T^b = \frac{T_{i,j-1} + T_{i,j}}{2} \tag{61}$$

I require that $T^r = 0$ at the right boundary, $T^l = 0$ at the left boundary, $T^t = T_t$ at the top boundary, and $T^b = T_b$ at the bottom boundary (see subsection 9.5.1). To enforce this, I use a boundary condition look-up function rather than introducing ghost cells into the computation, which simplifies the process. This look-up function returns the temperature of the bottom, top, left, and right walls. The left and right walls have the temperature that satisfies no heat flux and therefore are different at each time step. Similarly, only velocity boundary conditions (i.e. no ghost cells) are required in this formulation. Note that the thermal nonlinear term has dimensions of temperature, n_T.

$$u^r = u_{i,j} \tag{62}$$

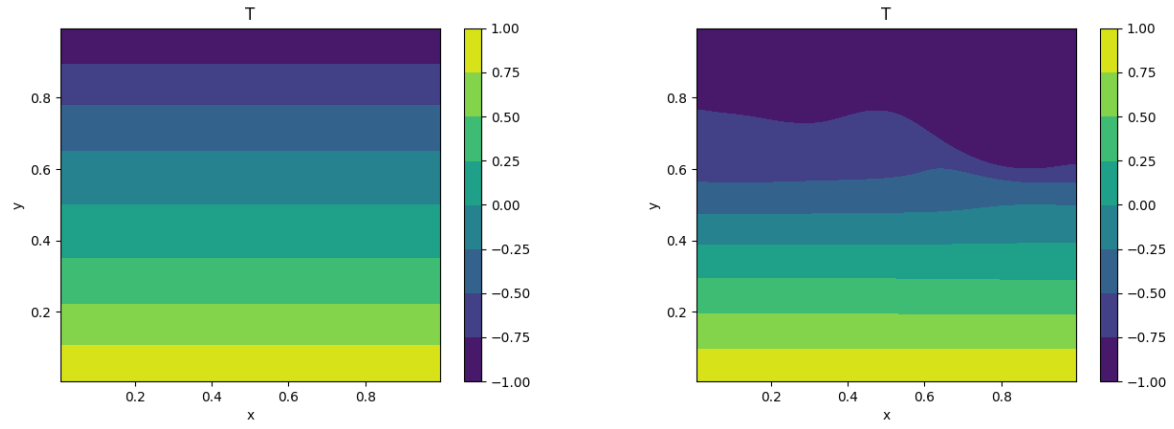
$$u^l = u_{i-1,j} \tag{63}$$

$$v^t = v_{i,j} \tag{64}$$

$$v^b = v_{i,j-1} \tag{65}$$

9.6 Preliminary results from Thermal Convection in a Static Cavity

Figure 22a shows the streamlines of a Re=400, 100x100 simulation with heating from below and cooling from above when the top wall is not moving. Note the streamfunction takes values between $O(10^{-10})$ and $O(10^{-5})$. For the chosen



(a) $u_{wall} = 0$ (b) $u_{wall} = 1$

Figure 23: Shows temperature profile of Re=400, 100x100 simulation with thermal convection at t=60s with and without a moving lid.

parameter values, the thermal convection appears to be quite weak. The convection is organized into four quadrants of the domain, with each quadrant further subdivided into multiple overturning cells. Overall, there is ascending flow along the right wall and descending flow along the left wall. Due to the organization of the convective cells, the v-velocities along the geometric centerline alternate between upward and downward motion. Along the bottom wall, the flow converges at x=0.5 and ascends. Similarly along the top wall, the flow converges at x=0.5 and descends.

Figure 23a shows the temperature profile of a Re=400, 100x100 simulation with heating from below and cooling from above when the top wall is not moving. Despite the overturning circulation, the temperature profile is unstably stratified. Temperatures are warmer near the bottom wall and colder near the top wall. This instability should generate convection which mixes denser (cooler) water downward and less dense (warmer) water upward. Note that a stable stratification would have warm fluid atop cold fluid. I think it's safe to say that I made a mistake somewhere in the code.

9.7 Preliminary results from Thermal Convection in a Lid-driven Cavity

Figure 22b shows the streamlines of a Re=400, 100x100 simulation with heating from below and cooling from above when the top wall is moving at $u_{wall} = 1ms^{-1}$. Note the streamfunction takes values between $O(10^{-5})$ and $O(10^{0})$. Without thermal convection, recall that the flow organizes into a primary circulation with bottom-corner eddies. With thermal convection, four stacked convective cells develop. These convective cells alternate between clockwise and and counter-clockwise flow. Due to the organization of the convective cells, the v-velocities along the left and right walls alternate between upward and downward motion. Along the bottom and top walls, however, there is rightward flow.

Figure 23b shows the temperature profile of a Re=400, 100x100 simulation with heating from below and cooling from above when the top wall is moving at $u_{wall} = 1ms^{-1}$. Despite the overturning circulation, the temperature profile is unstably stratified. Temperatures are warmer near the bottom wall and colder near the top wall. This instability should generate convection which mixes denser (cooler) water downward and less dense (warmer) water upward. In contrast to the case without the moving lid, here the moving lid induces a strong overturning circulation that mixes cooler water downward and displaces the isotherms. Despite this influence, the profile remains unstably stratified. Note that a stable stratification would have warm fluid atop cold fluid. I think it's safe to say that I made a mistake somewhere in the code.

9.8 Closing comments about thermal convection

From examining the scientific literature and based on my own physical intuition, I am confident that my thermal convection results are incorrect. More work is needed to find and correct the errors that I made when implementing the physics described in the sections above. There is not sufficient time to do so before this report is due, however.

References

[1] U Ghia, K.N Ghia, C.T Shin, High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method, Journal of Computational Physics, Volume 48, Issue 3, 1982, Pages 387-411, ISSN 0021-9991, https://doi.org/10.1016/0021-9991(82)90058-4.

[2] Kajishima, T., & Taira, K. (2017). Computational Fluid Dynamics. Springer International Publishing. https://doi.org/10.1007/978-3-319-45304-0

[3] Sa, Jong-Youb. and Kwak, Dochan. and Ames Research Center. A numerical method for incompressible flow with heat transfer [microform] / Jong-Youb Sa, Dochan Kwak National Aeronautics and Space Administration, Ames Research Center ; National Technical Information Service, distributor Moffett Field, Calif. : [Springfield, Va 1997] from https://ntrs.nasa.gov/api/citations/19970017611/downloads/19970017611.pdf

[4] J. R. Shewchuk, An Introduction to the Conjugate Gradient Method Without the Agonizing Pain, 1994, [online] Available: http://www.cs.cmu.edu/%7Equake-papers/painless-conjugate-gradient.pdf.

[5] Conjugate gradient method. Retrieved May 16, 2021, from https://en.wikipedia.org/wiki/Conjugate_gradient_method

[6] Stream function. Retrieved May 31, 2021, from https://en.wikipedia.org/wiki/Stream_function