Jon Grote

Dr. Wei

ME 860

13 March 2022

Project I Report: Lid-Driven Cavity Flow

BACKGROUND

In the software engineering world, the 'hello world' program is the first to be run, and the test that most easily highlights differences between programming languages. In CFD, the 'hello world' program is the Lid-Driven Cavity Flow problem. It is a benchmark for all numerical techniques, including new ones, to be compared and evaluated by, and a good introduction to actual CFD work. The problem is set up as such: A square (often square, but rectangular is also permitted if one wants to look at differences when $\Delta x \neq \Delta y$) domain consists of 3 stationary walls and 1 moving wall, or the lid. The lid moves with constant velocity in 1 direction. The boundary conditions are such that the horizontal and vertical velocities are zero at the stationary walls as per the no-slip condition. From these boundary conditions, U-velocity, V-velocity, and Pressure are solved from the discretized governing equations.

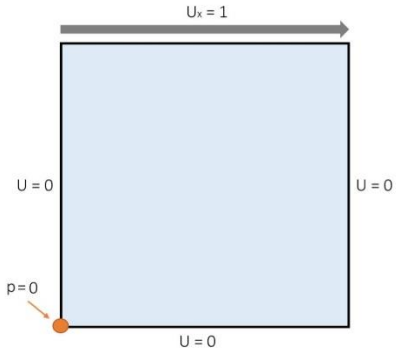


Figure 1: Boundary Conditions

GOVERNING EQUATIONS

The governing equations of the system are the continuity equation, u-momentum equation, and v-momentum equation, which together form the Navier-Stokes equations.

Continuity:

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

U-momentum:

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

V-momentum:

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

Of note is that these are in the non-dimensional form of the N-S equations, allowing for solving the equations for any given Reynold's number. Results produced at varying Reynold's numbers is an essential part of the validation step of numerical simulations, especially that of driven cavity flow.

DISCRETIZATION

Discretization of the governing Navier-Stokes equations is done using the Marker and Cell method. The use of the MAC method, which is done using a staggered grid (see below), permits coupling of u, v, and p solutions at adjacent grid points, without the appearance of oscillatory or checker-board solutions that can arise using a collocated grid.

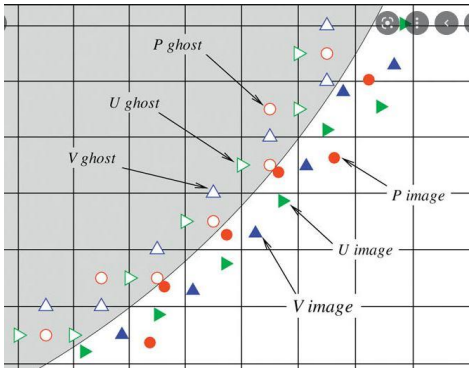


Figure 2: ghost cell conditions across physical boundary

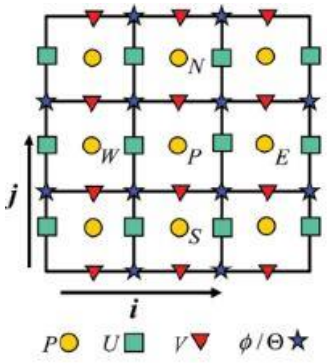


Figure 3: Staggered Mesh Configuration

Utilizing a staggered grid necessitates the use of ghost cell conditions, essentially extensions of the u, v, and p domains, that capture the conditions of u, v, and p across the physical boundary. More accurately they are the mirror (inverse, for u and v) value, such that when averaged with the corresponding cell in the real domain, they will cancel out to be 0 (for u and v). An interesting side-note, if anyone other than you, Dr. Wei, reads this in the future, regarding setting up the ghost cells in code; when MIT gives you a spatial discretization suggestion like so:

Spatial discretization

field quantity	interior resolution	resolution with boundary points
pressure P	$n_x \times n_y$	$(n_x+2) \times (n_y+2)$
velocity component U	$(n_x-1) \times n_y$	$(n_x+1) \times (n_y+2)$
velocity component V	$n_x \times (n_y-1)$	$(n_x+2) \times (n_y+1)$
stream function Q	$(n_x-1) \times (n_y-1)$	$(n_x+1) \times (n_y+1)$

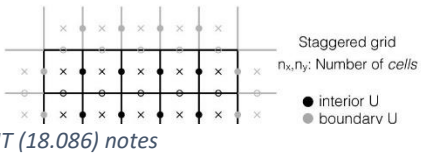


Figure 4: MIT (18.086) notes

the resolution given is actual *size* dimensions in the x and y directions, not *indicial* dimensions to initialize and slice through your arrays. I was thinking Pythonically, not dimensionally. For project I, I was able to build by domain in python visually, that is I could set my origin in the upper left corner, and build it row by column. Debugging would show the [0,0] origin in the UL corner,

and the stream function value in the cells that corresponded physically to the problem. Because of how the MAC algorithm equations work, this is not possible without significant adjustment for a staggered gird. The easiest way to illustrate what I am saying is that if one (as I did originally before fixing my code) built the cavity row by column, physically, then the debug will show the lid as the 0th row. For the MAC to work the way it is written, the lid must be the 0th column: basically the entire cavity is rotated CCW by 90 degrees. This is the issue that was completely braking my code until I had a helpful friend look at my data output and realize what was happening. But I digress, regarding the discretization of the N-S equations for a MAC method, specific steps must be followed sequentially to ensure a proper solution: 1) get F and G, 2) solve the pressure Poisson problem, 3) substitute the pressure solution back to get u and v for the next iteration level. The explicit algorithm employed by the MAC method looks like this:

1)

$$F^n_{i+\frac{1}{2},j} = u^n_{i+\frac{1}{2},j} + \Delta t\left\{\frac{u_{i+\frac{3}{2},j} - 2u_{i+\frac{1}{2},j} + u_{i-\frac{1}{2},j}}{Re\Delta x^2} + \frac{u_{i+\frac{1}{2},j+1} - 2u_{i+\frac{1}{2},j} + u_{i+\frac{1}{2},j-1}}{Re\Delta y^2} - \frac{u^2_{i+1,j} - u^2_{i,j}}{\Delta x} - \frac{(uv)_{i+\frac{1}{2},j+\frac{1}{2}} - (uv)_{i+\frac{1}{2},j-\frac{1}{2}}}{\Delta y}\right\}$$

$$G^n_{i,j+\frac{1}{2}} = v^n_{i,j+\frac{1}{2}} + \Delta t\left\{\frac{v_{i+1,j+\frac{1}{2}} - 2v_{i,j+\frac{1}{2}} + v_{i-1,j+\frac{1}{2}}}{Re\Delta x^2} + \frac{v_{i,j+\frac{3}{2}} - 2v_{i,j+\frac{1}{2}} + v_{i,j-\frac{1}{2}}}{Re\Delta y^2} - \frac{v^2_{i,j+1} - v^2_{i,j}}{\Delta y} - \frac{(uv)_{i+\frac{1}{2},j+\frac{1}{2}} - (uv)_{i-\frac{1}{2},j+\frac{1}{2}}}{\Delta x}\right\}$$

2)

$$\left[\frac{p_{i-1,j} - 2p_{i,j} + p_{i+1,j}}{\Delta x^2} + \frac{p_{i,j-1} - 2p_{i,j} + p_{i,j+1}}{\Delta y^2}\right]^{n+1} = \frac{1}{\Delta t}\left[\frac{F^n_{i+\frac{1}{2},j} - F^n_{i-\frac{1}{2},j}}{\Delta x} + \frac{G^n_{i,j+\frac{1}{2}} - G^n_{i,j-\frac{1}{2}}}{\Delta y}\right]$$

3)

$$u_{i+\frac{1}{2},j}^{n+1} = F_{i+\frac{1}{2},j}^{n} - \frac{\Delta t}{\Delta x}\left[p_{i+1,j}^{n+1} - p_{i,j}^{n+1}\right]$$

$$v_{i,j+\frac{1}{2}}^{n+1} = G_{i,j+\frac{1}{2}}^{n} - \frac{\Delta t}{\Delta y}\left[p_{i,j+1}^{n+1} - p_{i,j}^{n+1}\right]$$

When storing values in arrays, one *might* associate storage location (i,j) with $u_{i+\frac{1}{2},j}$ & $v_{i,j+\frac{1}{2}}$ & $p_{i,j}$.

When running the program, 2 special time step considerations MUST be implemented to ensure convergence:

$$.25(|u| + |v|)^2 \cdot \Delta t \cdot Re \leq 1$$

$$\frac{\Delta t}{(Re \cdot \Delta x^2)} \leq .25$$

NOTES ON CODE

For my program, I decided upon a Poisson pressure solution tolerance of {tol = .000001}, with an overall maximum error for u and v of {max_err = .00001}. The pressure solutions has a lower tolerance because errors are propagated from it to the solutions of u and v, so keeping the pressure error low optimizes the solutions of u and v. Computationally this is very intensive. And as python is an interpreted language, the code for the pressure solver equation (and the arrays) has to be "rebuilt" every time the interpreter comes to it. This is a major bottleneck in the performance of the program, and when compared to classic C, is just not fast enough to be practical for large arrays. However, Python offers a just-in-time compiler, *jit*, suggested by a friend. While the entire program can not be optimized using jit, the part of the code that solves the Poisson equation can be, and that is what I did. Essentially jit pre-compiles the entire function for the Poisson equation solution, so it doesn't require a "rebuild" every time the interpreter gets to the line for the function call. Some results of implementing jit are included in the results section.

VALIDATION

In showing the performance of the program for specific gird sizes and Reynold's numbers for the optimized and not-optimized versions of the code, it can be seen that the pre-compiled optimized version is significantly faster, thanks to the *jit* extension of the *numba* module.

	TIME COMPARISON, 21x21 grid	
	NO-JIT	JIT

Re	time	iterations	time	iterations
100	21.32"	295	3.22"	295
400	2'12"	2268	24.33"	2268
1000	5'43"	6573	1'07"	6573

Comparison with Kim and Moin was the first true step towards validation; specifically the u-midplane (cavity centerline) velocity at Re=400 for 11x11, 21x21, and 31x31 grids. The results are shown here.

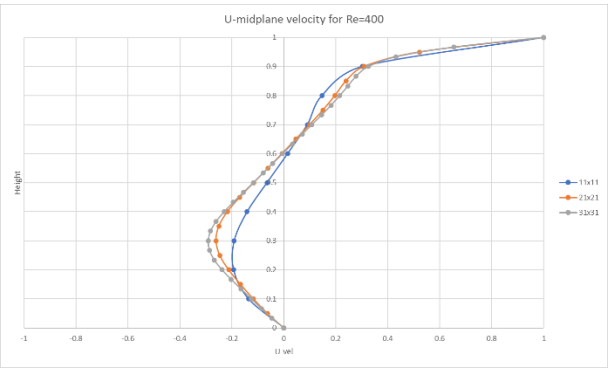


Figure 5: my comparison to K&M

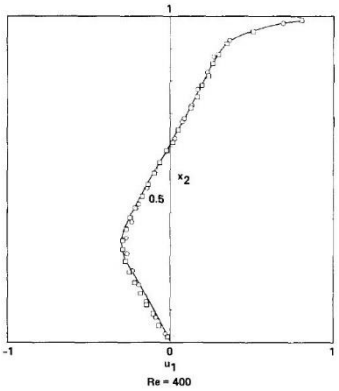


FIG. 6. Profile of streamwise velocity at the midplane of the cavity ($x_1 = 0.5L$) for Re = 400: —, 31 × 31, Burggraf [16], Goda [17]; ○, 21 × 21; □, 31 × 31, present results.

In the comparison, while not exact, at Re=400, a smaller grid of 11x11 does not completely match the precision of the 21x21 and 31x31 grid sizes, both of which are sufficient to capture the mid-plane u velocity accurately, and match the results of K&M(1985).

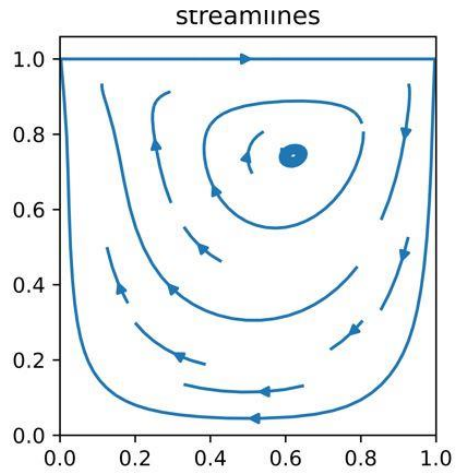
In comparing to Ghia, et. al (1982), again the midplane u velocity is taken, this time for a much larger grid size (129x129), at specific Reynold's numbers. Streamlines for specific Reynold's numbers are also compared between my results and Ghia. The results are shown here.

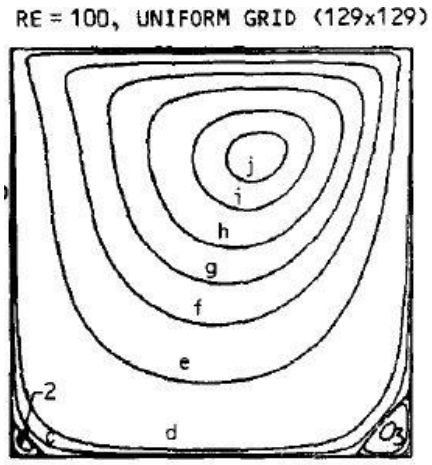
u-midplane					
	129x129	*129x129*	*129x129*	*129x129*	grid pt
(Yheight)	*Re=100*	*Re=400*	*Re=1000*	*Re=3200*	number
1	1	1	1	1	129
0.976562	0.841072	0.75842	0.655883	0.473183	126
0.96875	0.788469	0.684438	0.570518	0.410784	125
0.960938	0.73688	0.617539	0.505694	0.383595	124
0.953125	0.6867	0.558753	0.459445	0.373385	123
0.851562	0.226136	0.288302	0.326061	0.281357	110
0.734375	-0.0074	0.160785	0.185105	0.194724	95
0.617188	-0.14552	0.019322	0.05568	0.09573	80
0.5	-0.20657	-0.11758	-0.06353	-0.02791	65
0.453125	-0.20787	-0.17424	-0.10916	-0.07436	59
0.28125	-0.14664	-0.32484	-0.27631	-0.19995	37
0.171875	-0.09351	-0.23575	-0.37526	-0.25311	23
0.101562	-0.0592	-0.14007	-0.28336	-0.29838	14
0.070313	-0.04292	-0.09873	-0.20915	-0.33152	10
0.0625	-0.03866	-0.08843	-0.18961	-0.3393	9
0.054688	-0.0343	-0.07808	-0.16966	-0.34357	8
0	0	0	0	-0.37606	1

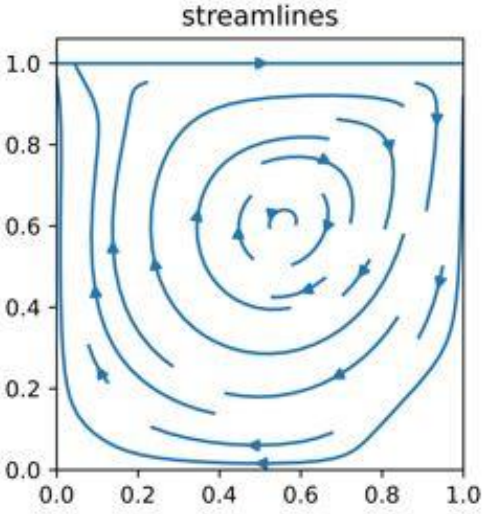
TABLE I

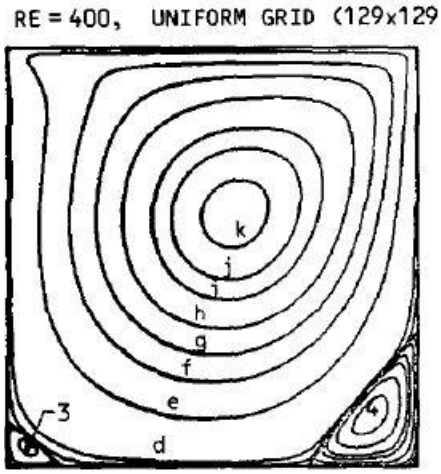
Results for u-velocity along Vertical Line through Geometric Center of Cavity

129-grid pt. no.	y	Re 100	400	1000	3200	5000	7500	10,000
129	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
126	0.9766	0.84123	0.75837	0.65928	0.53236	0.48223	0.47244	0.47221
125	0.9688	0.78871	0.68439	0.57492	0.48296	0.46120	0.47048	0.47783
124	0.9609	0.73722	0.61756	0.51117	0.46547	0.45992	0.47323	0.48070
123	0.9531	0.68717	0.55892	0.46604	0.46101	0.46036	0.47167	0.47804
110	0.8516	0.23151	0.29093	0.33304	0.34682	0.33556	0.34228	0.34635
95	0.7344	0.00332	0.16256	0.18719	0.19791	0.20087	0.20591	0.20673
80	0.6172	−0.13641	0.02135	0.05702	0.07156	0.08183	0.08342	0.08344
65	0.5000	−0.20581	−0.11477	−0.06080	−0.04272	−0.03039	−0.03800	0.03111
59	0.4531	−0.21090	−0.17119	−0.10648	−0.86636	−0.07404	−0.07503	−0.07540
37	0.2813	−0.15662	−0.32726	−0.27805	−0.24427	−0.22855	−0.23176	−0.23186
23	0.1719	−0.10150	−0.24299	−0.38289	−0.34323	−0.33050	−0.32393	−0.32709
14	0.1016	−0.06434	−0.14612	−0.29730	−0.41933	−0.40435	−0.38324	−0.38000
10	0.0703	−0.04775	−0.10338	−0.22220	−0.37827	−0.43643	−0.43025	−0.41657
9	0.0625	−0.04192	−0.09266	−0.20196	−0.35344	−0.42901	−0.43590	−0.42537
8	0.0547	−0.03717	−0.08186	−0.18109	−0.32407	−0.41165	−0.43154	−0.42735
1	0.0000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

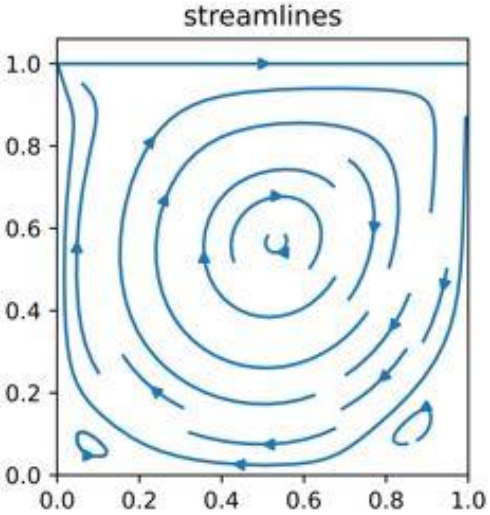


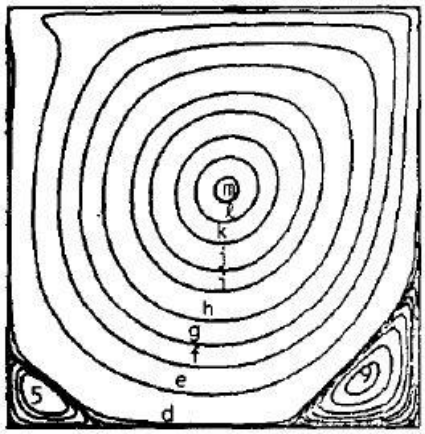






RE = 1000, UNIFORM GRID (129 x 12





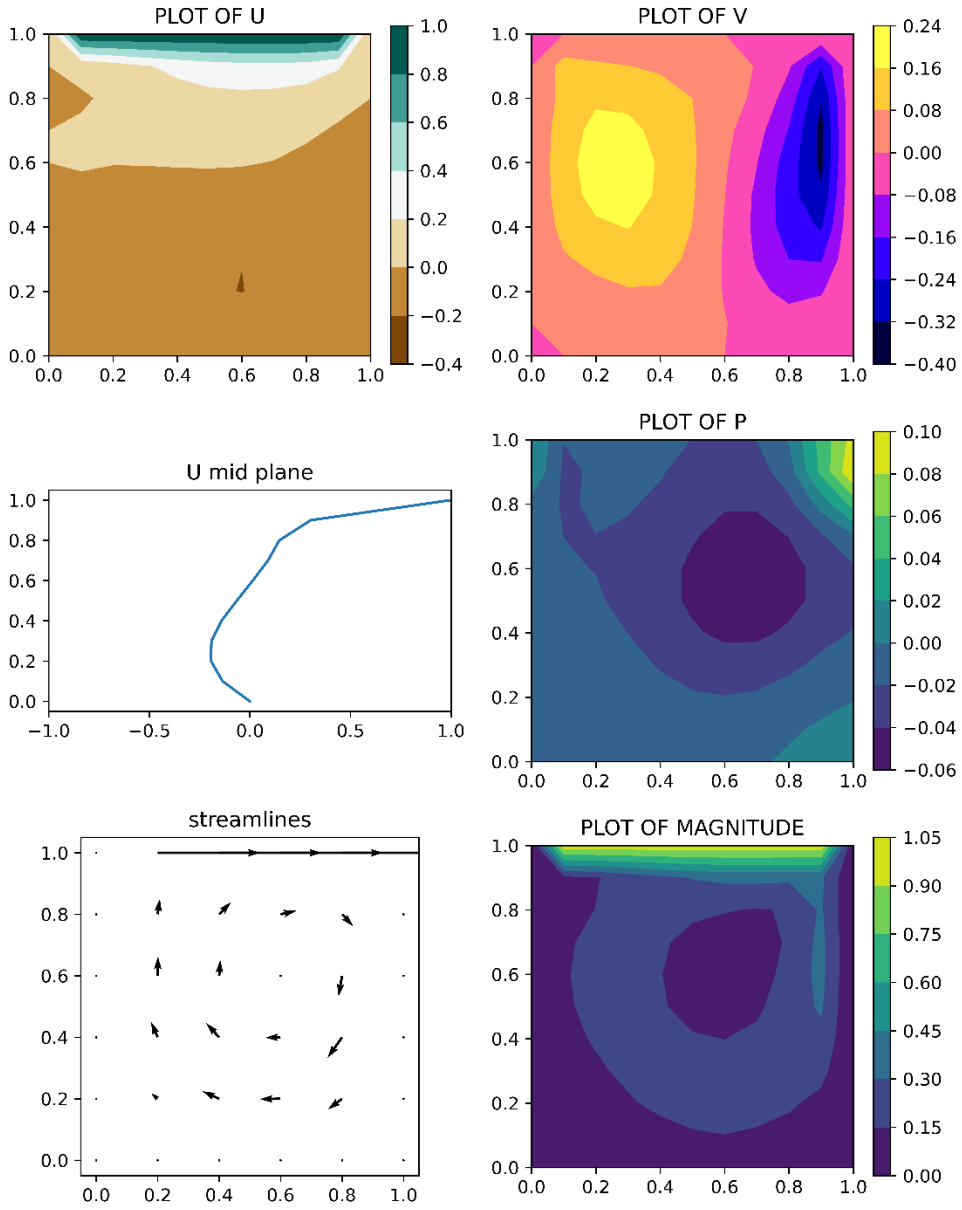
Here is the comparison between streamlines, shown above, with my results on the left and Ghia, et. al. on the right. At higher Reynold's numbers, it can be seen that the main vortex moves from the upper right corner towards the middle of the cavity. Also, while it is more evident in the Re=1000 results, secondary and tertiary vortices have formed in the bottom left and bottom right corners of the cavity for both the Re=400 and Re=1000. The only reason it is not seen in my Re=400 is the density of my streamlines is set to be lower than the default value to avoid a messy result, which can be seen when velocity vectors are displayed.

For the u-midplane velocity for a 129x129 grid, specific observations that do not conform to the validation criteria, are at Re=3200. Most results are very close, but near the lid, and in the middle of the cavity, values are slightly off expected (Ghia). For lower Reynold's numbers, results concur. This may be an error produced by excel, as I used a sort function (greatest to smallest). Numbers could have been jumbled, as I made an assumption about the gradient being constant.

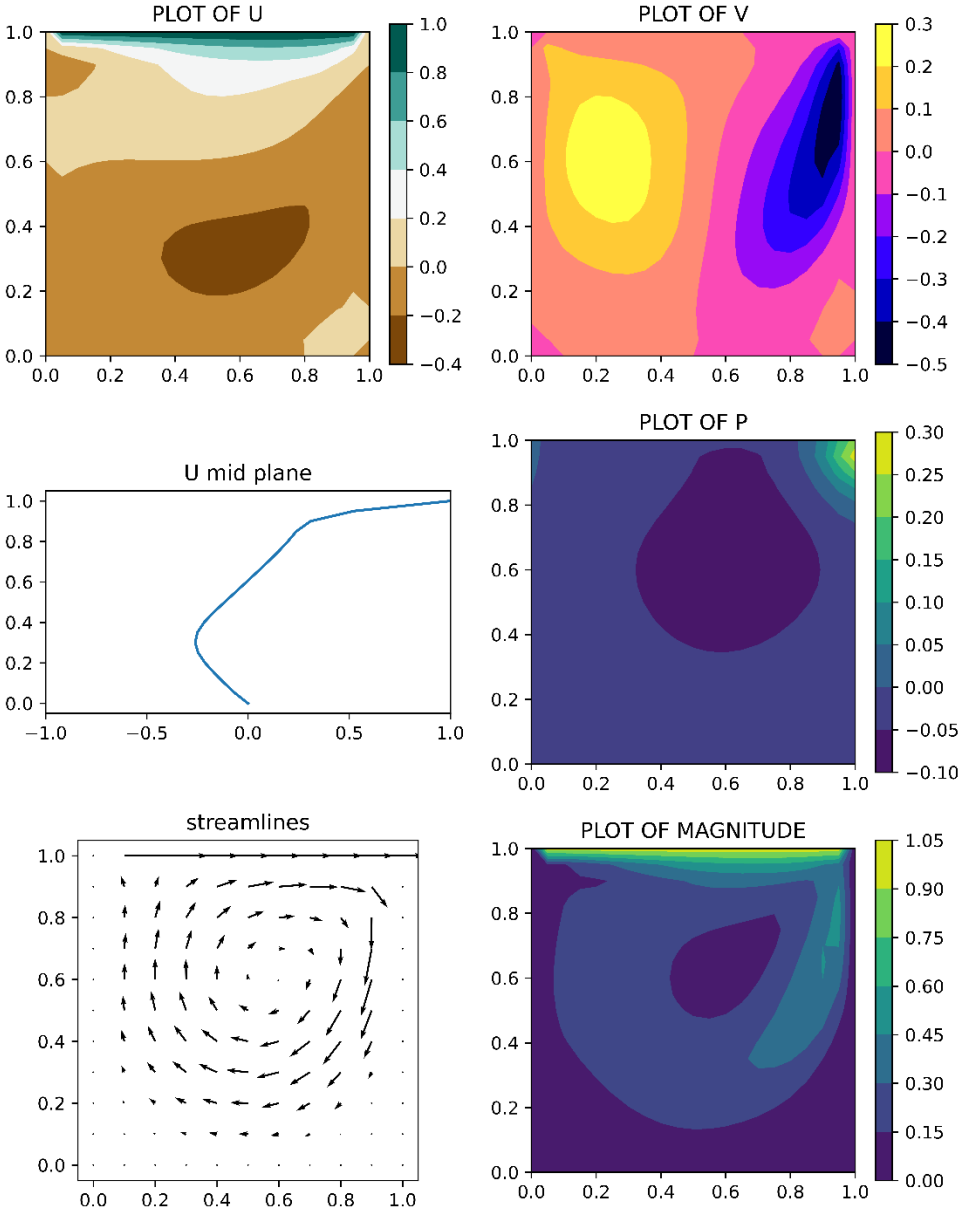
RESULTS

Displayed here are the full results for all tests run. These provide a deeper look at the nature of the flows in question.

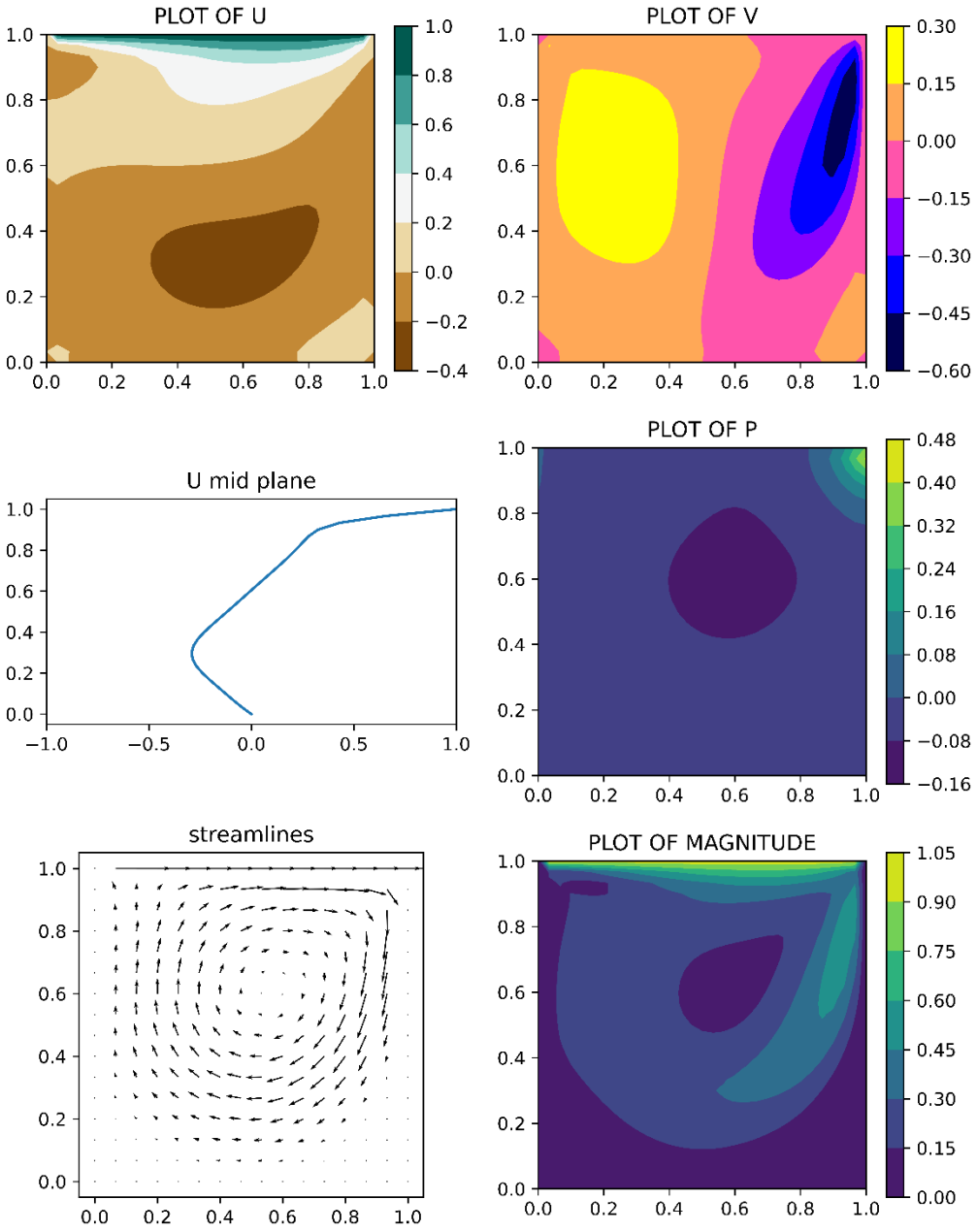
11x11 grid, Re=400



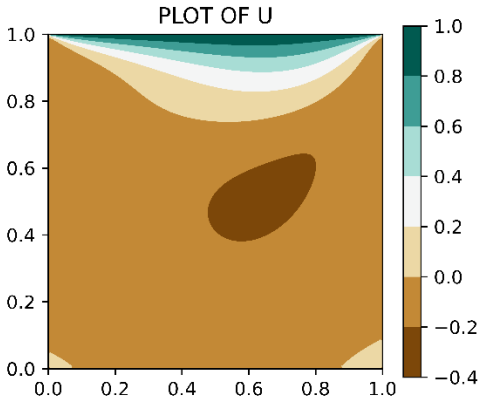
21x21 grid, Re=400

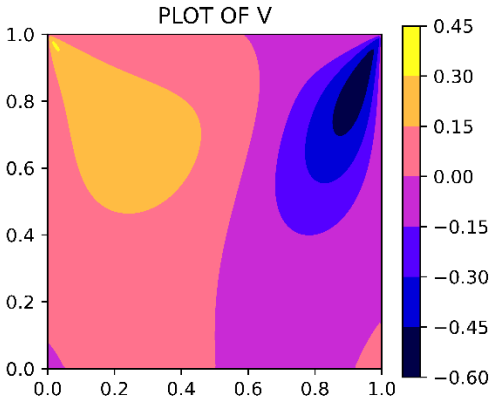


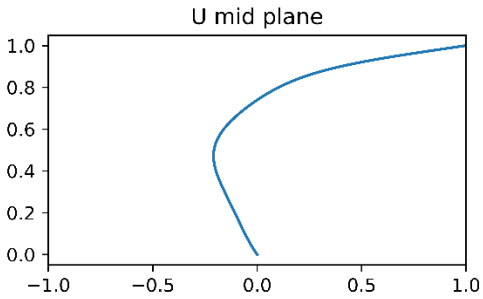
31x31 grid, Re=400

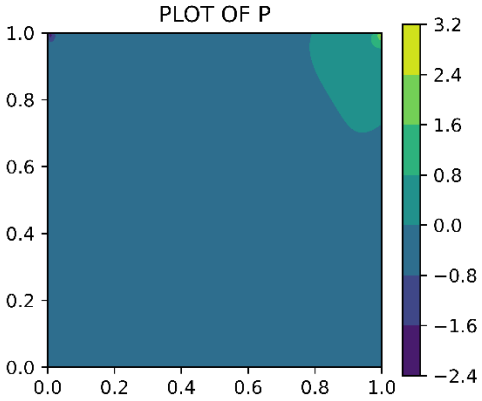


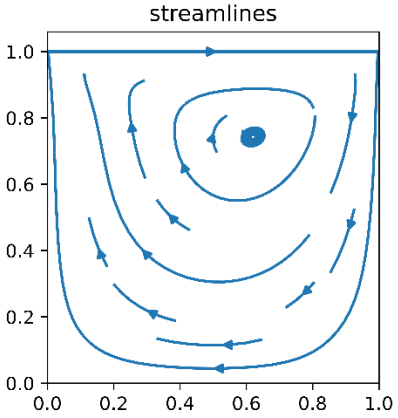
129x129 grid, Re=100

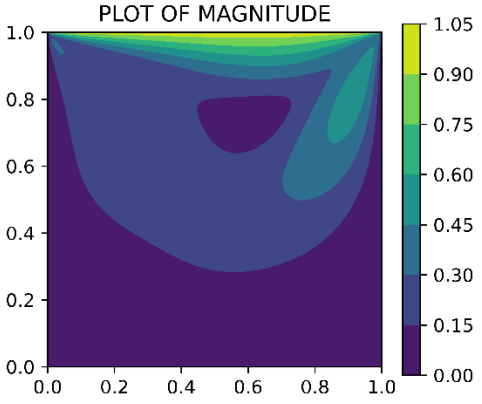




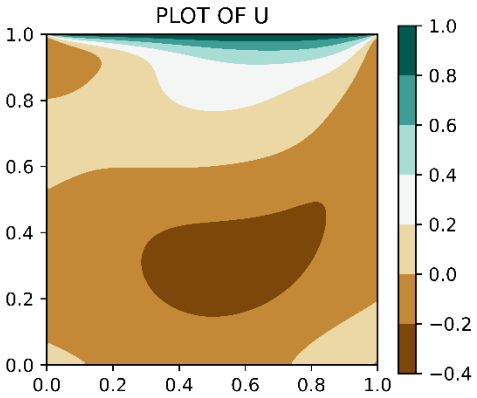


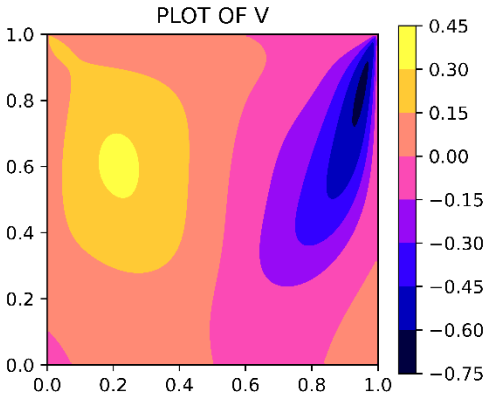


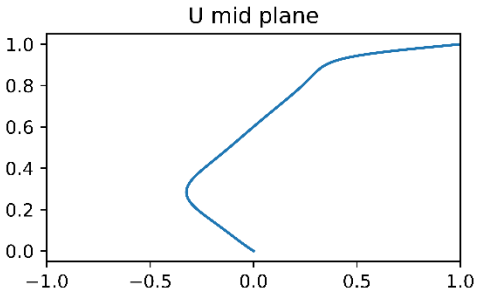


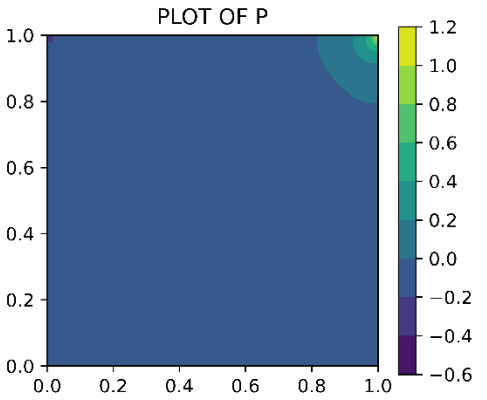


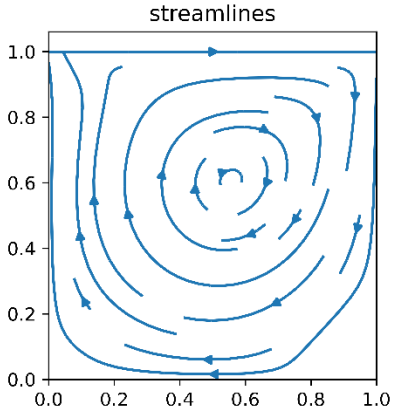
129x129 grid, Re=400

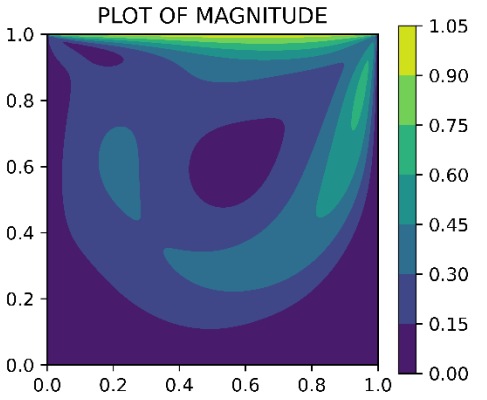




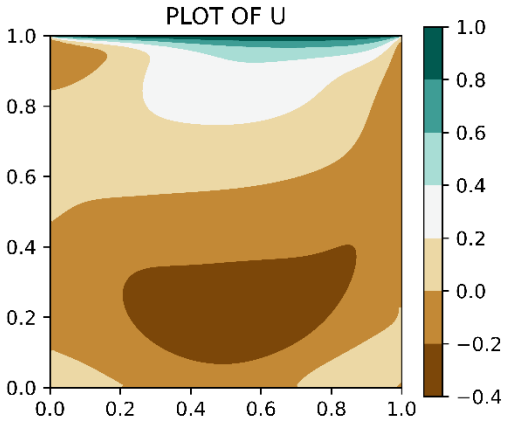


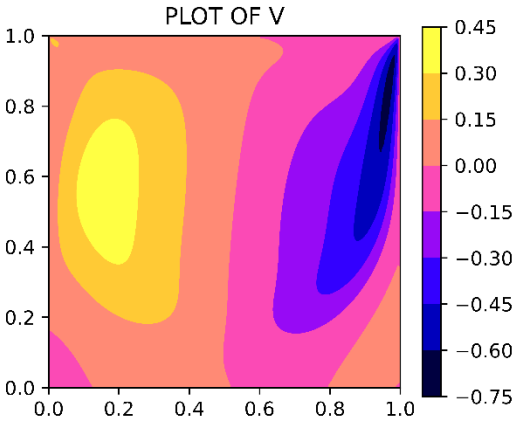


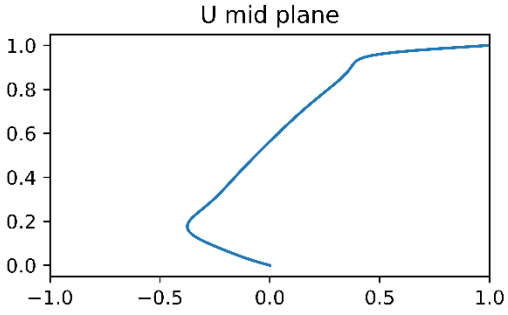


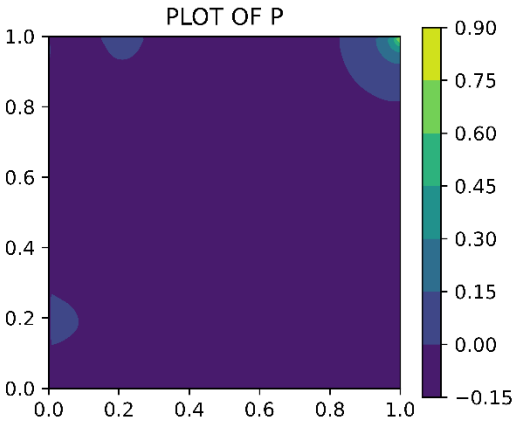


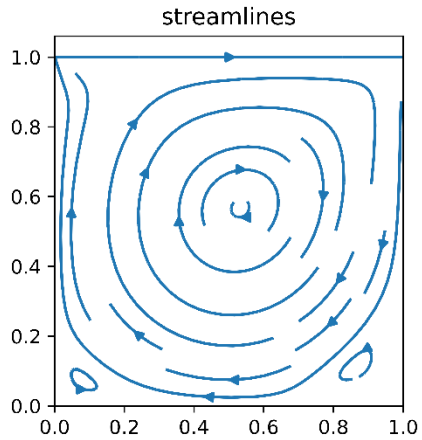
129x129 grid, Re=1000

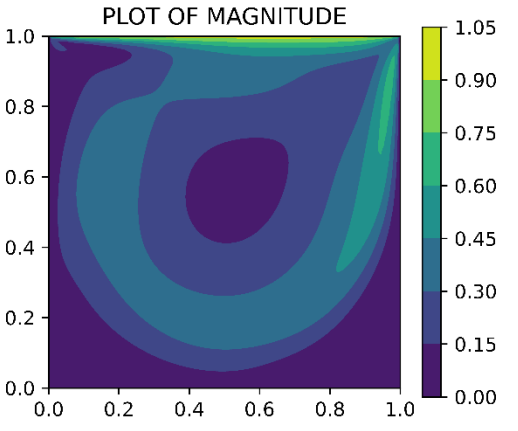




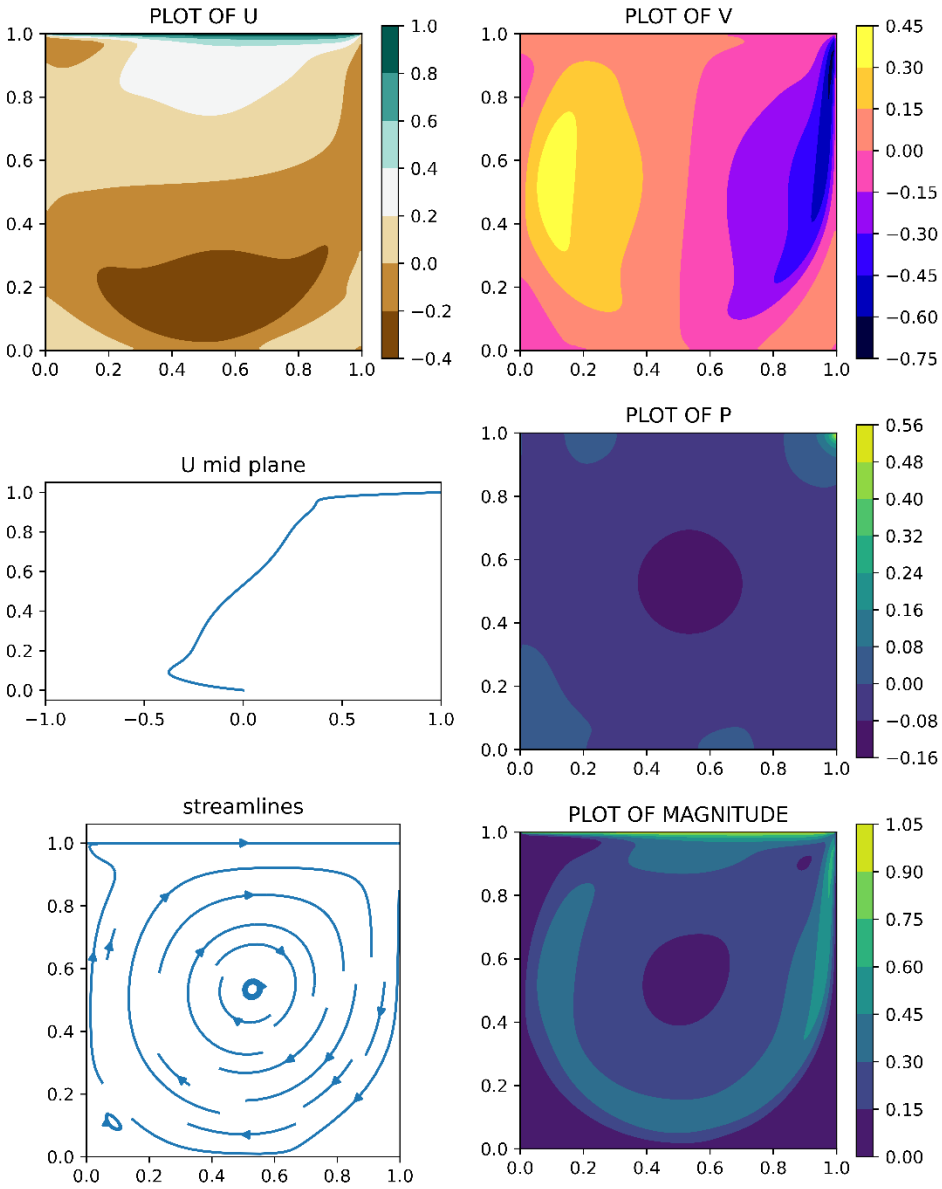








129x129 grid, Re=3200



CONLCUSION

Despite many setbacks regarding understand the ghost cell implementation off-paper, the results show that the MAC method developed here is valid when compared to the previous literature's results. They also show the strength of the MAC method in computing 2D flows in a confined space, given the governing Navier-Stokes equations. Conversely, given the difficulties early on, it showed to me the complications arising in the method itself, particularly the ghost cell conditions, which become even harder to implement when a non-square grid is used. They can be hard to visualize. If I were to do this project again, with a different cavity type, I would like to try to use the u* and v* intermediate velocity formulations, as it is more frequently referenced in the literature. Another future wish would be to implement this exact problem but in 3D, as a first step into the 3D flow regimes.

REFERENCES

D. L. Brown, R. Cortez, and M. L. Minion, "Accurate projection methods for the incompressible Navier-Stokes Equations", Journal of Computational Physics, vol. 168, 464–499 (2001)

Ghia H., Ghia, K. N., and Shin C. T., "High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method", Journal of Computational Physics, vol. 48, 387–411 (1982)

Kim, J., and Moin, P., "Application of a Fractional-Step Method to Incompressible Navier-Stokes Equations", Journal of Computational Physics, vol. 59, 308–323 (1985)

https://math.mit.edu/~stoopn/18.086/Lecture12.pdf