MAE 820 Project 2

D. W. Gould

Contents

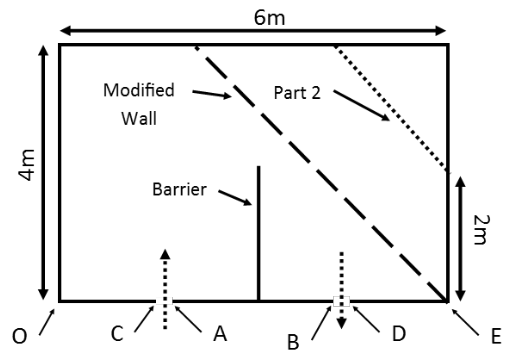


Figure 2.1: Diagram of model geometry

1. Introduction

In this project, the finite difference method was used to solve the lid driven cavity problem. The effectiveness of each solver used was also analyzed and additional suggestions for further reductions in computational times were presented.

2. Model

2.1. Geometry

In Figure 2.1, a basic diagram of the geometry is shown.

2.2. Governing Equations

Com, ComxComy

2.3. Discretization and Boundary Conditions

The governing equations of Section 2.2 were discretized using a staggered grid. Figure 2.2shows the node staggering method used in this work for an example grid consisting of 25 interior pressure nodes.

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = 0 \tag{2.3.1}$$

Aproximating each of the second derivatives in Equation 2.3.1 with second order central differencing scheme results in the discretized Equation 2.3.2. In this work a uniform grid with $\Delta x = \Delta y = 0.25$m was used. For boundary conditions, all exterior nodes - with the exception of those located between the points A and B - were defined to have a constant value of ψ=1. The nodes located at and between points A and B were defined to have a constant value of ψ=0. The mesh used in this work can be seen in Figures 4.1, 4.3, **??**, and 4.8.

$$\frac{\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}}{(\Delta x)^2} + \frac{\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}}{(\Delta y)^2} = 0 \tag{2.3.2}$$

3. Solution Method

3.1. Projection Method

In this work, Equation 2.3.2 was solved using several different iterative methods. Three of the methods used were explicit methods. The formula for each of these methods are provided as Equations 3.1.1 - 3.1.3.

In each of these three equations, the variable k indicates the level of iteration. The simplest of the three methods - that is, the Jacobi method shown in Equation 3.1.1 -

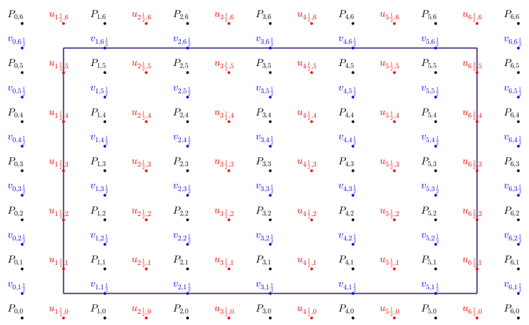

$P_{0,6}$ $u_{1\frac{1}{2},6}$ $P_{1,6}$ $u_{2\frac{1}{2},6}$ $P_{2,6}$ $u_{3\frac{1}{2},6}$ $P_{3,6}$ $u_{4\frac{1}{2},6}$ $P_{4,6}$ $u_{5\frac{1}{2},6}$ $P_{5,6}$ $u_{6\frac{1}{2},6}$ $P_{6,6}$
$v_{0,6\frac{1}{2}}$ $v_{1,6\frac{1}{2}}$ $v_{2,6\frac{1}{2}}$ $v_{3,6\frac{1}{2}}$ $v_{4,6\frac{1}{2}}$ $v_{5,6\frac{1}{2}}$ $v_{6,6\frac{1}{2}}$
$P_{0,5}$ $u_{1\frac{1}{2},5}$ $P_{1,5}$ $u_{2\frac{1}{2},5}$ $P_{2,5}$ $u_{3\frac{1}{2},5}$ $P_{3,5}$ $u_{4\frac{1}{2},5}$ $P_{4,5}$ $u_{5\frac{1}{2},5}$ $P_{5,5}$ $u_{6\frac{1}{2},5}$ $P_{6,5}$
$v_{0,5\frac{1}{2}}$ $v_{1,5\frac{1}{2}}$ $v_{2,5\frac{1}{2}}$ $v_{3,5\frac{1}{2}}$ $v_{4,5\frac{1}{2}}$ $v_{5,5\frac{1}{2}}$ $v_{6,5\frac{1}{2}}$
$P_{0,4}$ $u_{1\frac{1}{2},4}$ $P_{1,4}$ $u_{2\frac{1}{2},4}$ $P_{2,4}$ $u_{3\frac{1}{2},4}$ $P_{3,4}$ $u_{4\frac{1}{2},4}$ $P_{4,4}$ $u_{5\frac{1}{2},4}$ $P_{5,4}$ $u_{6\frac{1}{2},4}$ $P_{6,4}$
$v_{0,4\frac{1}{2}}$ $v_{1,4\frac{1}{2}}$ $v_{2,4\frac{1}{2}}$ $v_{3,4\frac{1}{2}}$ $v_{4,4\frac{1}{2}}$ $v_{5,4\frac{1}{2}}$ $v_{6,4\frac{1}{2}}$
$P_{0,3}$ $u_{1\frac{1}{2},3}$ $P_{1,3}$ $u_{2\frac{1}{2},3}$ $P_{2,3}$ $u_{3\frac{1}{2},3}$ $P_{3,3}$ $u_{4\frac{1}{2},3}$ $P_{4,3}$ $u_{5\frac{1}{2},3}$ $P_{5,3}$ $u_{6\frac{1}{2},3}$ $P_{6,3}$
$v_{0,3\frac{1}{2}}$ $v_{1,3\frac{1}{2}}$ $v_{2,3\frac{1}{2}}$ $v_{3,3\frac{1}{2}}$ $v_{4,3\frac{1}{2}}$ $v_{5,3\frac{1}{2}}$ $v_{6,3\frac{1}{2}}$
$P_{0,2}$ $u_{1\frac{1}{2},2}$ $P_{1,2}$ $u_{2\frac{1}{2},2}$ $P_{2,2}$ $u_{3\frac{1}{2},2}$ $P_{3,2}$ $u_{4\frac{1}{2},2}$ $P_{4,2}$ $u_{5\frac{1}{2},2}$ $P_{5,2}$ $u_{6\frac{1}{2},2}$ $P_{6,2}$
$v_{0,2\frac{1}{2}}$ $v_{1,2\frac{1}{2}}$ $v_{2,2\frac{1}{2}}$ $v_{3,2\frac{1}{2}}$ $v_{4,2\frac{1}{2}}$ $v_{5,2\frac{1}{2}}$ $v_{6,2\frac{1}{2}}$
$P_{0,1}$ $u_{1\frac{1}{2},1}$ $P_{1,1}$ $u_{2\frac{1}{2},1}$ $P_{2,1}$ $u_{3\frac{1}{2},1}$ $P_{3,1}$ $u_{4\frac{1}{2},1}$ $P_{4,1}$ $u_{5\frac{1}{2},1}$ $P_{5,1}$ $u_{6\frac{1}{2},1}$ $P_{6,1}$
$v_{0,1\frac{1}{2}}$ $v_{1,1\frac{1}{2}}$ $v_{2,1\frac{1}{2}}$ $v_{3,1\frac{1}{2}}$ $v_{4,1\frac{1}{2}}$ $v_{5,1\frac{1}{2}}$ $v_{6,1\frac{1}{2}}$
$P_{0,0}$ $u_{1\frac{1}{2},0}$ $P_{1,0}$ $u_{2\frac{1}{2},0}$ $P_{2,0}$ $u_{3\frac{1}{2},0}$ $P_{3,0}$ $u_{4\frac{1}{2},0}$ $P_{4,0}$ $u_{5\frac{1}{2},0}$ $P_{5,0}$ $u_{6\frac{1}{2},0}$ $P_{6,0}$

Figure 2.2: Example of staggered grid

calculates the new value of stream at the point of interest simply as a function of the values of the of the surrounding nodes in the previous iteration step.

The Gauss-Seidel (GS) algorithm, shown in Equation 3.1.2, improves on the efficiency of the Jacobi algorithm by incorporating the most recent values obtained for its adjacent nodes in its calculations. Thus, while the GS algorithm will not provide any per-iteration reductions computational time, its use of adjacent node values from the current iteration level where available should allow it to converge more quickly than the baseline Jacobi method.

$$\beta = \frac{\Delta x}{\Delta y}$$

$$\psi_{i,j}^{k+1} = \frac{\psi_{i+1,j}^{k} + \psi_{i-1,j}^{k} + \beta^2\left(\psi_{i,j+1}^{k} + \psi_{i,j-1}^{k}\right)}{2\left(1+\beta^2\right)} \tag{3.1.1}$$

$$\psi_{i,j}^{k+1} = \frac{\psi_{i+1,j}^{k} + \psi_{i-1,j}^{k+1} + \beta^2\left(\psi_{i,j+1}^{k} + \psi_{i,j-1}^{k+1}\right)}{2\left(1+\beta^2\right)} \tag{3.1.2}$$

$$\psi_{i,j}^{k+1} = \left(1-\omega\right)\psi_{i,j}^{k} + \omega\frac{\psi_{i+1,j}^{k} + \psi_{i-1,j}^{k+1} + \beta^2\left(\psi_{i,j+1}^{k} + \psi_{i,j-1}^{k+1}\right)}{2\left(1+\beta^2\right)} \tag{3.1.3}$$

Equation 3.1.3 shows the general formula for the Successive Over-Relaxation (SOR) method. The SOR method improves on the GS solution method by incorporating over - or, if increased stability is desired, under - relaxation into the equation using the relaxation parameter ω. When ω is equal to 1, the SOR method becomes identical to the GS method. Raising ω to values greater than one can increase the rate of convergence at the cost of stability. Conversely, lowering the relaxation parameter below unity will result in dramatically increased computational times, but provides greater stability.

3.2. Poisson Solver

Two different partially implicit solution methods were used, Successive Over-Relaxation by Lines (SLOR) and the Alternating Direction Implicit (ADI) method. In Equation 3.2.3, three unknown values are present. These unknowns are the value of the stream function at the central point, $\psi_{i,j}^{k+1}$, and the values at the the two points adjacent to the central point on row j.<<<<<<< HEAD:Project22.lyx Interpolation was used whenever the scheme required information at a location between nodes.

Examples of this discretization for some of the terms in Equations **??** – **??** are given in Equations 3.2.1 and 3.2.2.

$$\left[\frac{\partial}{\partial x}\left(u^2\right)\right]_{i+\frac{1}{2},j} = \frac{\left(u_{i+1,j}\right)^2 - \left(u_{i,j}\right)^2}{\Delta x} \tag{3.2.1}$$

$$\left[\frac{\partial}{\partial y}\left(uv\right)\right]_{i+\frac{1}{2},j} = \frac{\left(uv\right)_{i+\frac{1}{2},j+\frac{1}{2}} - \left(uv\right)_{i+\frac{1}{2},j-\frac{1}{2}}}{\Delta y} \tag{3.2.2}$$

For the u ======= Solving this equation requires that an implicit method be used so that the values of can be solved for the entire row j simultaneously.

While more difficult to implement, this method has the major advantage that it communicates information from the BC at the ends of each row instantly in a single iteration. This is in contrast to the very slow dispersal of information found in the aforementioned explicit methods in which it could take 7 iterations for a node in the center of the domain might to be affected by a change in the boundary conditions.

For the ADI method, the principal remains the same. However, instead of each iteration consisting of solving the domain row by row, in the Alternating Direction Implicit method will follow a sweep of the rows of a domain with a similar sweep of the columns of a domain. This method allows information from each extremity of the domain to quickly propagate to the center. The equation used for the ADI method is the same as that used for the SLOR method, but which terms are known and which terms are solved for depend on the direction of the sweep.

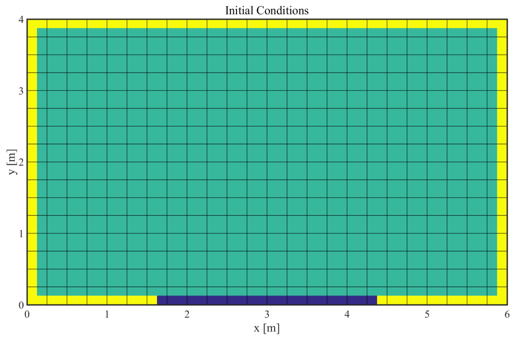
$$\psi_{i,j}^{k+1} = (1-\omega)\,\psi_{i,j}^{k} + \frac{\omega}{2\left(1+\beta^2\right)}\psi_{i+1,j}^{k+1} + \psi_{i-1,j}^{k+1} + \beta^2\left(\psi_{i,j+1}^{k} + \psi_{i,j-1}^{k+1}\right) \tag{3.2.3}$$

3.2.1. Thomas Algorithm

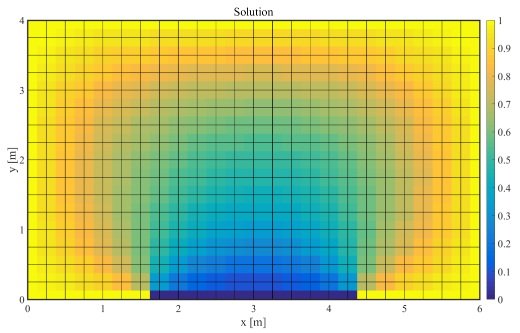
The tridiagonal matrices resulting from the implementation of both the SLOR and ADI methods was simultaneously solved using the Thomas Algorithm. While only valid for sparse, tridiagonal matrices such as those created by the SLOR and ADI methods, the Thomas Algorithm can theoretically be made to be highly efficient. The general algorithm consists of working down each row of the matrix, eliminating the subdiagonal term in each row. This results in an upper diagonalizing of the equation matrix, allowing for the unknown stream values to be obtained by back substitution. The specific code written to do this can be found in Appendix **??**.

4. Results - Part 1

The inital conditons used and results obtained for part one can be seen in Figure 4.1. In each of these figures, the mesh used is overlayed.



(a) Part 1 initial conditions with mesh overlay

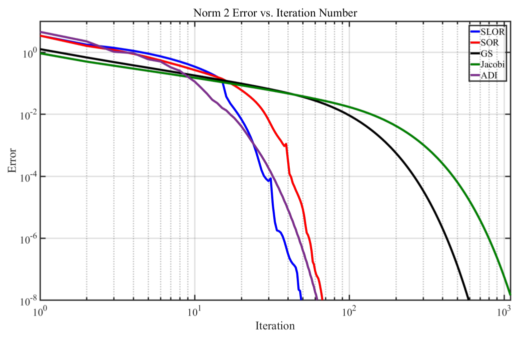


(b) Part one solution with mesh overlay

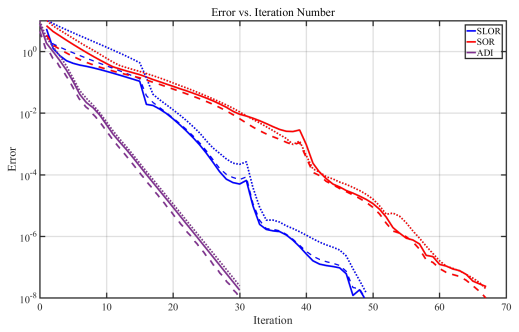
Figure 4.1: Solution of part 1

In Figure 4.2, the effects each solver on the convergence of the solution is shown. A convergence criteria of 1E-8 was used solvers in this project.

For the second part of the this project, the box geometry was modified so that a portion of the right corner was removed. The line defining the removed corner is shown as the dotted line in 2.1. This change in the size of the domain was implemented by adjusting the initial conditions and boundary conditions used when solving for ψ. The new wall was applied by initially assigning all nodes beyond the new boundary line as boundary nodes possessing constant values of $\omega = 1$. A visual depiction of this application



(a) Error vs iteration number for each solver



(b) Comparison of norm 1, norm 2, norm∞ error

Figure 4.2: Relative error from examined solution methods

	Jacobi	**G-S**	**SOR**	**SLOR**	**ADI**
Time per Iteration [s]	5.75E-5	5.78E-5	6.58E-5	2.0E-4	2.34E-4
Solution Time [s]	6.5E-2	3.4E-2	4.5E-3	1.0E-2	7.25E-3
Optimal ω	N/A	N/A	1.7189	1.2421	1.3
Minimum Iterations	1127	592	68	50	31

Table 4.1: Solver effectiveness

can be seen in Figure 4.3a.

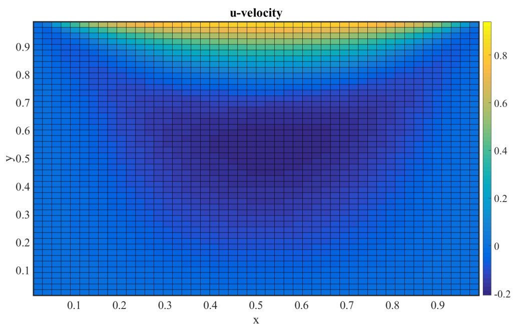
The restricted domain was seen to slightly modify the stream function calculated within the domain. This change can be seen in Figure 4.3b in which the a contour plot with the results from the geometry of part 1 is overlayed with the new results from the domain with the removed corner. In Figure 4.3b, the black contours are those from the geometry of part 1, and the colored contours are from the results calculated with the new geometry.

While Figure 4.3b does make it clear that *some* change in the results did indeed occur due to the modified corner boundary, in order to both better understand the effect of geometry on the flow patterns within the box and provided validation for the geometry adjustment method used, the domain of analysis was further restricted by removing all nodes beyond the boundary identified in Figure 2.1 as the "Modified Wall". Unsurprisingly, moving the wall this far into the domain proved to have a much larger effect.

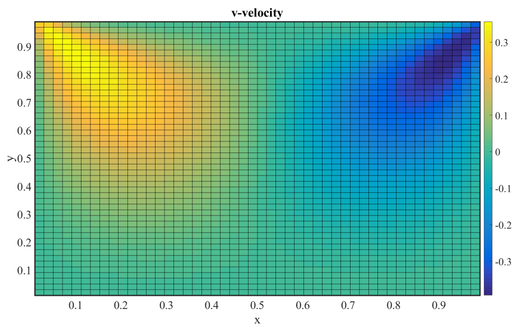
Finally, to further disturb the flow, a partial barrier was added in the region between the inlet and outlet by assigning a line of nodes at x=3m to be equal 0. This assignment created a new wall attached to the wall between points A and C. This is seen in Figure 4.8a. Unsurprisingly, the addition of the barrier worked in conjunction with the extended wall and resulted in a highly disturbed flow path. A contour and pseudocolor plot depicting this is presented in Figure 4.8b. The code calculating this effect is provided in Appendix **??**.

		dt	dx	Nodes	Tolerance	Computational Time
Re	1					
	10					
	100					
	400					
	400	2.0E-4				
	400	0.005	0.0101	100x100	5.0E-7	270
	1000	0.0036	0.0204	50x50	5.0E-7	363

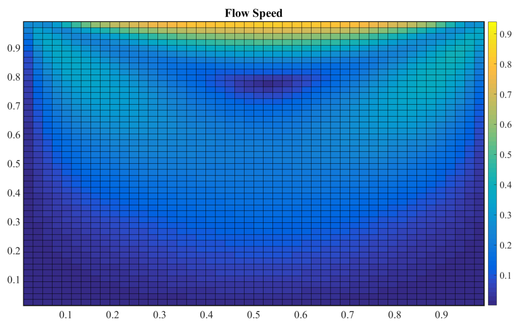
Table 4.2: Results



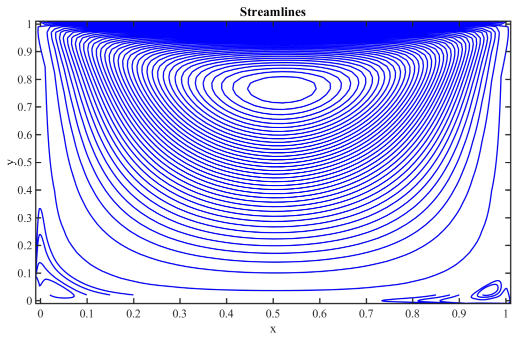
(a) u



(b) v



(c) Flow Speed

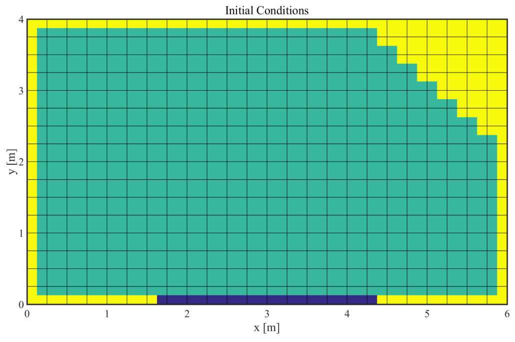


(d) ψ

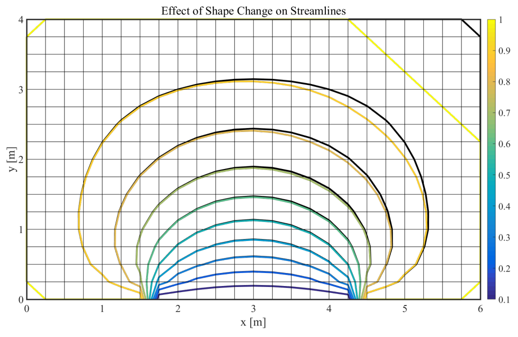
Figure 4.3: Results - Re 10

5. Conclusion

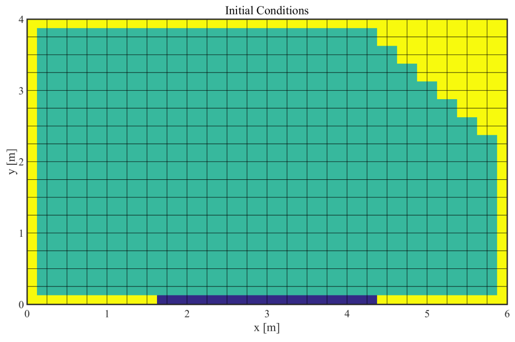
Despite the limited resolution, the methods employed in solving the Laplace equation for the stream function within the domain of interest proved to be very effective. Regarding the different solution methods, it was found that when using Matlab, the ability to vectorize an equation is of significant importance. This is admittedly a significant drawback if one is going to be using the language for iterative solutions to finite difference equations. Unfortunately, it was also discovered by the author that all other coding options are horrible and evil. Thus, in the future, this author would recommend that FDE equations can be best solved by either using working in Matlab and using Matlab's built in, vectorized functions as much as possible, or simply buying some computer science grad student a 6 pack of beer to program the solution code for you in a awful, but fast language such as C or Fortran.



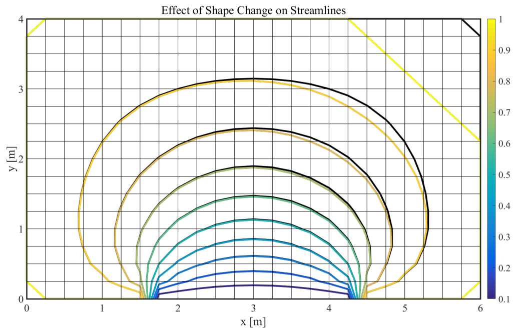
(a) u



(b) v

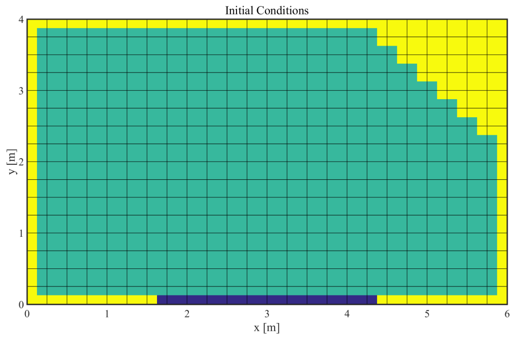


(c) Flow Speed

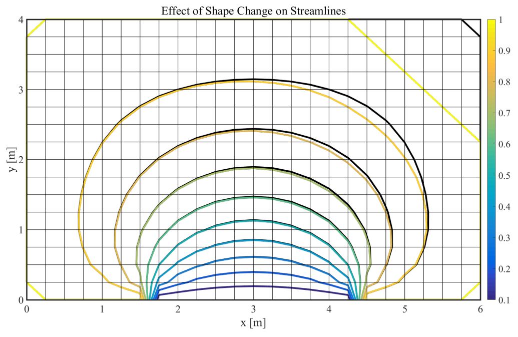


(d) ψ

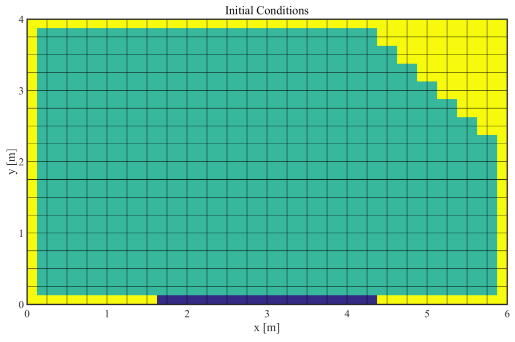
Figure 4.4: Results - Re 100



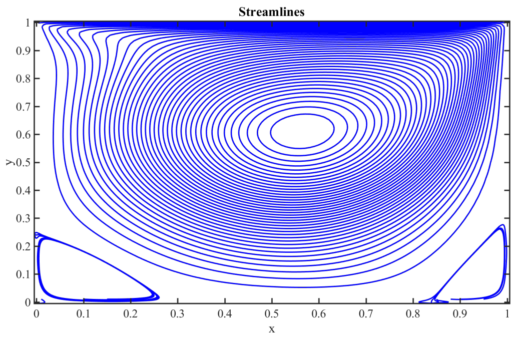
(a) u



(b) v

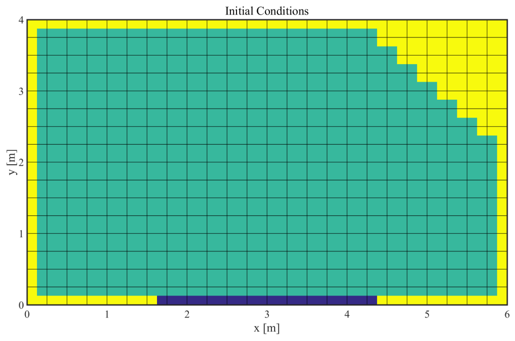


(c) Flow Speed

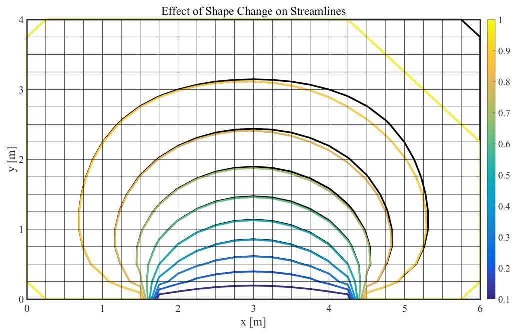


(d) ψ

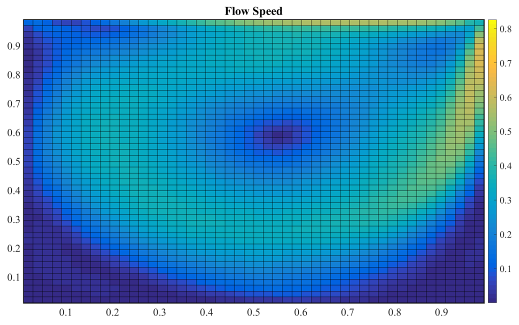
Figure 4.5: Results - Re 400



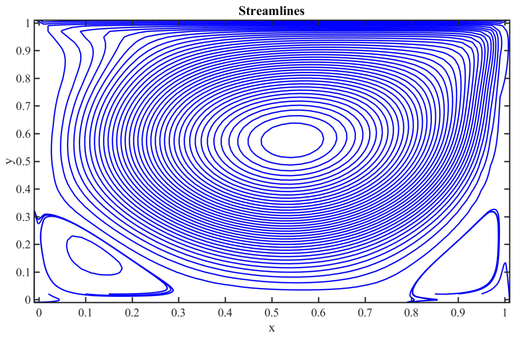
(a) u



(b) v

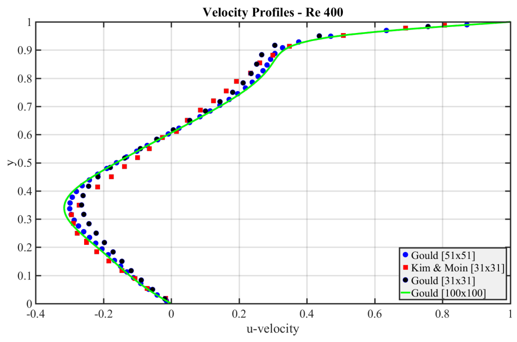


(c) Flow Speed

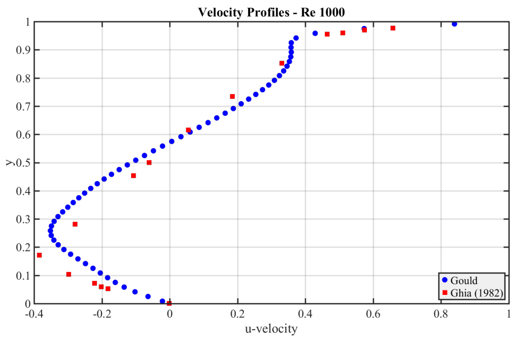


(d) ψ

Figure 4.6: Results - Re 1000

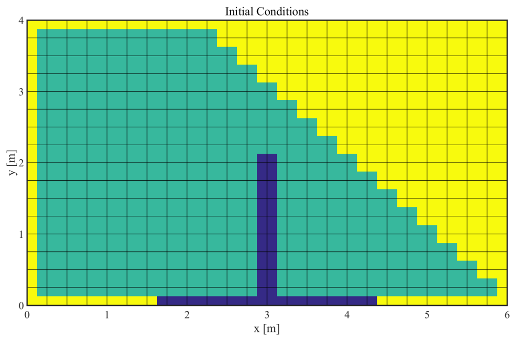


(a) Re 400

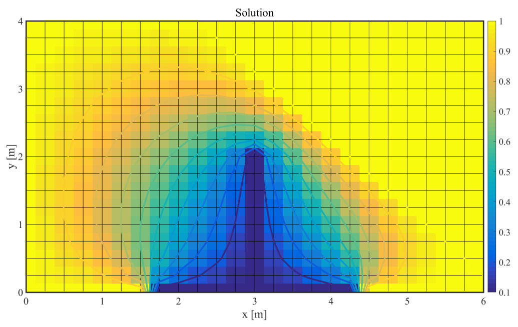


(b) Re400

Figure 4.7: Comparison of calculated u velocity along cavity centerline



(a) Initial conditions for simulation with barrier



(b) Effect of barrier on solution

Figure 4.8: Modified geometry - shape and barrier