# Homework 3

Cristóbal Armaza, Chris Silvia

March 16, 2017

## Organization

We met to discuss the involved equations, as well as to figure how to efficiently handle the data for the code. We set up a Github repo to collaborate. Cristóbal began the code, Chris also began the code in parallel, and then merged the progress Cristóbal had made into his code to complete the code. With the code fully operating, Chris made the graphs to compare with Hou et al. (1995), while Cristóbal made the streamline plots and finished the write up.

## 1 Analytical Equations

### 1.1 Navier-Stokes for a compressible fluid

We want to solve the continuity equation

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_k}\left(\rho u_k\right) = 0, \tag{1}$$

and the momentum equation

$$\frac{\partial}{\partial t}\left(\rho u_i\right) + \frac{\partial}{\partial x_k}\left(\rho u_k u_i\right) = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_k}\left[\mu\left(\frac{\partial u_i}{\partial x_k} + \frac{\partial u_k}{\partial x_i} - \frac{2}{3}\delta_{ik}\frac{\partial u_l}{\partial x_l}\right)\right], \tag{2}$$

where we will assume the equation of state $p = c^2\rho$, $c$ being the speed of sound in the fluid. Note that the momentum equation can also be written in conservative form as

$$\frac{\partial}{\partial t}\left(\rho u_i\right) + \frac{\partial T_{ik}}{\partial x_k} = 0, \tag{3}$$

where

$$T_{ik} := \rho u_i u_k + p\delta_{ik} - \mu\left(\frac{\partial u_i}{\partial x_k} + \frac{\partial u_k}{\partial x_i} - \frac{2}{3}\delta_{ik}\frac{\partial u_l}{\partial x_l}\right) \tag{4}$$

is the stress-energy tensor.

### 1.2 2-D Equations in Cartesian Equations

Replacing $x_k = (x, y)$, $u_i = (u, v)$, we obtain

$$T_{xx} = \rho u^2 + c^2\rho - \frac{2\mu}{3}\left(2\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}\right), \tag{5}$$

$$T_{yy} = \rho v^2 + c^2\rho - \frac{2\mu}{3}\left(2\frac{\partial v}{\partial y} - \frac{\partial u}{\partial x}\right), \tag{6}$$

$$T_{xy} = T_{yx} = \rho uv - \mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right), \tag{7}$$

and the equations we want to solve are

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}(\rho u) + \frac{\partial}{\partial y}(\rho v) = 0, \tag{8}$$

$$\frac{\partial}{\partial t}(\rho u) + \frac{\partial}{\partial x}\left(\rho u^2 + c^2\rho\right) + \frac{\partial}{\partial y}(\rho uv) - \mu\left(\frac{4}{3}\frac{\partial^2 u}{\partial x^2} + \frac{1}{3}\frac{\partial^2 v}{\partial x \partial y} + \frac{\partial^2 u}{\partial y^2}\right) = 0, \tag{9}$$

$$\frac{\partial}{\partial t}(\rho v) + \frac{\partial}{\partial x}(\rho uv) + \frac{\partial}{\partial y}\left(\rho v^2 + c^2\rho\right) - \mu\left(\frac{\partial^2 v}{\partial x^2} + \frac{1}{3}\frac{\partial^2 u}{\partial x \partial y} + \frac{4}{3}\frac{\partial^2 v}{\partial y^2}\right) = 0. \tag{10}$$

Here we have explicitly separated the hyperbolic part of each equation from the viscous (diffusive) terms.

## 2 Numerics

### 2.1 Finite Difference

First off, we discretize the domain using a regular mesh in 2D over a finite amount of time:

$$t_n = t_0 + n\,\Delta t_n, \qquad n = 0, \ldots, n_t, \tag{11}$$

$$x_i = x_0 + i\,\Delta x, \qquad i = 0, \ldots, n_x - 1, \tag{12}$$

$$y_j = y_0 + j\,\Delta x, \qquad j = 0, \ldots, n_y - 1, \tag{13}$$

and set our coordinate system so that the left bottom corner is the origin, $x_0 = y_0 = 0$, and the simulation starts at $t_0 = 0$. The interval $\Delta t_n$ will be discussed later. Every physical quantity $\phi(t, x, y)$ is discretized as $\phi(t_n, x_i, y_j) = \phi^n_{i,j}$. On this mesh, we apply the MacCormack scheme for the hyperbolic part of each equation. To do so, these parts must be discretized using forward and backward schemes for the predictor and corrector steps, respectively. These discretizations read

$$\phi'_i = \frac{1}{\Delta x}(\phi_{i+1} - \phi_i) - \frac{\Delta x}{2}\phi''_i + \mathcal{O}(\Delta x^3), \tag{14}$$

$$\phi'_i = \frac{1}{\Delta x}(\phi_i - \phi_{i-1}) + \frac{\Delta x}{2}\phi''_i + \mathcal{O}(\Delta x^3), \tag{15}$$

respectively.

For the viscous term, we can just use a central scheme,

$$\phi''_i = \frac{1}{\Delta x^2}(\phi_{i+1} - 2\phi_i + \phi_{i-1}) - \frac{\Delta x^2}{12}\phi^{(iv)}_i + \mathcal{O}(\Delta x^4). \tag{16}$$

The discretization for the cross derivative is a little less trivial. Recalling the central scheme for a first derivative,

$$\phi_i' = \frac{1}{2\,\Delta x}\left(\phi_{i+1} - \phi_{i-1}\right) - \frac{\Delta x^2}{6}\phi_i''' + \mathcal{O}(\Delta x^4), \tag{17}$$

we differentiate this with respect to $y$ to get

$$\left.\frac{\partial^2 \phi}{\partial y \partial x}\right|_{i,j} = \frac{1}{2\,\Delta x}\left(\frac{\partial \phi_{i+1,j}}{\partial y} - \frac{\partial \phi_{i-1,j}}{\partial y}\right) - \frac{\Delta x^2}{6}\frac{\partial^4 \phi}{\partial y \partial x^3} + \frac{\partial}{\partial y}\mathcal{O}(\Delta x^4). \tag{18}$$

Applying the same formula on all these terms (including the leading error term), and collecting terms, we arrive at

$$\left.\frac{\partial^2 \phi}{\partial y \partial x}\right|_{i,j} = \frac{1}{4\,\Delta x\,\Delta y}\left(\phi_{i+1,j+1} - \phi_{i+1,j-1} - \phi_{i-1,j+1} + \phi_{i-1,j-1}\right) \tag{19}$$

$$- \frac{\Delta x^2}{6}\frac{\partial^4 \phi}{\partial x^3 \partial y} - \frac{\Delta y^2}{6}\frac{\partial^4 \phi}{\partial x \partial y^3}, \tag{20}$$

so we still have a second-order accurate discretization independently for each coordinate. Also, this expression clearly preserves the symmetry of cross derivatives.

## 2.2 Expressions for Predictors and Correctors

Replacing all the above finite difference expressions, we get

$$\rho^\star_{i,j} = \rho^n_{i,j} - a_1\left[(\rho u)^n_{i+1,j} - (\rho u)^n_{i,j}\right] - a_2\left[(\rho v)^n_{i,j+1} - (\rho v)^n_{i,j}\right], \tag{21}$$

$$(\rho u)^\star_{i,j} = (\rho u)^n_{i,j} - a_1\left[(\rho u^2)^n_{i+1,j} - (\rho u^2)^n_{i,j} + c^2\left(\rho^n_{i+1,j} - \rho^n_{i,j}\right)\right] - a_2\left[(\rho uv)^n_{i,j+1} - (\rho uv)^n_{i,j}\right] \tag{22}$$

$$+ \frac{4a_3}{3}\left[u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}\right] + a_4\left[u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1}\right] \tag{23}$$

$$+ a_5\left[v^n_{i+1,j+1} - v^n_{i+1,j-1} - v^n_{i-1,j+1} + v^n_{i-1,j-1}\right], \tag{24}$$

$$(\rho v)^\star_{i,j} = (\rho v)^n_{i,j} - a_1\left[(\rho uv)^n_{i+1,j} - (\rho uv)^n_{i,j}\right] - a_2\left[(\rho v^2)^n_{i,j+1} - (\rho v^2)^n_{i,j} + c^2\left(\rho^n_{i,j+1} - \rho^n_{i,j}\right)\right] \tag{25}$$

$$+ a_3\left[v^n_{i+1,j} - 2v^n_{i,j} + v^n_{i-1,j}\right] + \frac{4a_4}{3}\left[v^n_{i,j+1} - 2v^n_{i,j} + v^n_{i,j-1}\right] \tag{26}$$

$$+ a_5\left[u^n_{i+1,j+1} - u^n_{i+1,j-1} - u^n_{i-1,j+1} + u^n_{i-1,j-1}\right] \tag{27}$$

for the predictor step, and

$$\rho^{n+1}_{i,j} = \frac{1}{2}\left[\rho^n_{i,j} + \rho^\star_{i,j}\right] - \frac{a_1}{2}\left[(\rho u)^\star_{i,j} - (\rho u)^\star_{i-1,j}\right] - \frac{a_2}{2}\left[(\rho v)^\star_{i,j} - (\rho v)^\star_{i,j-1}\right], \tag{28}$$

$$(\rho u)^{n+1}_{i,j} = \frac{1}{2}\left[(\rho u)^n_{i,j} + (\rho u)^\star_{i,j}\right] - \frac{a_1}{2}\left[(\rho u^2)^\star_{i,j} - (\rho u^2)^\star_{i-1,j} + c^2\left(\rho^\star_{i,j} - \rho^\star_{i-1,j}\right)\right] \tag{29}$$

$$- \frac{a_2}{2}\left[(\rho uv)^\star_{i,j} - (\rho uv)^\star_{i,j-1}\right] + \frac{2a_3}{3}\left[u^\star_{i+1,j} - 2u^\star_{i,j} + u^\star_{i-1,j}\right] \tag{30}$$

$$+ \frac{a_4}{2}\left[u^\star_{i,j+1} - 2u^\star_{i,j} + u^\star_{i,j-1}\right] + \frac{a_5}{2}\left[v^\star_{i+1,j+1} - v^\star_{i+1,j-1} - v^\star_{i-1,j+1} + v^\star_{i-1,j-1}\right], \tag{31}$$

$$(\rho v)^{n+1}_{i,j} = \frac{1}{2}\left[(\rho v)^n_{i,j} + (\rho v)^\star_{i,j}\right] - \frac{a_2}{2}\left[(\rho v^2)^\star_{i,j} - (\rho v^2)^\star_{i,j-1} + c^2\left(\rho^\star_{i,j} - \rho^\star_{i,j-1}\right)\right] \tag{32}$$

$$- \frac{a_1}{2}\left[(\rho uv)^\star_{i,j} - (\rho uv)^\star_{i-1,j}\right] + \frac{a_3}{2}\left[v^\star_{i+1,j} - 2v^\star_{i,j} + v^\star_{i-1,j}\right] \tag{33}$$

$$+ \frac{2a_4}{3}\left[v^\star_{i,j+1} - 2v^\star_{i,j} + v^\star_{i,j-1}\right] + \frac{a_5}{2}\left[u^\star_{i+1,j+1} - u^\star_{i+1,j-1} - u^\star_{i-1,j+1} + u^\star_{i-1,j-1}\right], \tag{34}$$

for the corrector step, where

$$a_1 = \frac{\Delta t}{\Delta x}, \quad a_2 = \frac{\Delta t}{\Delta y}, \quad a_3 = \frac{\mu \Delta t}{\Delta x^2}, \quad a_4 = \frac{\mu \Delta t}{\Delta y^2}, \quad a_5 = \frac{\mu \Delta t}{12 \Delta x\, \Delta y}. \tag{35}$$

These formulas are valid for interior points only.

## 2.3 Boundary Conditions

Boundary conditions depend on the particular situation we are studying. For the lid-cavity problem with the sliding wall being the top boundary, the boundary conditions for the velocity are

$$u^n_{i,0} = 0,\ u^n_{i,n_y-1} = U \quad \text{for } i = 1, \ldots, (n_x - 2), \tag{36}$$

$$u^n_{0,j} = u^n_{n_x-1,j} = 0 \quad \text{for } j = 0, \ldots, n_y - 1, \tag{37}$$

$$v^n_{i,j} = 0. \tag{38}$$

The boundary conditions for the density are trickier. They can be obtained from the continuity equation applied at every boundary. For example, for the boundary $x = 0$, $v = 0$ so $\partial(\rho v)/\partial y = 0$ there. Then, the continuity equation reads

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}(\rho u) = 0, \qquad (x = 0). \tag{39}$$

In order to maintain second-order accuracy, we use a 3-point forward stencil

$$\phi'_i = \frac{-3\phi_i + 4\phi_{i+1} - \phi_{i+2}}{2\,\Delta x} \tag{40}$$

to construct a MacCormack method at the boundary,

$$\rho^\star_{0,j} = \rho^n_{0,j} - \frac{a_1}{2}\left[-3(\rho u)^n_{0,j} + 4(\rho u)^n_{1,j} - (\rho u)^n_{2,j}\right], \tag{41}$$

$$\rho^{n+1}_{0,j} = \frac{1}{2}\left[\rho^n_{0,j} + \rho^\star_{0,j}\right] - \frac{a_1}{4}\left[-3(\rho u)^\star_{0,j} + 4(\rho u)^\star_{1,j} - (\rho u)^\star_{2,j}\right], \tag{42}$$

for $j = 1, \ldots, n_y - 2$. Similarly, for the right wall ($x = (n_x - 1)\Delta x$), we apply a 3-point backward stencil,

$$\rho^\star_{n_x-1,j} = \rho^n_{n_x-1,j} - \frac{a_1}{2}\left[(\rho u)^n_{n_x-3,j} - 4(\rho u)^n_{n_x-2,j} + 3(\rho u)^n_{n_x-1,j}\right], \tag{43}$$

$$\rho^{n+1}_{n_x-1,j} = \frac{1}{2}\left[\rho^n_{n_x-1,j} + \rho^\star_{n_x-1,j}\right] - \frac{a_1}{4}\left[(\rho u)^\star_{n_x-3,j} - 4(\rho u)^\star_{n_x-2,j} + 3(\rho u)^\star_{n_x-1,j}\right], \tag{44}$$

for $j = 1, \ldots, n_y - 2$. Note that these two schemes do not include the points at the four corners; they deserve special attention. For the bottom, $u = 0$ and thus $\partial(\rho u)/\partial x = 0$, so

$$\rho^\star_{i,0} = \rho^n_{i,0} - \frac{a_2}{2}\left[-3(\rho v)^n_{i,0} + 4(\rho v)^n_{i,1} - (\rho v)^n_{i,2}\right], \tag{45}$$

$$\rho^{n+1}_{i,0} = \frac{1}{2}\left[\rho^n_{i,0} + \rho^\star_{i,0}\right] - \frac{a_2}{4}\left[-3(\rho v)^\star_{i,0} + 4(\rho v)^\star_{i,1} - (\rho v)^\star_{i,2}\right]. \tag{46}$$

for $i = 1, \ldots, n_x - 2$. For the top, $u = U$ so $\partial(\rho u)/\partial x = U \partial\rho/\partial x$. Thus,

$$\rho^\star_{i,n_y-1} = \rho^n_{i,n_y-1} - \frac{a_1 U}{2}\left[\rho^n_{i+1,n_y-1} - \rho^n_{i-1,n_y-1}\right] - \frac{a_2}{2}\left[(\rho v)^n_{i,n_y-3} - 4(\rho v)^n_{i,n_y-2} + 3(\rho v)^n_{i,n_y-1}\right], \tag{47}$$

$$\rho^{n+1}_{i,n_y-1} = \frac{1}{2}\left[\rho^n_{i,n_y-1} + \rho^\star_{i,n_y-1}\right] - \frac{a_1 U}{4}\left[\rho^\star_{i+1,n_y-1} - \rho^\star_{i-1,n_y-1}\right] \tag{48}$$

$$- \frac{a_2}{4}\left[(\rho v)^\star_{i,n_y-3} - 4(\rho v)^\star_{i,n_y-2} + 3(\rho v)^\star_{i,n_y-1}\right], \tag{49}$$

for $i = 1, \ldots, (n_x - 2)$. Note that for $\partial\rho/\partial x$ we use a central stencil, which is also second-order accurate. Finally, for each corner, one notes that there are two available formulas provided by the above conditions. We simple take the conditions determined by the left and right sides, respectively.

### 2.4 System Ready to Be Solved!

In summary, our grid consists of $n_x \times n_y$ points, out of which $2(n_y - 2) + 2(n_x - 2) + 4$ are determined by boundary conditions. The other $n_x \times n_y - 2n_y - 2n_x + 4 = (n_x - 2)(n_y - 2)$ points are determined by the $(n_x - 2)(n_y - 2)$ equations for the interior points.

### 2.5 Initial Conditions

We took uniform density everywhere and zero momentum (except at the top boundary, where there is the velocity $U$) as initial conditions.

### 2.6 Stability

We took the God-given formula

$$\Delta t \leq \frac{\sigma}{1 + 2\,/\mathrm{Re}_\Delta} \left[ \frac{U}{\Delta x} + \frac{U}{\Delta y} + c\sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}} \right]^{-1}, \tag{50}$$

where $\sigma \leq 1$ is a safety parameter, which is arbitrary, and $\mathrm{Re}_\Delta = \min(\rho\Delta x|u|/\mu, \rho\Delta y|v|/\mu)$, where the values in bar are understood to be the maximum values over the domain.

## 3 Results

### 3.1 Comparison with Hou et al. (1995)

Figure 3.1 shows the $u$ velocity along the center line of the simulation region, at $x = H/2$, and compares it to Hou et al., who determined the velocity using a lattice boltzmann method. We see that there is good agreement with Hou's profile, in both the qualitative shape, and the approximate location of the transition from negative to positive velocities.

### 3.2 Streamlines

Figure 3.2 show the streamlines for different timesteps during the evolution. In these plots, $\mathrm{Re} = 100$ and the color code indicates the speed of the flow, defined as $\sqrt{u^2 + v^2}$. Starting from the initial condition of zero momentum, we observe the formation of a primary circulating eddy, which induces the formation of two secondary counter-rotating eddies at the bottom corners. This is in agreement with Hou et al. (1995) and the results analyzed there. By the end of the simulation, we confirm that the flow pattern remains stable in time, presenting the steady flow shown in Fig. 3.2. For comparison, we also include the result from Hou et al., finding a remarkable agreement between both results.

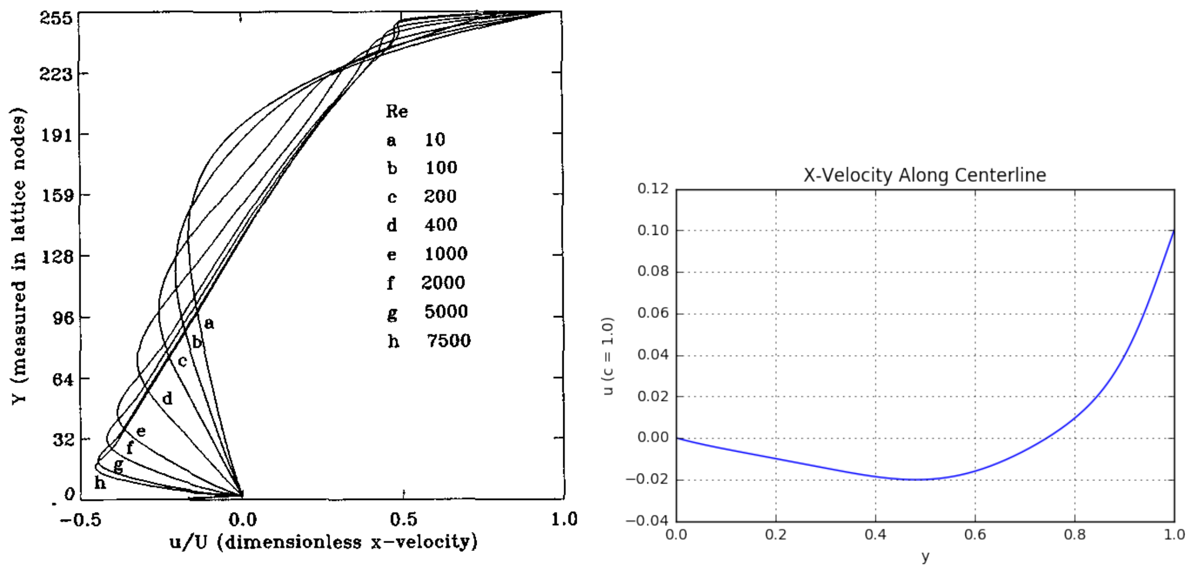

Figure 1: Central Streamline, compared to Hou et al. (1995). Recall for us, $Re = 100$.

## A Code

```
#include <iostream>
#include <math.h>
#include <ctime>
#include <stdio.h>
#include <fstream>

using namespace std;

// Global Mesh Variables
int nx;
int ny;

int ind(int i, int j) {
        return nx*j + i;
}

int main() {
        cout << "Make␣sure␣that␣there␣is␣a␣data␣directory␣created\n";
```

```
// Number of Timesteps
int nsteps = 400000;

// Mesh Parameters
nx = 256;
ny = 256;
int N = nx*ny;
float H = 1.0;
float dx = H/(nx-1);
float dy = H/(ny-1);

// Specified Parameters
float Ma = 0.1;
float Re = 100.0;

// Dimensional Parameters
float rho_0 = 1.0;
float c = 1.0;
float c_squared = c*c;
float U = Ma*c;
float mu = rho_0*U*H/Re;

// Time Parameters
float sigma = 0.3;
float Re_delta = rho_0*U/mu*min(dx, dy);
float dt = sigma/((1.0 + 2.0/Re_delta)
        *(U/dx + U/dy + c*sqrt(1/dx/dx + 1/dy/dy)));

// Product Coefficients
float a1 = dt/dx;
float a2 = dt/dy;
float a3 = mu*dt/dx/dx;
float a343 = (4.0/3.0)*a3;
float a4 = mu*dt/dy/dy;
float a443 = (4.0/3.0)*a4;
float a5 = mu*dt/(12.0*dx*dy);
float b1 = 0.5*dt/dx;
float b2 = 0.5*dt/dy;

// Initializing Arrays
```

```
float rho[N];
float rho_u[N];
float rho_v[N];
float rho_p[N];
float rho_u_p[N];
float rho_v_p[N];
float u[N];
float v[N];

// Giving Arrays Initial Values
for(int i=0; i<nx; i++){
        for(int j=0; j<ny-1; j++){
                rho[ind(i,j)] = rho_0;
                rho_u[ind(i,j)] = 0.0;
                rho_v[ind(i,j)] = 0.0;
                rho_p[ind(i,j)] = rho_0;
                rho_u_p[ind(i,j)] = 0.0;
                rho_v_p[ind(i,j)] = 0.0;
        }
        rho[ind(i,ny-1)] = rho_0;
        rho_u[ind(i,ny-1)] = U*rho_0;
        rho_v[ind(i,ny-1)] = 0.0;
        rho_p[ind(i,ny-1)] = rho_0;
        rho_u_p[ind(i,ny-1)] = U*rho_0;
        rho_v_p[ind(i,ny-1)] = 0.0;
}

for(int step_index=0; step_index<nsteps; step_index++){

        // Calculate Velocities
        for(int j=0; j<ny; j++){
                for(int i=0; i<nx; i++){
                        u[ind(i,j)] = rho_u[ind(i,j)]/rho[ind(i,j)];
                        v[ind(i,j)] = rho_v[ind(i,j)]/rho[ind(i,j)];
                }
        }



        // Bottom Boundary
        for(int i=1; i<nx-1; i++){
```

```
        // Bottom Boundary
        rho_p[ind(i,0)] = rho[ind(i,0)]
                        - b2*(  -3*rho_v[ind(i,0)]
                                +4*rho_v[ind(i,1)]
                                -1*rho_v[ind(i,2)]);
}

// Interior First
for(int j=1; j<ny-1; j++){
        // Left Boundary
        rho_p[ind(0,j)] = rho[ind(0,j)]
                        - b1*(  -3*rho_u[ind(0,j)]
                                +4*rho_u[ind(1,j)]
                                -1*rho_u[ind(2,j)]);

        for(int i=1; i<nx-1; i++){

                // Mass equation
                rho_p[ind(i,j)] = rho[ind(i,j)]

                - a1*(rho_u[ind(i+1,j)] - rho_u[ind(i,j)])

                - a2*(rho_v[ind(i,j+1)] - rho_v[ind(i,j)]);

                // x momentum equation
                rho_u_p[ind(i,j)] = rho_u[ind(i,j)]

                - a1*(  rho_u[ind(i+1,j)]*u[ind(i+1,j)]
                        + c_squared*rho[ind(i+1,j)]
                        - rho_u[ind(i,j)]*u[ind(i,j)]
                        - c_squared*rho[ind(i,j)])

                - a2*(  rho_u[ind(i,j+1)]*v[ind(i,j+1)]
                        - rho_u[ind(i,j)]*v[ind(i,j)])

                + a343*(u[ind(i+1,j)]
                        - 2*u[ind(i,j)]
                        + u[ind(i-1,j)])

                + a4*(  u[ind(i,j+1)]
                    - 2*u[ind(i,j)]
                    + u[ind(i,j-1)] )

                + a5*(  v[ind(i+1,j+1)]
                    + v[ind(i-1,j-1)]
                    - v[ind(i+1,j-1)]
                    - v[ind(i-1,j+1)] );

                // y momentum equation
                rho_v_p[ind(i,j)] = rho_v[ind(i,j)]

                - a1*(  rho_v[ind(i+1,j)]*u[ind(i+1,j)]
                    - rho_v[ind(i,j)]*u[ind(i,j)])

                - a2*(  rho_v[ind(i,j+1)]*v[ind(i,j+1)]
                    + c_squared*rho[ind(i,j+1)]
                    - rho_v[ind(i,j)]*v[ind(i,j)]
                    - c_squared*rho[ind(i,j)])

                + a3*(  v[ind(i+1,j)]
                    - 2*v[ind(i,j)]
                    + v[ind(i-1,j)])

                + a443*(v[ind(i,j+1)]
                    - 2*v[ind(i,j)]
                    + v[ind(i,j-1)] )

                + a5*(  u[ind(i+1,j+1)]
                    + u[ind(i-1,j-1)]
                    - u[ind(i+1,j-1)]
                    - u[ind(i-1,j+1)] );
            }
            // Right Boundary
            rho_p[ind(nx-1,j)] = rho[ind(nx-1,j)]
                    + b1*(  -3*rho_u[ind(nx-1,j)]
                            +4*rho_u[ind(nx-2,j)]
                            -1*rho_u[ind(nx-3,j)]);
        }
```

```
// Top Boundary
for(int i=1; i<nx-1; i++){
        rho_p[ind(i,ny-1)] = rho[ind(i,ny-1)]
                        + b2*(  -3*rho_v[ind(i,ny-1)]
                                +4*rho_v[ind(i,ny-2)]
                                -1*rho_v[ind(i,ny-3)])
                        - b1*U*(rho[ind(i+1,ny-1)]
                                -rho[ind(i-1,ny-1)]);
        rho_u_p[ind(i,ny-1)] = U*rho_p[ind(i,ny-1)];
}

///////////////////////////////////////////////////////
// CORRECTOR
///////////////////////////////////////////////////////
// Calculate Velocities
for(int j=0; j<ny; j++){
        for(int i=0; i<nx; i++){
                u[ind(i,j)] = rho_u_p[ind(i,j)]/rho_p[ind(i,j)];
                v[ind(i,j)] = rho_v_p[ind(i,j)]/rho_p[ind(i,j)];
        }
}

for(int i=1; i<nx-1; i++){
        // Bottom Boundary
        rho[ind(i,0)] = 0.5*(rho[ind(i,0)] + rho_p[ind(i,0)]
                        - b2*(  -3*rho_v_p[ind(i,0)]
                                +4*rho_v_p[ind(i,1)]
                                -1*rho_v_p[ind(i,2)]));
}

// Interior First
for(int j=1; j<ny-1; j++){
        // Left Boundary
        rho[ind(0,j)] = 0.5*(rho[ind(0,j)] + rho_p[ind(0,j)]
                        - b1*(  -3*rho_u_p[ind(0,j)]
                                +4*rho_u_p[ind(1,j)]
                                -1*rho_u_p[ind(2,j)]));
        for(int i=1; i<nx-1; i++){

                // Mass Equation
```

```
rho[ind(i,j)] = 0.5*(rho[ind(i,j)]
        + rho_p[ind(i,j)]

- a1*(rho_u_p[ind(i,j)] - rho_u_p[ind(i-1,j)])

- a2*(rho_v_p[ind(i,j)] - rho_v_p[ind(i,j-1)]));

// x momentum equation
rho_u[ind(i,j)] = 0.5*(rho_u[ind(i,j)]
        + rho_u_p[ind(i,j)]

- a1*(  rho_u_p[ind(i,j)]*u[ind(i,j)]
        + c_squared*rho_p[ind(i,j)]
        - rho_u_p[ind(i-1,j)]*u[ind(i-1,j)]
        - c_squared*rho_p[ind(i-1,j)])

- a2*(  rho_u_p[ind(i,j)]*v[ind(i,j)]
        - rho_u_p[ind(i,j-1)]*v[ind(i,j-1)])

+ a343*(u[ind(i+1,j)]
        - 2*u[ind(i,j)]
        + u[ind(i-1,j)])

+ a4*(  u[ind(i,j+1)]
        - 2*u[ind(i,j)]
        + u[ind(i,j-1)] )

+ a5*(  v[ind(i+1,j+1)]
        + v[ind(i-1,j-1)]
        - v[ind(i+1,j-1)]
        - v[ind(i-1,j+1)] ));

// y momentum equation
rho_v[ind(i,j)] = 0.5*(rho_v[ind(i,j)]
        + rho_v_p[ind(i,j)]

- a1*(  rho_v_p[ind(i,j)]*u[ind(i,j)]
        - rho_v_p[ind(i-1,j)]*u[ind(i-1,j)])

- a2*(  rho_v_p[ind(i,j)]*v[ind(i,j)]
                            + c_squared*rho_p[ind(i,j)]
                            - rho_v_p[ind(i,j-1)]*v[ind(i,j-1)]
                            - c_squared*rho_p[ind(i,j-1)])

                        + a3*(  v[ind(i+1,j)]
                            - 2*v[ind(i,j)]
                            + v[ind(i-1,j)])

                        + a443*(v[ind(i,j+1)]
                            - 2*v[ind(i,j)]
                            + v[ind(i,j-1)]  )

                        + a5*(  u[ind(i+1,j+1)]
                            + u[ind(i-1,j-1)]
                            - u[ind(i+1,j-1)]
                            - u[ind(i-1,j+1)]  ));
        }
        // Right Boundary
        rho[ind(nx-1,j)] = 0.5*(rho[ind(nx-1,j)]
                            + rho_p[ind(nx-1,j)]
                            + b1*(  -3*rho_u_p[ind(nx-1,j)]
                                    +4*rho_u_p[ind(nx-2,j)]
                                    -1*rho_u_p[ind(nx-3,j)]));
}

// Top Boundary
for(int i=1; i<nx-1; i++){
        rho[ind(i,ny-1)] = 0.5*(rho[ind(i,ny-1)]
                            + rho_p[ind(i,ny-1)]
                            + b2*(  -3*rho_v[ind(i,ny-1)]
                                    +4*rho_v[ind(i,ny-2)]
                                    -1*rho_v[ind(i,ny-3)])
                            - b1*U*(rho[ind(i+1,ny-1)]
                                    -rho[ind(i-1,ny-1)]));
        rho_u[ind(i,ny-1)] = U*rho[ind(i,ny-1)];
}

// Printer
ofstream data_file_object;
char data_buffer[15];
```

```
                sprintf(data_buffer, "data/%06d.csv", step_index);
                data_file_object.open(data_buffer);
                for(int j=0; j<ny; j++){
                        for(int i=0; i<nx; i++){
                                data_file_object << i*dx << "\t"
                                        << j*dy << "\t"
                                        << rho[ind(i,j)] << "\t"
                                        << rho_u[ind(i,j)] << "\t"
                                        << rho_v[ind(i,j)] << "\n";
                        }
                }
                data_file_object.close();
                cout << "Completed Iteration: " << step_index << "\n";
        }


        cout << "Completed Simulation\n";
}
```

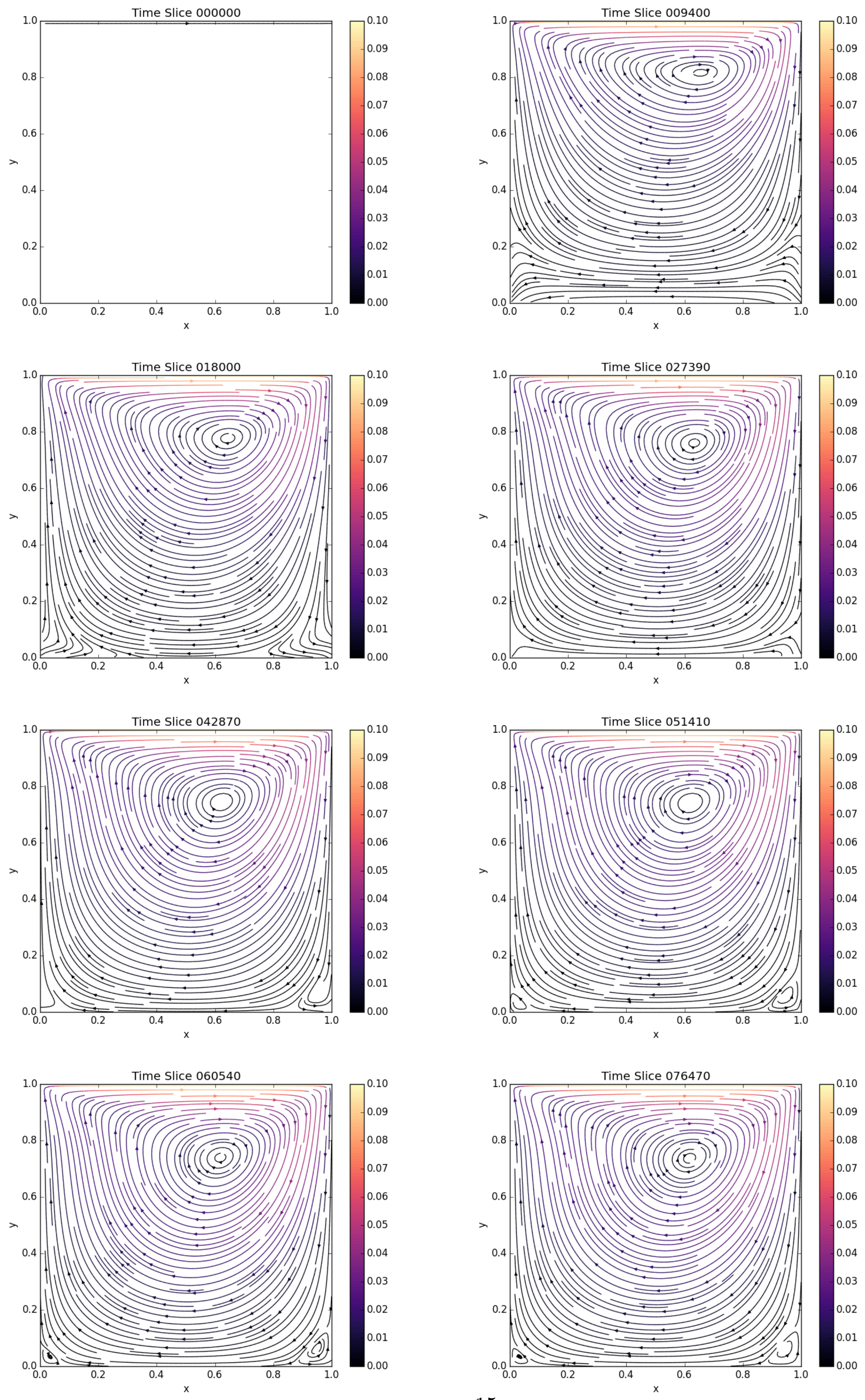

Figure 2: Streamlines at different times.

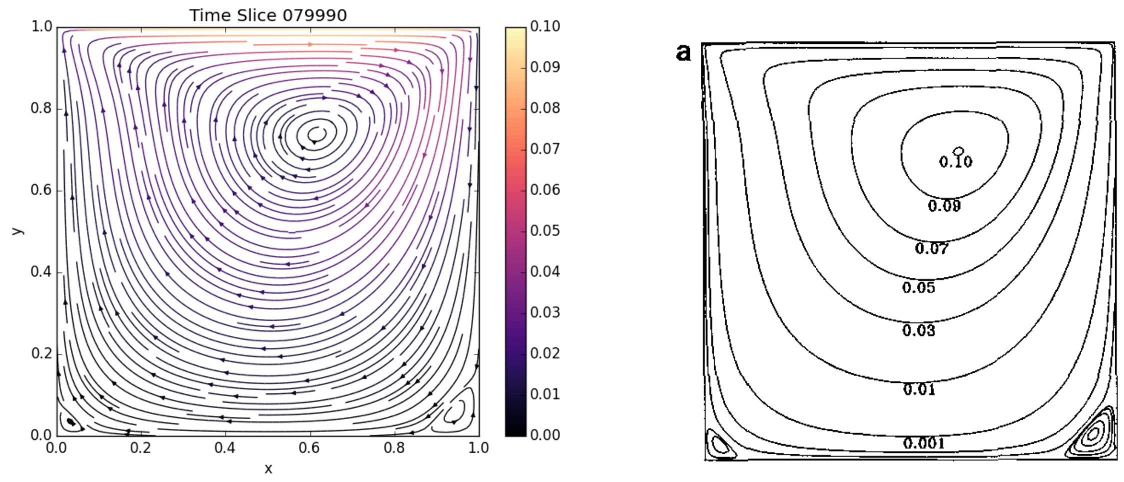

Figure 3: Steady state. Comparison with Hou et al. (1995)