Analysis of 2-D Lid-Driven Cavity Problem

Aakash Yadav
Email: me16b001@iittp.ac.in

Indian Institute of Technology Tirupati

April 2019

Abstract

The lid driven cavity problem is a very standard problem in domain of Computational Fluid Dynamics and is being used as a benchmark problem in CFD. The problem although appears to be simple , holds an ample opportunity for one to explore and has endless strings attached to it. The ultimate goal of this paper is to gain insight of some of the physics that is powering the modern CFD packages. In the current work, we present our analysis for both the steady and unsteady states. We have used the derived variable approach for carrying out the analysis. For the steady state solutions, uniform grid has been used while for the unsteady state we have inplemented the non-uniform grid in order to capture the dynamics of the problem more accurately where the velocity gradients are very high. Accurate results inline with results obtained by Ghia et al [6] have been obtained. This paper also presents the methodolgy for Von-Neumann stability analysis in great depth.

1 Introduction

This problem has been solved by large numbers of people in the past by using different methodologies and schemes starting from 1966 by Burggraf [7], yet it is an active area of research even in the present. Tamer et al [10] provides a summary of all the work that has been done from 1966 till 2014 in very concise tabular format in chronological order. The problem have been simulated upto Reynolds number, $Re = 15000$ by Bruneau et al [3] in their 1990s paper by using a finite difference approach. It can be observed that Finite Difference scheme has been heavily adopted in the past for carrying out the analysis. In present work we have used the Finite Difference scheme for the unsteady state analysis while upwind scheme [5] (both first and second order) for the steady state analysis.

In this work, we have analyzed the unsteady, viscous, incompressible, isothermal, two-dimensional, laminar flow of Newtonian fluid taking place in the cavity, driven by lid of infinite length (Figure 1). This problem is solved using an in-house Python code developed by the author (https://github.com/AakashSYadav/LidDrivenCavityProblem), which solves both unsteady and steady Navier-Stokes equations using derived variable apporach, using the Finite Difference Method (FDM).

The geometery of the problem is quite simple, it consists of a rectangular cavity ABCD in the $x - y$ plane with dimensions as shown in the Figure 1, where the walls AB, CD, and AD are rigid walls, whereas BC is open. The cavity is of length, D in the x-direction and height, H in the y direction. The aspect ratio is defined as $r = H/D$. The Reynolds number is defined based on the velocity scale, U and the length scale, D. Acceleration due to gravity acts in the negative z direction. The top of the cavity, BC, is covered with an infinitely-long rigid lid. Initially ($t \leq 0$), the fluid inside the cavity is at rest. At time $t > 0$, the lid is set in motion to the right with a constant velocity U.

In section 2, we bring equtions governing the fluid flow in the appropriate form, the section that follows provides insights to boundary and initial conditions that are exisitng for this problem, and which can be used in solving the problem numerically. In section 4 and 5 we non-dimensionalise the governing equations and carry out their discretization in the later. Section 6, is the most important part of this paper which gives insight of the algorithm(s) used in this work. In the final section we present the conclusions. All the algorithms implemented in this work can be found in the appendix.

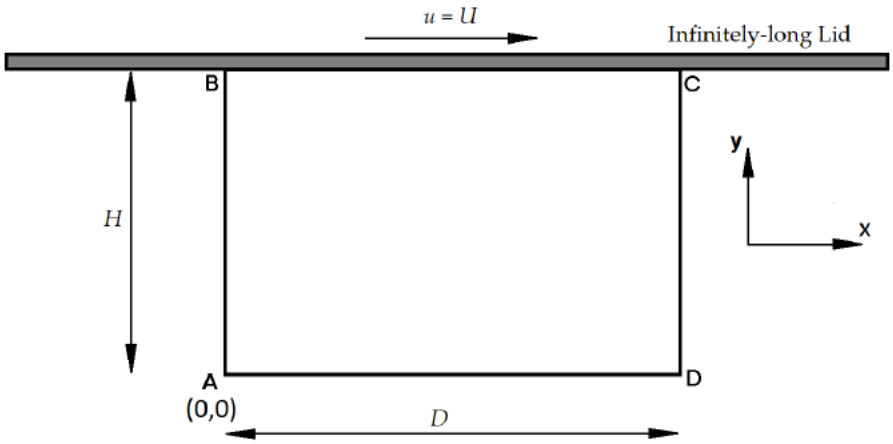

Figure 1: Lid driven cavity in $x-y$ plane

2 Governing Equations

The equation of continuity for 2-D incompressible isothermal flow [8, 2] is given by the equation

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0 \tag{1}$$

where u and v are the components of veloity in x and y directions respectively. Furthermore we have the stream function, ψ defined as

$$u=\frac{d\psi}{dy} \tag{2a}$$

$$v=-\frac{d\psi}{dx} \tag{2b}$$

The vorticity equation of fluid dynamics describes evolution of the vorticity ω of fluid as it moves with its flow, i.e. the local rotation of the fluid. Mathematically vorticity is the curl of the flow velocity.

$$\vec{\omega}=\vec{\nabla}\times\vec{V} \tag{3a}$$

$$\vec{\omega}=(v_x-u_y)\hat{k} \tag{3b}$$

$$\vec{\omega}=-\left[\frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2}\right]=-\nabla^2\psi \tag{3c}$$

Equation 3c is a elliptic equation and can be obtained by combining equations 2a, 2b and 3b.

Navier-Stokes equations which follows are perhaps the most important equations in the field of Fluid Dynamics. These equations are essentially the momentum conservation equations for isothermal, Newtonian and incopressible fluid. This can also be seen as application of Isaac Newton's second law to fluid motion.

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}=-\frac{1}{\rho}\frac{\partial P}{\partial x}+g_x+\nu\left[\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right] \tag{4a}$$

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}=-\frac{1}{\rho}\frac{\partial P}{\partial y}+g_y+\nu\left[\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right] \tag{4b}$$

Equations 4a and 4b are for the fluid flow along the x and y direction respectively. One may observe that these equations also comprises of the pressure term P, which may not be required in some of the cases. We can eliminate the pressure term P in order to reduce the number of variable and simplify the anylysis by computing $\frac{\partial(\ 4a)}{\partial y}-\frac{\partial(\ 4b)}{\partial x}$, thus resulting in the below equation know as vorticity relation. This technique of approaching the problem is known as derived variable approach.

$$\frac{\partial \omega_z}{\partial t}+u\frac{\partial \omega_z}{\partial x}+v\frac{\partial \omega_z}{\partial y}=\nu\left[\frac{\partial^2 \omega_z}{\partial x^2}+\frac{\partial^2 \omega_z}{\partial y^2}\right] \tag{5}$$

The equation thus obtained is parabolic in nature. For the sake of convienance we will drop the subscript from the ω_z hereafter.

3 Initial Conditions and Boundary Conditions

At time $t = 0$ everything is at rest and hence all the velocities, both of the fluid and the plate are zero. At the moment $t \geq 0$ the lid will be moving at speed $u = U$, which will try to set the fluid in motion. We have the following boundary conditions for $t \geq 0$:
No slip condition will result in zero velocity along the wall tangent yields

$$u(x, 0) = 0 \ldots \text{ Bottom wall} \tag{6a}$$

$$v(D, y) = 0 \ldots \text{ Right wall} \tag{6b}$$

$$u(x, H) = U \ldots \text{ Top wall} \tag{6c}$$

$$u(0, y) = 0 \ldots \text{ Left wall} \tag{6d}$$

While the no penetration condition that the walls of the cavity are impervious and impenetrable results into

$$v(x, 0) = 0 \ldots \text{ Bottom wall} \tag{7a}$$

$$u(D, y) = 0 \ldots \text{ Right wall} \tag{7b}$$

$$v(x, H) = 0 \ldots \text{ Top wall} \tag{7c}$$

$$u(0, y) = 0 \ldots \text{ Left wall} \tag{7d}$$

More discussion of the boundary conditions will be followed in preceding section after their non-dimensionalization.

4 Non-dimensionalization of the Governing Equations

In this section, we nondimensionalize the flow governing equations obtained in section 2. This eases the analysis of the problem which, and reduces the number of free parameters, furthermore it helps to gain a greater insight into the relative size of various terms present in the equation. For carrying out non-dimensionalization we need appropriate scalings for the dimensionless variables, the scales are chosen for different quantites as shown below-

$$\hat{x} = \frac{x}{D}, \hat{y} = \frac{y}{H}, \hat{t} = \frac{U}{D}t, \hat{u} = \frac{u}{U}, \hat{v} = \frac{v}{V_{ref}}, \hat{\psi} = \frac{\psi}{UD}, \hat{\omega} = \frac{\omega D}{U} \tag{8}$$

We can obtain the scaling factor for the velocity in y direction v by carrying out non-dimensionalization of equation 1 which results into

$$\frac{U}{D}\frac{\partial \hat{u}}{\partial \hat{x}} + \frac{V_{ref}}{H}\frac{\partial \hat{v}}{\partial \hat{y}} = 0 \tag{9a}$$

$$\frac{U}{D} \sim \frac{V_{ref}}{H} \tag{9b}$$

$$V_{ref} = \left(\frac{H}{D}\right) U = RU \tag{9c}$$

Non-dimensionalising the stream function equation (3c)

$$\frac{\hat{\omega} U}{D} = -\left[\frac{\partial^2(\hat{\psi} U D)}{\partial(\hat{x}D)^2} + \frac{\partial^2(\psi \hat{U} D)}{\partial(\hat{y}H)^2}\right] \tag{10a}$$

$$\hat{\omega} = -\left[\frac{\partial^2\hat{\psi}}{\partial\hat{x}^2} + \frac{1}{r^2}\frac{\partial^2\hat{\psi}}{\partial\hat{y}^2}\right] \tag{10b}$$

Non-dimensionalising the vorticity function equation (5)

$$\frac{\partial\left(\frac{U}{D}\hat{\omega_z}\right)}{\partial\left(\frac{D}{U}\hat{t}\right)} + \hat{u}U\frac{\partial\left(\frac{U}{D}\hat{\omega_z}\right)}{\partial(\hat{x}D)} + \frac{UH}{D}\hat{v}\frac{\partial\left(\frac{U}{D}\hat{\omega_z}\right)}{\partial(\hat{y}H)} = \nu\left[\frac{\partial^2\left(\frac{U}{D}\hat{\omega_z}\right)}{\partial(\hat{x}D)^2} + \frac{\partial^2\left(\frac{U}{D}\hat{\omega_z}\right)}{\partial(\hat{y}H)^2}\right] \tag{11a}$$

$$\frac{\partial\hat{\omega_z}}{\partial\hat{t}} + \hat{u}\frac{\partial\hat{\omega_z}}{\partial\hat{x}} + \hat{v}\frac{\partial\hat{\omega_z}}{\partial\hat{y}} = \frac{\nu}{UD}\left[\frac{\partial^2\hat{\omega_z}}{\partial\hat{x}^2} + \frac{1}{r^2}\frac{\partial^2\hat{\omega_z}}{\partial\hat{y}^2}\right] \tag{11b}$$

$$\frac{\partial\hat{\omega_z}}{\partial\hat{t}} + \hat{u}\frac{\partial\hat{\omega_z}}{\partial\hat{x}} + \hat{v}\frac{\partial\hat{\omega_z}}{\partial\hat{y}} = \frac{1}{Re}\left[\frac{\partial^2\hat{\omega_z}}{\partial\hat{x}^2} + \frac{1}{r^2}\frac{\partial^2\hat{\omega_z}}{\partial\hat{y}^2}\right] \tag{11c}$$

Where, $Re = \frac{UD}{\nu}$, also we have

$$u = \frac{\partial\psi}{\partial y}, v = -\frac{\partial\psi}{\partial x}$$

Non-dimensionalisation of the above equation results into

$$\hat{u} = \frac{1}{r}\frac{\partial\hat{\psi}}{\partial\hat{y}}, \hat{v} = -\frac{1}{r}\frac{\partial\hat{\psi}}{\partial\hat{x}} \tag{12}$$

Substituting equation (12) in equation (11c) yields

$$\frac{\partial\hat{\omega_z}}{\partial\hat{t}} + \frac{1}{r}\left[\frac{\partial\hat{\psi}}{\partial\hat{y}}\frac{\partial\hat{\omega_z}}{\partial\hat{x}} - \frac{\partial\hat{\psi}}{\partial\hat{x}}\frac{\partial\hat{\omega_z}}{\partial\hat{y}}\right] = \frac{1}{Re}\left[\frac{\partial^2\hat{\omega_z}}{\partial\hat{x}^2} + \frac{1}{r^2}\frac{\partial^2\hat{\omega_z}}{\partial\hat{y}^2}\right] \tag{13}$$

Deriving the boundary conditions in the form of vorticity and stream function

$$\hat{\omega}(\hat{x}, 0) = -\frac{1}{r^2}\left(\frac{\partial^2\hat{\psi}}{\partial\hat{y}^2}\right)_{\hat{y}=0} \dots \text{Bottom wall} \tag{14a}$$

$$\hat{\omega}(\hat{x}, 1) = -\frac{1}{r^2}\left(\frac{\partial^2\hat{\psi}}{\partial\hat{y}^2}\right)_{\hat{y}=1} \dots \text{Top wall} \tag{14b}$$

$$\hat{\omega}(0, \hat{y}) = -\left(\frac{\partial^2\hat{\psi}}{\partial\hat{x}^2}\right)_{\hat{x}=0} \dots \text{Left wall} \tag{14c}$$

$$\hat{\omega}(1, \hat{y}) = -\left(\frac{\partial^2\hat{\psi}}{\partial\hat{x}^2}\right)_{\hat{x}=1} \dots \text{Right wall} \tag{14d}$$

We can expand ψ using the Taylor series expansion as

$$\psi_{2,j} = \psi_{1,j} + \frac{\partial\psi}{\partial x}_{1,j}\Delta x + \frac{\partial^2\psi}{\partial x^2}_{1,j}\frac{\Delta x}{2!}\dots \tag{15a}$$

$$\frac{\partial^2\psi}{\partial x^2}_{1,j} = \frac{2(\psi_{2,j} - \psi_{1,j})}{\partial x^2} + \frac{2v_{1,j}}{\Delta x} + \mathcal{O}(\Delta x) \tag{15b}$$

Similarly we can obtain the expressions as above for the other walls [1].

For the bottom surface $\hat{u} = \hat{v} = 0$, substituiting this equation (12) yields

$$\hat{\psi}_{(\hat{y}=0)} = c_1 \tag{16}$$

Where c_1 is the constant of integration. Similarly we can obtain the equations for the other walls as

$$\hat{\psi}_{(\hat{x}=0)} = c_2, \hat{\psi}_{(\hat{x}=1)} = c_3 \tag{17}$$

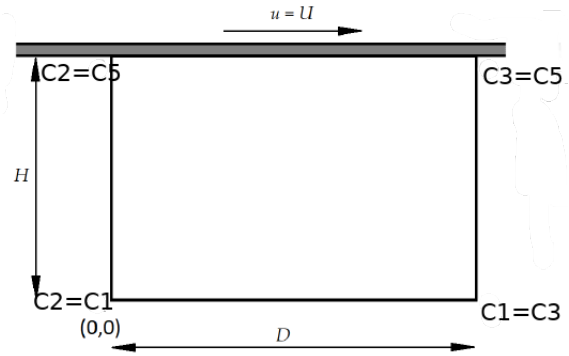

Figure 2: Continuity of the integration contants at the vertices

For the top wall we have

$$\hat{u} = U = \frac{1}{r}\frac{\partial \hat{\psi}}{\partial \hat{y}} \Rightarrow \hat{\psi} = rUH + c_4 = c_5 \tag{18}$$

$$\hat{\psi}_{(\hat{y}=1)} = c_5 \tag{19}$$

We can choose the constants to be equal to each other i.e. $c_1 = c_2 = c_3 = c_5 = c$ because of the continuity at the corners as shown in the Figure 2. Without any loss of generality we can set $c = 0$ as ψ is a relative term. Hence

$$\hat{\psi}_{(y=0)} = \hat{\psi}_{(y=1)} = \hat{\psi}_{(x=0)} = \hat{\psi}_{(x=1)} = 0 \tag{20}$$

5 Discretization Schemes

In this section we focus on discretisation of the domain in smaller regions. This is inherently a two step process, first we meshing i.e. divide the domain into smaller regions. These smaller regions may be triangles and rectangles (in 2D) and tetrahedrons, hexahedrons (in 3D) and other types of geometric entities, but in our case we stick to the rectangular grids. The vertices of these geometric entities are called nodes. Secondly, we discretize the governing equations over the predefined mesh. In the analysis that follows we will drop the 'hat' notation for dimensionless variables for convienance. We present this process in to different subsections one each for steady (uniform grid) and unsteady state (non-uniform grid).

5.1 Discretization for Non-uniform grid

We have used the Forward Time Central Space (FTCS) scheme for the non-uniform grid. The main reason for using the non-uniform grid is enable our analysis to capture accurately the dynamics near the walls where the velocity gradients are very high. The grid size is relatively bigger near the center in order to save some computaion time, also close to the center ther wont be any drastic changes. The non-uniform grid that is not equally spaced as shown in Figure 3. The grid has been obtained by using *tanh* function, the grid stretching can be changed as required by supplying the stretching parameter γ to the grid stretching function (see Appendix for implementaion). Function used for grid generation -

$$y_j = 1 - \frac{tanh\left[\gamma\left(1 - \frac{2j}{N}\right)\right]}{tanh(\gamma)} \quad j = 1, 2, 3 \ldots N \tag{21}$$

Now we derive the FTCS scheme for non-uniform grid. We use Taylor series expansions to develop and/or analyse the accuracy of numerical approximations for derivatives. Using Taylor series expansions for the grid point as shown in Figure 4 we can write

$$f(x_{i+1}) = f(x_i) + \Delta x_i f^{'}(x_i) + \frac{(\Delta x_i)^2}{2!} f^{''}(x_i) + \frac{(\Delta x_i)^3}{3!} f^{'''}(x_i) \ldots \tag{22a}$$

$$f(x_{i-1}) = f(x_i) - \Delta x_{i-1} f^{'}(x_i) + \frac{(\Delta x_{i-1})^2}{2!} f^{''}(x_i) - \frac{(\Delta x_{i-1})^3}{3!} f^{'''}(x_i) \ldots \tag{22b}$$

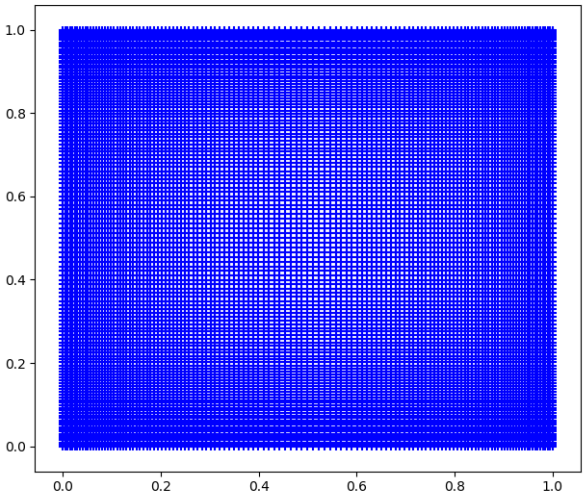

Figure 3: A 128 × 128 Non-uniform grid generated by the program

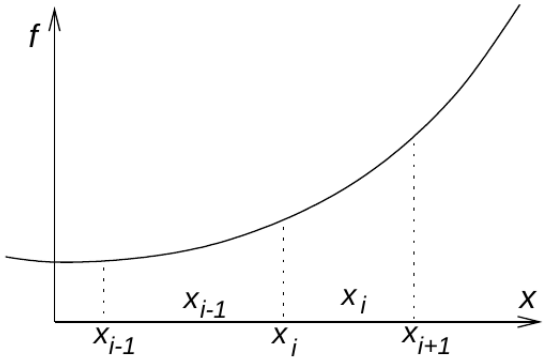

Figure 4: Non-unform spacing along the $x-$axis (Credit- Craft)

Using the above to equation we can discretized derivatives to be

$$f^{'}(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{\Delta x_i + \Delta x_{i-1}} + \mathcal{O}(\Delta x) \tag{23a}$$

$$f^{''}(x_i) = \frac{\Delta x_{i-1} f(x_{i+1}) + \Delta x_i f(x_{i-1}) - (\Delta x_i + \Delta x_{i-1} f(x_i))}{\Delta x_{i-1}\Delta x_i + \Delta x_i \Delta x_{i-1}} + \mathcal{O}(\Delta x) \tag{23b}$$

Although, the equations appear to be only first order accurate for the non-uniform grid, they approach second order accuracy when the grid size are very close to each other i.e. the case of uniform grid. We can write the discretised form for the equation 10b and 11c as

$$\begin{aligned} -\omega_{ij} = 2 &\left[\frac{\Delta x_{i-1}\psi(x_{i+1}) + \Delta x_i \psi(x_{i-1}) - (\Delta x_i + \Delta x_{i-1}\psi(x_i))}{\Delta x_{i-1}\Delta x_i + \Delta x_i \Delta x_{i-1}}\right] \\ &+ \frac{2}{r^2}\left[\frac{\Delta y_{i-1}\psi(y_{i+1}) + \Delta y_i \psi(y_{i-1}) - (\Delta y_i + \Delta y_{i-1}\psi(y_i))}{\Delta y_{i-1}\Delta y_i + \Delta y_i \Delta y_{i-1}}\right] \end{aligned} \tag{24a}$$

$$\frac{\omega^{n+1} - \omega^n}{\Delta t_i} + \left[u_{ij}\left(\frac{\omega(x_{i+1}) - \omega(x_{i-1})}{\Delta x_i + \Delta x_{i-1}}\right) + v_{i,j}\left(\frac{\omega(y_{i+1}) - \omega(y_{i-1})}{\Delta y_i + \Delta y_{i-1}}\right)\right] = \frac{1}{Re} A \tag{24b}$$

Where A, is defined as

$$A = 2\left[\frac{\Delta x_{i-1}\omega(x_{i+1}) + \Delta x_i\omega(x_{i-1}) - (\Delta x_i + \Delta x_{i-1}\omega(x_i))}{\Delta x_{i-1}\Delta x_i + \Delta x_i\Delta x_{i-1}}\right] + \frac{2}{r^2}\left[\frac{\Delta y_{i-1}\omega(y_{i+1}) + \Delta y_i\omega(y_{i-1}) - (\Delta y_i + \Delta y_{i-1}\omega(y_i))}{\Delta y_{i-1}\Delta y_i + \Delta y_i\Delta y_{i-1}}\right]$$

5.2 Discretization for Uniform grid

In this section we discretize equation for uniform grid to be used in the steady state analysis using the upwind scheme. We discretize equation 10b using central differencing scheme, the resulting equation in second order accurate in space.

$$\frac{\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}}{(\Delta x)^2} + \frac{1}{r^2}\frac{\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}}{(\Delta y)^2} = -\omega_{zij} \tag{25}$$

For discretization of equation 11c we drop the temporal derivative terms and discretize using upwind scheme, we have used first order accurate upwind scheme for the nodes one step inside the boundary while second order accurate for all the interior nodes. Moreover from literature it is known that upwind scheme has been more accurate than the downwind scheme. [9]

$$u\frac{\partial\omega_z}{\partial x} + v\frac{\partial\omega_z}{\partial y} = \frac{1}{Re}\left[\frac{\partial^2\omega_z}{\partial x^2} + \frac{1}{r^2}\frac{\partial^2\omega_z}{\partial y^2}\right] \tag{26}$$

$$\left[u_{ij}\frac{\partial\omega_z}{\partial x} + v_{ij}\frac{\partial\omega_z}{\partial y}\right] = \frac{1}{Re}\left[\left(\frac{\omega_{i+1,j} - 2\omega_{i,j} + \omega_{i-1,j}}{(\Delta x)^2}\right) + \frac{1}{r^2}\left(\frac{\omega_{i,j+1} - 2\omega_{i,j} + \omega_{i,j-1}}{(\Delta y)^2}\right)\right] \tag{27}$$

For first order accurate upwind scheme we have

$$\begin{aligned} u_{ij}\frac{\partial\omega}{\partial x}_{ij} &= u_{ij}\left[\frac{\omega_{i,j} - \omega_{i-1,j}}{\Delta x}\right] \text{ if } u_{ij} > 0 \\ &= u_{ij}\left[\frac{\omega_{i+1,j} - \omega_{i,j}}{\Delta x}\right] \text{ if } u_{ij} < 0 \end{aligned} \tag{28a}$$

$$\begin{aligned} v_{ij}\frac{\partial\omega}{\partial y}_{ij} &= v_{ij}\left[\frac{\omega_{i,j} - \omega_{i,j-1}}{\Delta y}\right] \text{ if } v_{ij} > 0 \\ &= v_{ij}\left[\frac{\omega_{i,j+1} - \omega_{i,j}}{\Delta y}\right] \text{ if } v_{ij} < 0 \end{aligned} \tag{28b}$$

For second order accurate upwind scheme for all the interior nodes (except two outer layers) we have

$$\begin{aligned} u_{ij}\frac{\partial\omega}{\partial x}_{ij} &= u_{ij}\left[\frac{3\omega_{i,j} - 4\omega_{i-1,j} + \omega_{i-2,j}}{2\Delta x}\right] \text{ if } u_{ij} > 0 \\ &= u_{ij}\left[\frac{-3\omega_{i,j} + 4\omega_{i+1,j} - \omega_{i+2,j}}{2\Delta x}\right] \text{ if } u_{ij} < 0 \end{aligned} \tag{29a}$$

$$\begin{aligned} v_{ij}\frac{\partial\omega}{\partial y}_{ij} &= v_{ij}\left[\frac{3\omega_{i,j} - 4\omega_{i,j-1} + \omega_{i,j-2}}{2\Delta y}\right] \text{ if } v_{ij} > 0 \\ &= v_{ij}\left[\frac{-3\omega_{i,j} + 4\omega_{i,j+1} - \omega_{i,j+2}}{2\Delta y}\right] \text{ if } v_{ij} < 0 \end{aligned} \tag{29b}$$

6 Courant–Friedrichs–Lewy (CFL) analysis

This section deals with the stabilty analysis, it is a necessary condition for convergence while solving PDEs numerically. It plays an important role explicit time integration schemes as a consequence, the time

step must be less than a certain critical time step, otherwise the simulation blow away. The condition is named after Richard Courant, Kurt Friedrichs, and Hans Lewy who described it in their 1928 paper [4]. In our case the values of Δx, Δy, u and v are varying throughout the domain. Hence for finding the solution at $t+\Delta t$ we need to compute Δt such that our computations are stable at each and every node, thus we need to find the grid Courant number is also going to vary for every node, and every time step.

We have the non dimentionalized streamline vorticity formulation equation as

$$\frac{\partial\hat{\omega}}{\partial\hat{t}} + \hat{u}\frac{\partial\hat{\omega}}{\partial\hat{x}} + \hat{v}\frac{\partial\hat{\omega}}{\partial\hat{y}} = \frac{1}{Re}\left[\frac{\partial^2\hat{\omega}}{\partial\hat{x}^2} + \frac{1}{r^2}\frac{\partial^2\hat{\omega}}{\partial\hat{y}^2}\right] \tag{30}$$

In the analysis that follows we have dropped the hat symbol i.e. $\hat{}$ for non-dimensionalised quantities. We shall now use the Fourier wave form to bring out the required conditions. The amplification is easily identified using the Fourier wave form, moreover the waveform remains preserved. Also only one term is sufficient because of the linearity of the equation (Although this equation is not linear the analysis can provide apporximate guesses).

$$\omega = \sum_m A^n(t)e^{\hat{i}(k_m x + k_n y)} \tag{31}$$

Here we can assume the wavenumber of the wave in both x and y direction to be same i.e. $k_n = k_m$

$$\omega = A^n e^{\hat{i}(ik_m\Delta x + jk_n\Delta y)} \tag{32}$$

$$\omega_{i,j}^n = A^n e^{\hat{i}(ik_m\Delta x + jk_n\Delta y)} \tag{33a}$$

$$\omega_{i,j}^{n+1} = A^{n+1} e^{\hat{i}(ik_m\Delta x + jk_n\Delta y)} \tag{33b}$$

$$\omega_{i+1,j}^n = A^n e^{\hat{i}((i+1)k_m\Delta x + jk_n\Delta y)} \tag{33c}$$

$$\omega_{i-1,j}^n = A^n e^{\hat{i}((i-1)k_m\Delta x + jk_n\Delta y)} \tag{33d}$$

$$\omega_{i,j+1}^n = A^n e^{\hat{i}(ik_m\Delta x + (j+1)k_n\Delta y)} \tag{33e}$$

$$\omega_{i,j-1}^n = A^n e^{\hat{i}(ik_m\Delta x + (j-1)k_n\Delta y)} \tag{33f}$$

Substituting this in our main equation after its discretization and simplification results in

$$\frac{\frac{A^{n+1}}{A^n} - 1}{\Delta t} + u\left(\frac{e^{ik_m\Delta x} - e^{-ik_m\Delta x}}{2\Delta x}\right) + v\left(\frac{e^{ik_n\Delta y} - e^{-ik_n\Delta y}}{2\Delta y}\right) = \frac{1}{Re}\left[\left(\frac{e^{ik_m\Delta x} - 2 + e^{-ik_m\Delta x}}{\Delta x^2}\right) + \frac{1}{r^2}\left(\frac{e^{ik_n\Delta y} - 2 + e^{-ik_n\Delta y}}{\Delta y^2}\right)\right]$$

Let $\theta_1 = k_m\Delta x$ and $\theta_2 = k_n\Delta y$, in order to simplyfy the analysis we can choose the k_i's such that $\theta_1 = \theta_2 = \theta$. We can use the below identity on the obtained equation.

$$e^{i\theta} = cos\theta + isin\theta \tag{34}$$

Hence we can obtain the amplification factor G as

$$G = \frac{A^{n+1}}{A^n} = 1 - \frac{1}{Re}\left(\frac{4\Delta t}{\Delta x^2} + \frac{4\Delta t}{\Delta y^2}\right)\left(\frac{1-cos\theta}{2}\right) - i\left(\frac{u\Delta t}{\Delta x} + \frac{v\Delta t}{\Delta y}\right)sin\theta \tag{35}$$

This equation can be re-written in terms of α and β as

$$G = 1 - \alpha\left(\frac{1-cos\theta}{2}\right) + i\beta sin\theta \tag{36}$$

$$|G|^2 = 1 + \frac{\alpha^2}{4}(1-q)^2 + \beta^2(1-q^2) - \alpha(1-q) \tag{37}$$

where,

$$q = cos\theta$$

$$\alpha = \frac{1}{Re}\left[\frac{4\Delta t}{\Delta x^2} + \frac{4\Delta t}{\Delta y^2}\right]$$

$$\beta = \left[\frac{u\Delta t}{\Delta x} + \frac{v\Delta t}{\Delta y}\right]$$

For the scheme to be stable, amplification factor $|G| \leq 1$ or $|G|^2 \leq 1$. We define a polinomial in terms of q as

$$p(q) = \frac{\alpha^2}{4}(1-q)^2 + \beta^2(1-q^2) - \alpha(1-q) \leq 0 \tag{38}$$

$$p(q) = \frac{\alpha^2}{4}(1-q)^2 + \beta^2(1-q^2) - \alpha(1-q) \leq 0 \tag{39}$$

As $q = 1$ is a root of $p(q)$, we can have four types of possible parabolas as shown in Figure 5. But as $p(q) \leq 0$ the parabolas shown with dotted red line can be rejected. Hence leaving the parabolas in blue, having positive slope i.e. $p^{'}(q) > 0$ at $q = 1$. Thus at $q = 1$ we have $p^{'}(1) > 0$ or $p^{'}(1) = \alpha - 2\beta^2 > 0$ which results in the condition

$$\alpha > 2\beta^2 \tag{40}$$

Moreover, we also have the condition that $p(-1) \leq 0$ which results in $\alpha \leq 2$. Hence we have $2\beta^2 < \alpha \leq 2$. Finally, we obtain two conditions for the time step, we shall us the minimum of the two.

$$\Delta t \leq \frac{Re}{2}\left[\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2 r^2}\right]^{-1} \tag{41}$$

$$\Delta t \leq \left[\frac{u}{\Delta x} + \frac{v}{\Delta y}\right]^{-1} \tag{42}$$

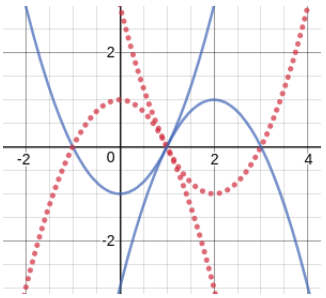

Figure 5: Possible parabolas passing through $q = 1$

7 Algorithm

In this section we provide the algorithms which have been used for solving this problem. Table 1 lists the algorithm flow for unsteady state analysis while table 2 provides the algorithm for steady state analysis.

8 Results

In this final section we present the results, for both steady and unsteady analysis. The stream function plots for unsteady flow has been given in 6

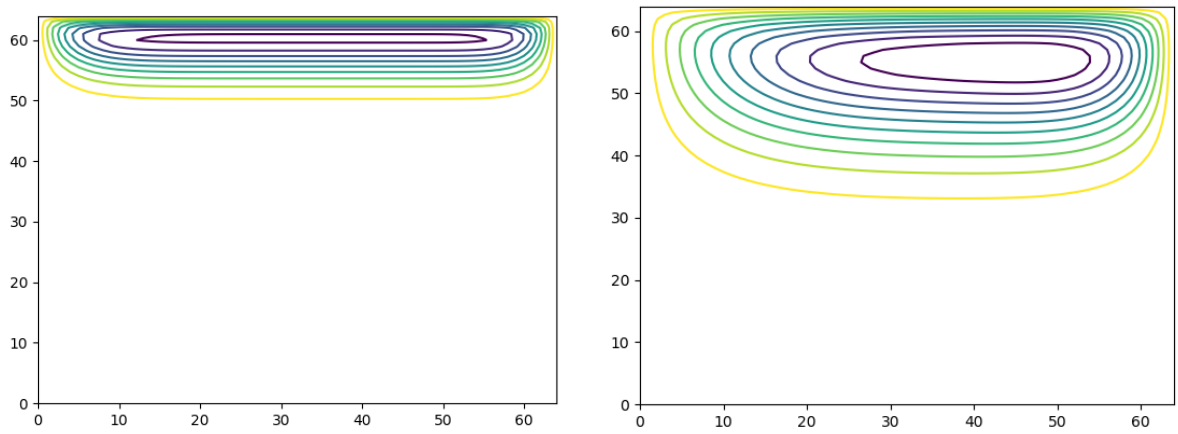

Figure 6: Time evolution of Streamlines, unsteady state for 64×64 non-uniform grid, $Re = 10$

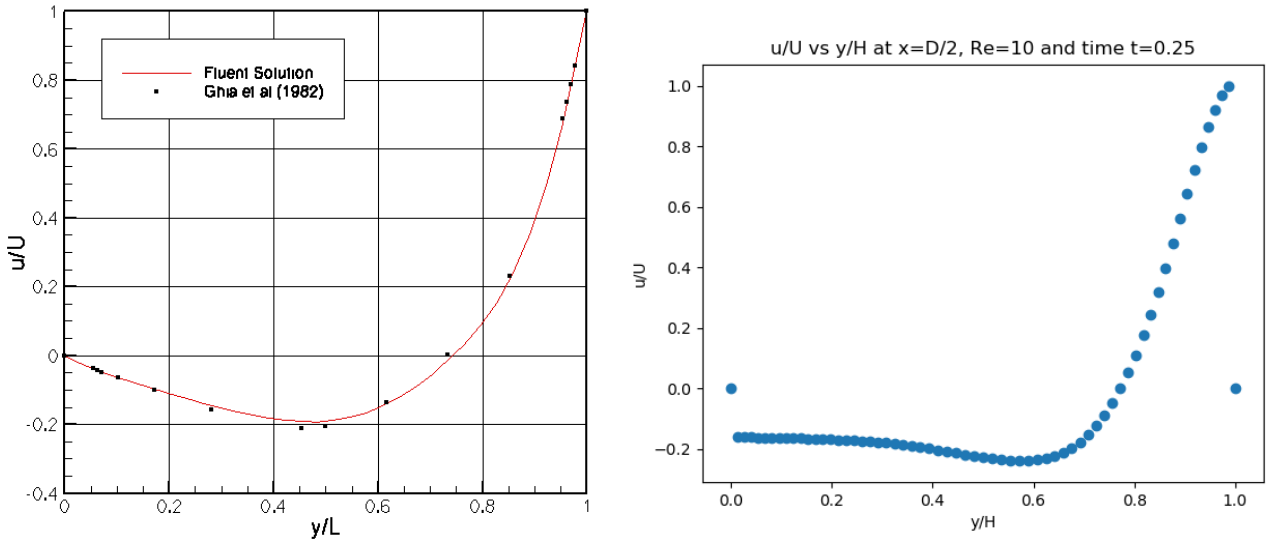

Figure 7: Comparitive plots between obtained plot and standard plot by Ghia et al

No.	Steps
1	Initialize u, v, ψ and ω matrices
2	Apply boundary conditions (Section 3,4) for u, v, ψ and ω
3	Use CFL criteria at every node and find the stable Δt for next time step
4	Solve vorticity transport equation at a forward time step $t + \Delta t$ and find ω^{n+1}
5	Solve the Poisson equation for stream function (ψ^{n+1}) at $t + \Delta t$ by iterative method
6	Find u and v using the iterated streamfunciton (ψ^{n+1})
7	Repeat the loop till the specified time or steady state is reached

Table 1: Algorithm for Unsteady state analysis

No.	Steps
1	Initialize u, v, ψ and ω matrices
2	Apply boundary conditions for u, v, ψ and ω
3	Start the iteration loop
4	Solve vorticity transport equation at a 1st iterative step
5	Solve Poisson equation for stream function at 1st iterative step
6	Find u and v using the vorticity function equation
7	Repeat the loop until the values converge

Table 2: Algorithm for Steady state analysis

8.1 Grid Independence Studies

The plot for u/U vs y/H at x = D/2, t=0.25 and Re = 10 follows similar profile as compared to the data obtained by Ghia et al has been given in 7.

8.2 Streamlines for Varying R and Re

TBD

8.3 Profiles for Steady Flow

TBD

9 Discussion

Understanding of the dynamics of this problem plays a very crucial role before stepping into more involved problems. This problem formulation can be further extended to account for temperture and heat flows, this can find many applications in digital electronics where designing of proper system for heat removal is required.

Appendix A

Algorithm implimentation for unsteady state.

```
from numpy import *
import scipy.linalg
import numpy as np
import matplotlib.pyplot as plt
```

```
from gridGen import grid,gridPlot,spacing,uGrid

# function to perform cfl analysis
def cflAnal(u,v,tIt):
    # applying cfl criteria
    dt1=[]
    for i in range(1,ny):
        for j in range(1,nx):
            dt1.append(abs((Re/2)*((1/(dx[i]**2))+(1/(r*r*dy[j]**2)))**(-1)))
    dt2=[]
    for i in range(1,ny):
        for j in range(1,nx):
            dt2.append(abs((u[i,j]/dx[i] + v[i,j]/dy[j])**(-1)))
    return(0.99*min(min(dt1),min(dt2)))

# function to initialize psi,w,u,v
def initialize():
    return(zeros((nx+1,ny+1)),zeros((nx+1,ny+1)),zeros((nx+1,ny+1)),zeros((nx+1,ny+1)))

# function to apply the boundary conditions
def bcsApply(wMat,psiMat,u,v):
    # applying bc on omega
    for j in range(ny+1):
        wMat[0,j] = 2*(psiMat[0,j]-psiMat[0,j-1])/(dy[0]**2)-(2*U)/dy[0]
    # use central differncing for second row
    print(u)
    for j in range(ny+1):
        wMat[1,j] = -(u[0,j]-u[2,j])/(dy[0]+dy[1])

    # applying BC, psi is zero at all the boundaries
    psiMat[0,:]=0    # left wall
    psiMat[nx,:]=0   # right wall
    psiMat[:,0]=0    # bottom wall
    psiMat[:,ny]=0   # top wall
    return(wMat,psiMat)

# main function performing the transient analysis
def calculation():
    a,b,u,v = initialize()
    u[0,:]=1  #  top layer with vel = U
    wMat,psiMat = bcsApply(a,b,u,v)
    print("\nBC applied\n wmatrix",wMat,"\n psiMat\n",psiMat)

    tItMax=10000
    tIt=0
    timeSteps = []
    while tIt<tItMax:
        # find in the velocities u,v from psiMat
        for i in range(1,ny):
            for j in range(1,nx):
                u[i,j] = (psiMat[i,j+1]-psiMat[i,j-1])/(dy[i]+dy[i-1])
                v[i,j] = -(psiMat[i+1,j]-psiMat[i-1,j])/(dx[i]+dx[i-1])
        # print("\nu",u,"\nv",v)

        dt = cflAnal(u,v,tIt)
        timeSteps.append(dt)
        print("\ntime step : ",dt)
        # calculation of wMat at time n+1
```

```
for i in range(1,nx):
    for j in range(1,ny):
        LHS = u[i,j]*((wMat[i+1,j]-wMat[i-1,j])/(dx[i]+dx[i-1])) + v[i,j]*((wMat
        ddx = dx[i-1]*(dx[i]**2)+dx[i]*(dx[i-1]**2)
        ddy = dy[i-1]*(dy[i]**2)+dy[i]*(dy[i-1]**2)

        wMat[i,j]=wMat[i,j]+ dt*(-LHS + (1/Re)*(2*(dx[i-1]*wMat[i+1,j]+dx[i]*wM
                                        (1/(r**2))*2*(dy[i-1]*wMat[i,j+1]+dy[i]*wM

        # print(LHS)
print("\nwMat at time ",tIt," step ",wMat)

# calculation of psiMat at time n+1 using Iteration
pItMax=1000
pIt=0
while pIt<pItMax:
    for i in range(1,nx):
        for j in range(1,ny):
            # stream function equation
            ddx = dx[i-1]*(dx[i]**2)+dx[i]*(dx[i-1]**2)
            ddy = dy[i-1]*(dy[i]**2)+dy[i]*(dy[i-1]**2)

            a = 0.5*((r**2)*ddx*ddy)/((r**2)*ddy*(dx[i]+dx[i-1])+ddx*(dy[i]+dy[i

            psiMat[i,j] = a*( wMat[i,j] + (2*(dx[i-1]*psiMat[i+1,j]+dx[i]*psiMat
                                ((1/(r**2))*2*(dy[i-1]*psiMat[i,j+1]+dy[i]*psiMat
            # resiudal
            res1 = wMat[i,j] + (2*(dx[i-1]*psiMat[i+1,j]+dx[i]*psiMat[i-1,j]-(dx
                  + ((1/(r**2))*2*(dy[i-1]*psiMat[i,j+1]+dy[i]*psiMat[i,j-1]-(dy

    if abs(res1)<(10**(-6)):
        print("Iteration finished")
        break

    pIt=pIt+1 # counter for psiMat Iterations
print("\npsiMat at ",tIt," step ",psiMat)

if sum(timeSteps)>=time:
    print("Reached the time ",sum(timeSteps))
    y=[0]
    for i in range(ny):
        y.append(y[len(y)-1]+dy[i])
    plt.scatter(y,u[int((nx-1)/2),:])
    plt.xlabel('y/H')
    plt.ylabel('u/U')
    plt.title("u/U vs y/H at x=D/2, Re="+str(Re)+" and time t=" + str(time))
    plt.savefig("r"+str(Re)+"t"+str(time)+".png")
    break
tIt=tIt+1 # counter for time steps

##time varying contour plot, uncomment below code to see streamLinePlot
# plt.ion()
# cs = plt.contour(flipud(psiMat),10,  extend='both')
# cs.cmap.set_over('red')
# cs.cmap.set_under('blue')
# cs.changed()
# plt.pause(0.0001)
# plt.clf()
```

```
        print("\n U \n",u)
        print("\n V \n",v)

# users input params
nx=64  # grids in x dir
ny=64  # grids in y dir
U = 1  # mormalized plate velocity
Re = 1 # reynolds number
r = 1 # aspect ratio
time = 4
x,y=grid(nx,ny,3)   # accepts (nx, ny, stretching param)
dx=spacing(x)  # grid divisions, symmetric
dy=spacing(y)  # grid divisions, symmetric
# gridPlot(x,y)     # plot the grid
calculation()
```

Appendix B

Algorithm implimentation for steady state.

```
from numpy import *
import scipy.linalg
from numpy import linalg as LA
import numpy as np
import matplotlib.pyplot as plt
from gridGen import grid,gridPlot,spacing,uGrid

def plotContour(matrix):
    cs = plt.contourf(matrix,  extend='both')
    cs.cmap.set_over('red')
    cs.cmap.set_under('blue')
    cs.changed()
    plt.show()

def initialize():
    return(zeros((nx+1,ny+1)),zeros((nx+1,ny+1)),zeros((nx+1,ny+1)),zeros((nx+1,ny+1)))

def bcsApply(wMat,psiMat,u,v):
    # applying bc on omega
    # wMat[0,:]=-2*psiMat[1,:]*(r**2)/(dx**2)
    # wMat[nx,:]=-2*psiMat[nx,:]*(r**2)/(dx**2)
    # wMat[:,1]=-2*psiMat[:,1]/(dy**2)
    # wMat[:,nx]=-2*psiMat[:,nx]/(dy**2)-2*u[:,nx]/dy

    # bcs as per the book
    for i in range(ny+1):
            wMat[0,i] = -((2*U)/dy)
    wMat[0,:]=-(3/(dy**2))*psiMat[1,:]-0.5*wMat[1,:]-3*U/dy     # top wall
    wMat[ny,:]=-(3/(dy**2))*psiMat[ny-1,:]-0.5*wMat[ny-1,:]      # bottom wall
    wMat[:,nx]=-(3*r*r/(dx**2))*psiMat[:,nx-1]-0.5*wMat[:,nx-1]     # right wall
    wMat[:,0]=-(3*r*r/(dx**2))*psiMat[:,1]-0.5*wMat[:,1]       # left wall

    # applying BC, psi is zero at all the boundaries
    psiMat[0,:]=0
    psiMat[nx,:]=0
    psiMat[:,0]=0
    psiMat[:,ny]=0
```

```
    return(wMat,psiMat)

def bcwMat(wMat,psiMat):
    wMat[0,:]=-(3/(dy**2))*psiMat[1,:]-0.5*wMat[1,:]-3*U/dy     # top wall
    wMat[ny,:]=-(3/(dy**2))*psiMat[ny-1,:]-0.5*wMat[ny-1,:]      # bottom wall
    wMat[:,nx]=-(3*r*r/(dx**2))*psiMat[:,nx-1]-0.5*wMat[:,nx-1]     # right wall
    wMat[:,0]=-(3*r*r/(dx**2))*psiMat[:,1]-0.5*wMat[:,1]       # left wall
    return(wMat)

def calculation():
    a,b,u,v = initialize()
    u[0,:]=1  #  top layer with vel = U
    wMat,psiMat = bcsApply(a,b,u,v)
    print("\nBC applied\n wMat",wMat,"\n psiMat\n",psiMat)

    pItMax=1000
    pIt=0
    while pIt<pItMax:
        # vorticity stream function relation
        for i in range(1,ny):
            for j in range(1,nx):
                wMatOld = wMat
                RHS=(1/Re)*(((wMat[i+1,j]+wMat[i-1,j])/(dx*dx)) + (1/(r*r))*((wMat[i,j+
                # 1st order upwind over one inner layer
                if i==1 or i==ny-1 or j==1 or j==nx-1:
                    if u[i,j]<=0 and v[i,j]>=0:
                        wMat[i,j]=((-u[i,j]/dx + v[i,j]/dy + (2/Re)*(1/(dx*dx)+(1/r**2)*
                    elif u[i,j]<=0 and v[i,j]<=0:
                        wMat[i,j]=((-u[i,j]/dx - v[i,j]/dy + (2/Re)*(1/(dx*dx)+(1/r**2)*
                    elif u[i,j]>=0 and v[i,j]<=0:
                        wMat[i,j]=((u[i,j]/dx - v[i,j]/dy + (2/Re)*(1/(dx*dx)+(1/r**2)*(
                    elif u[i,j]>=0 and v[i,j]>=0:
                        wMat[i,j]=((u[i,j]/dx + v[i,j]/dy + (2/Re)*(1/(dx*dx)+(1/r**2)*(
                # 2nd oder upwind for interor nodes
                else:
                    if u[i,j]<=0 and v[i,j]>=0:
                        wMat[i,j]=((-1.5*u[i,j]/dx + 1.5*v[i,j]/dy + (2/Re)*(1/(dx*dx)+(
                    elif u[i,j]<=0 and v[i,j]<=0:
                        wMat[i,j]=((-1.5*u[i,j]/dx - 1.5*v[i,j]/dy + (2/Re)*(1/(dx*dx)+(
                    elif u[i,j]>=0 and v[i,j]<=0:
                        wMat[i,j]=((1.5*u[i,j]/dx - 1.5*v[i,j]/dy + (2/Re)*(1/(dx*dx)+(1
                    elif u[i,j]>=0 and v[i,j]>=0:
                        wMat[i,j]=((1.5*u[i,j]/dx + 1.5*v[i,j]/dy + (2/Re)*(1/(dx*dx)+(1
                # print("old",wMatOld)
                # print("new",wMat)
                rs1 =np.amax(abs(wMatOld-wMat))
                print("rs1 ",rs1)
                print("-----------------------------------")

        # stream function equation
        for i in range(1,nx):
            for j in range(1,ny):
                psiMatOld=psiMat
                psiMat[i,j] = 0.5*((1/(dx**2) + (1/(r**2))*(1/dy*dy))**(-1))*(wMat[i,j]+
                rs2 = np.amax(abs(psiMatOld-psiMat))
                print("########################################")
                print("rs2",rs2)
        # # break if resiudal is close to zero
```

```
        # if abs(rs1)<1 and abs(rs2)<1:
        #     break
        print("pIt",pIt)

        # find in the velocities u,v from psiMat
        for i in range(1,ny):
            for j in range(1,nx):
                u[i,j] = (psiMat[i,j+1]-psiMat[i,j-1])/(2*dy)
                v[i,j] = -(psiMat[i+1,j]-psiMat[i-1,j])/(2*dx)
        print("\nu",u,"\nv",v)

        # bcs on omega
        wMat=bcwMat(wMat,psiMat)

        pIt=pIt+1 # counter for psiMat Iterations
    print("\npsiMat at ",pIt," Iteration ",psiMat)
    plotContour(flipud(psiMat))

nx=10  # elements in x dir
ny=10  # elements in y dir
U = 1  # mormalized plate velocity
Re = 100 # reynolds number
r = 1 # aspect ratio
H=1
D=1
dx=H/nx
dy=D/ny
calculation()
```

Appendix C

Algorithm implimentation for grid generation.

```
"""
Reference(s):
http://caefn.com/cfd/hyperbolic-tangent-stretching-grid
"""
from sympy import *
import matplotlib.pyplot as plt
import numpy as np
from array import *
from scipy.sparse import *

# TODO: fix the grid function for odd value of grid elements
def grid(nx,ny, gama):
    # use ty tx to change number of elements
    tx=(2)/((nx+1)+1)
    ty=(2)/((ny+1)+1)
    x=[]
    y=[]
    nx=[]
    ny=[]

    # x elements on left half
    for i in np.arange(0., 1., tx):
        nx.append(i)
    for j in nx:
        x.append(1-(np.tanh(gama*(1-(2*j)/len(nx))))/(np.tanh(gama)))
```

```
    # y elements on right half
    for i in np.arange(0., 1., ty):
        ny.append(i)
    for i in ny:
        y.append(1-(np.tanh(gama*(1-(2*i)/len(ny))))/(np.tanh(gama)))

    # mirroring x and y elements for the right half
    for i in range(len(nx)-1):
        x.append(x[len(nx)+i-1]+x[len(nx)-(i+1)]-x[len(nx)-(i+2)])
    for i in range(len(ny)-1):
        y.append(y[len(ny)+i-1]+y[len(ny)-(i+1)]-y[len(ny)-(i+2)])

    xd=[]
    yh=[]
    for i in x:
        xd.append(i/(x[len(x)-1]))
    for i in y:
        yh.append(i/(y[len(y)-1]))
    return(xd,yh)

def gridPlot(c,d):
    for i in c:
        for k in d:
            plt.scatter(i,k,color='black',marker='+')
    plt.show()

def spacing(test_list):
    res = [test_list[i + 1] - test_list[i] for i in range(len(test_list)-1)]
    return(res)

def uGrid(nx,ny):
    l1=[]
    l2=[]
    for i in range(nx+1):
        l1.append(1/(nx+1))
    for i in range(ny+1):
        l2.append(1/(ny+1))
    return(l1,l2)

## print(uGrid(10,10))
# print(grid(10,10,3))
## NOTE: call from any program using
## from gridGen import grid,gridPlot
# x,y=grid(10,10,3)    # accepts (nx, ny, game)
# gridPlot(grid(10,10,3)[0],grid(10,10,3)[1])
```

References

[1] Salih A. Streamfunction-vorticity formulation. 2013.

[2] J. Anderson. *Computational Fluid Dynamics*. Computational Fluid Dynamics: The Basics with Applications. McGraw-Hill Education, 1995.

[3] Bruneau C.H.and Jouron C. An efficient scheme for solving steady incompressible navier-stokes equations. *J. Comput.Phys. 89*, page 389–413, 1990.

[4] Friedrichs K. Courant R. and Lewy H. Math. Ann. Über die partiellen differenzengleichungen der mathematischen physik. *Springer-Verlag*, 1928.

[5] Isaacson E. Courant R. and Rees M. On the solution of nonlinear hyperbolic differential equations by finite differences. *Comm. Pure Appl. Math. 5*, pages 243–255, 1952.

[6] Ghia K.N. Ghia U. and Shin C.T. High-re solutions for incompressible flow using the navier-strokes equation and a multigrid method. 1982.

[7] Burggraf O.R. Analytical and numerical studies of the structure of steady separated flows. *J. Fluid Mech. 24*, page 113–151, 1966.

[8] Joseph Pedlosky. *Geophysical Fluid Dynamics*. Springer, 1987.

[9] Agarwal R.K. A third-order-accurate upwind scheme for navier-strokes solutions at high reynolds numbers. 1981.

[10] Mohamed A. Kotb Tamer A. AbdelMigid, Khalid M. Saqr and Ahmed A. Aboelfarag. Revisiting the lid-driven cavity flow problem: Review and new steady state benchmarking results using gpu accelerated code. 2016.