Numerical Analysis

Of the Lid-Driven Cavity Problem

MEMS1055, Computer Aided Analysis in Transport Phenomena
Dr. Peyman Givi
Seth Strayer
4/19/19

Contents

1 Introduction

The lid-driven cavity problem has long been considered a benchmark problem for validating numerical solutions to the viscous, incompressible Navier-Stokes equations. In this problem, wall boundaries surround the entire computational region, while the top wall drives the flow via uniform translation. The remaining three walls are defined by the no-slip boundary condition, i.e., the velocity field $\vec{v} = 0$. This problem has relatively simple, well-known boundary conditions which are depicted in Figure 1.[1]

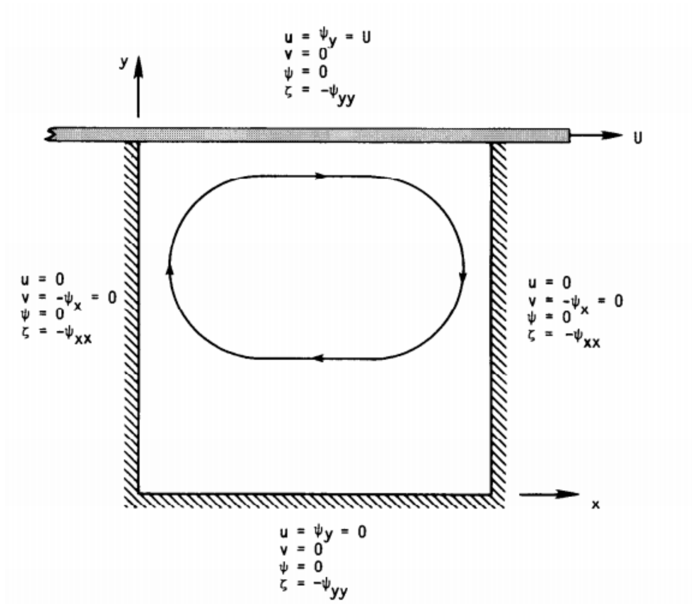

Figure 1: The Lid-Driven Cavity Problem - Figure 9.3 in Tannehill, Anderson, and Pletcher

Standard test cases have been performed by countless investigators. Among the most detailed of which source from Ghia et al. (1982)[3], as it includes tabular results for various grid sizes and increasing Reynolds numbers.[1,2] It is also documented that this problem is often susceptible to instabilities at higher Reynolds numbers. Thus, a standard test condition of

$$Re_L = \frac{UL}{\nu} = 100 \tag{1}$$

is frequently chosen when performing test comparisons.[1] In this report, we will consider the voricity-stream function approach, which is among the most popular methods for solving the 2-D viscous, incompressible Navier-Stokes equations. This approach replaces the standard velocity components u and v with vorticity ω and stream function ψ.[1] Full derivations of this method are well-documented and thus will be omitted from this report. Our main goal is to perform the numerical simulation for varying Reynolds numbers of $Re_L = 10, 100, 1000$, plot the vorticity, stream function, and velocity field, and compare our results with those in literature.

2 Formulation

2.1 Derivation of the Vorticity-Stream Function Approach

The vorticity ω is well-defined as

$$\omega = \nabla \times V = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \tag{2}$$

Also, the stream function ψ is defined by the equations

$$\frac{\partial \psi}{\partial y} = u \tag{3}$$

$$\frac{\partial \psi}{\partial x} = -v \tag{4}$$

Note that these equations automatically satisfy the incompressibility conditions. By substituting Equations 3 and 4 into Equation 2, we obtain

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\omega \tag{5}$$

Equation 5 is an Elliptic PDE known as the *Poisson equation*. This equation can be solved for numerically using the *successive over-relaxation* (SOR) method. By nondimensionalizing and manipulation of the Navier-Stokes equations as discussed in various sources, we can obtain the *vorticity transport equation*:

$$\frac{\partial \omega}{\partial t} + u\frac{\partial \omega}{\partial x} + v\frac{\partial \omega}{\partial y} = \frac{1}{Re_L}\left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}\right) \tag{6}$$

Equations 5 and 6 represent the Navier-Stokes equations in vorticity-stream function form. Note that Equation 6 is a Parabolic PDE that can be solved for using methods such as FTCS Explicit, FTCS Implicit, Upwind, and MacCormack. In our simulation, the FTCS Explicit method will be used to solve for the vorticity. Note that this method is stable under certain conditions for the diffusive case, but is unconditionally unstable for the inviscid case. Thus, if the inviscid case were to be considered, another method would have to be employed.

Given the boundary conditions outlined in Figure 1 and the solving methods as described above, we may solve these equations sequentially using a time-marching procedure, which is described in the following steps:[1]

1. Specify initial values for ω and ψ at time $t = 0$.
2. Solve the vorticity transport equation for ω at each interior grid point at time $t + \Delta t$.
3. Iterate for new ψ values at all points by solving the Poisson equation using new ω at interior points.
4. Find the velocity components from $u = \psi_y$ and $v = -\psi_x$.
5. Determine the values of ω on the boundaries using ψ and ω values at interior points.
6. Return to Step 2 if the solution has not converged.

3 Results

The MATLAB scrip used to implement the iterative solving algorithm as described in Section 2.1 is "Project_5.m". Full code listings for this script are provided in the Appendix, Section 5.2. Stream function and vorticity plots for various values of Re_L are presented in the following sections.

3.1 $Re_L = 10$

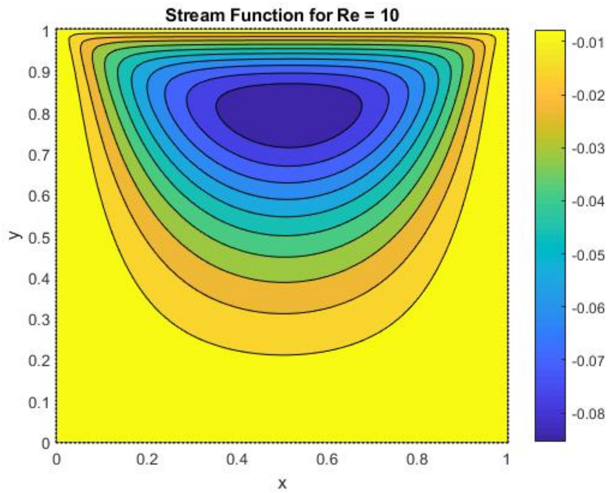

Figure 2: Stream function for $Re_L = 10$

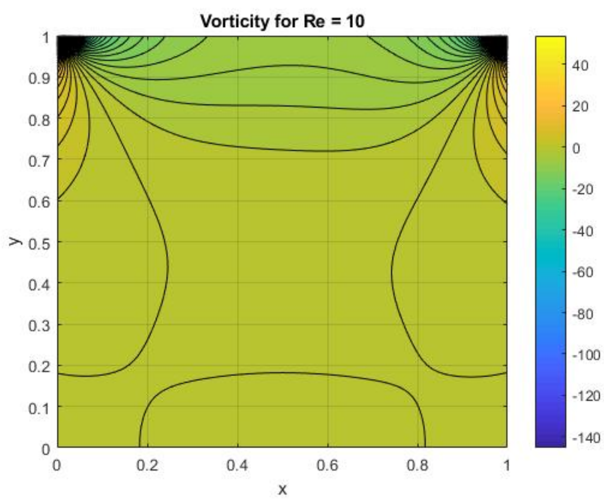

Figure 3: Vorticity for $Re_L = 10$

3.2 $Re_L = 100$

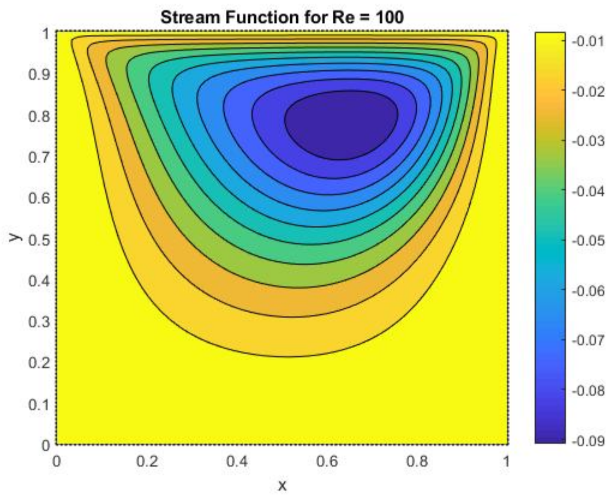

Figure 4: Stream function for $Re_L = 100$

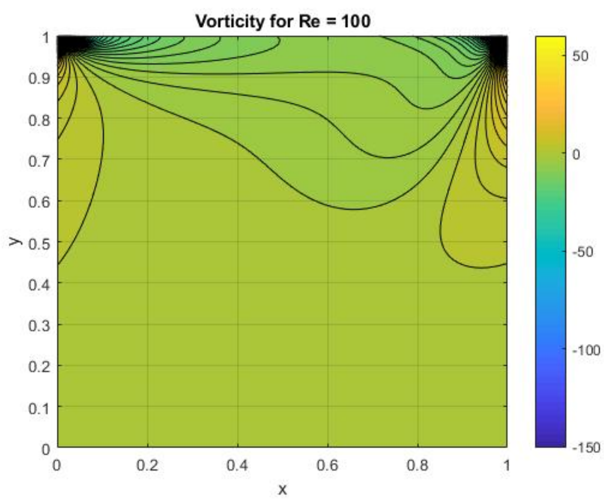

Figure 5: Vorticity for $Re_L = 100$

3.3 $Re_L = 1000$

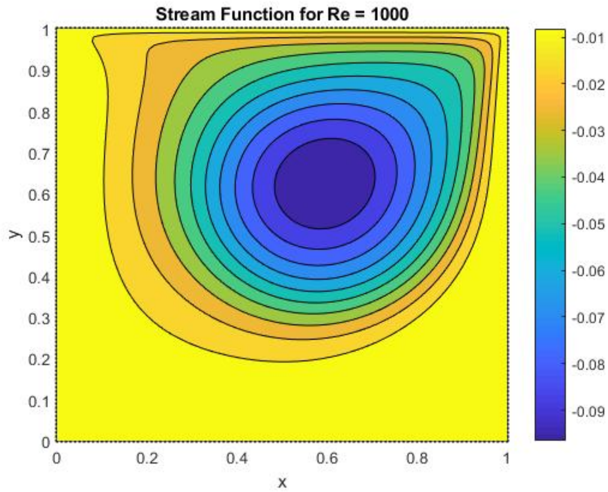

Figure 6: Stream function for $Re_L = 1000$

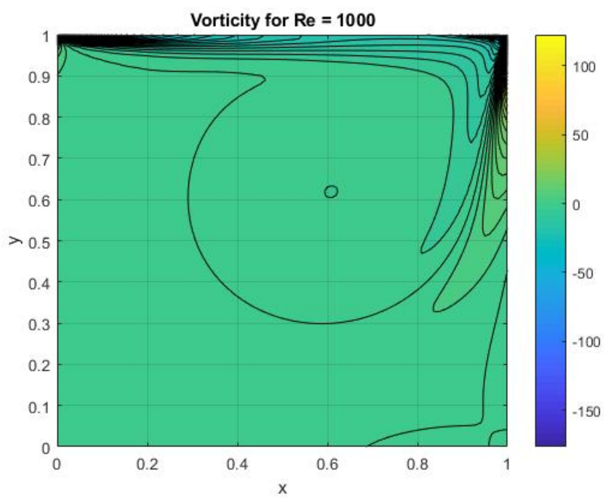

Figure 7: Vorticity for $Re_L = 1000$

4 Conclusion

In this report we were able to calculate numerical solutions to the well-known lid-driven cavity problem for varying Reynolds numbers. Stream function and vorticity plots for Reynolds numbers of $Re_L = 10$, 100, and 1000 were provided in Section 3, and these plots were loosely verified with the results of those obtained from the works of Ghia et al. (1982)[3]. The solutions were also validated by providing semilog plots of the residual error vs. the number of iterations. These plots are provided in Figures 9, 11, and 13 in the Appendix, Section 5.1. The exponential decrease in error with the number of iterations indicates that our computational solution is stable, consistent, and thus yields a valid solution. Overall, we were able to apply the vorticity-stream function approach to solve this well-known problem and are satisfied with our results. Further investigation could be perfmormed by increasing the Reynolds number to values $Re_L > 1000$. These trials were not performed in this report due to the instabilities present with this method for large Re_L and the excessively long run-times associated with refining the mesh grid size, which is necessary to achieve convergence for large Re_L.

5 Appendix

5.1 Additional Figures

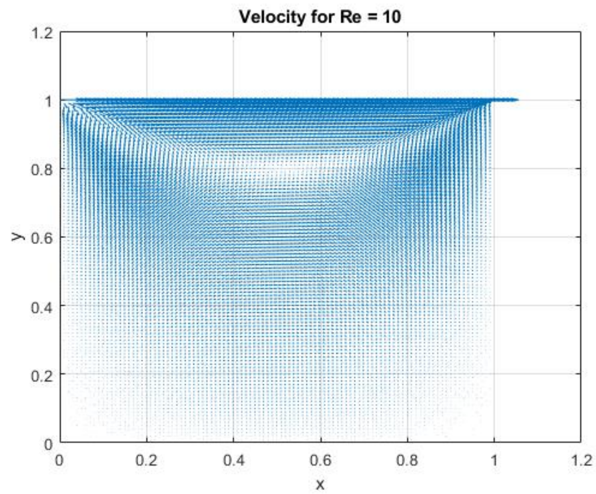

Figure 8: Velocity Field for $Re_L = 10$

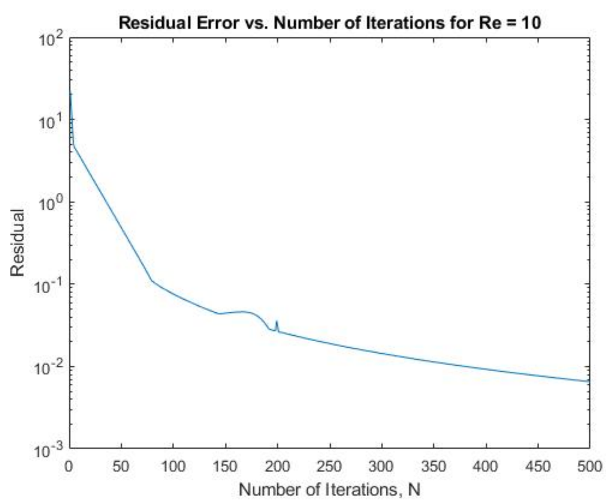

Figure 9: Residual Behavior for $Re_L = 10$

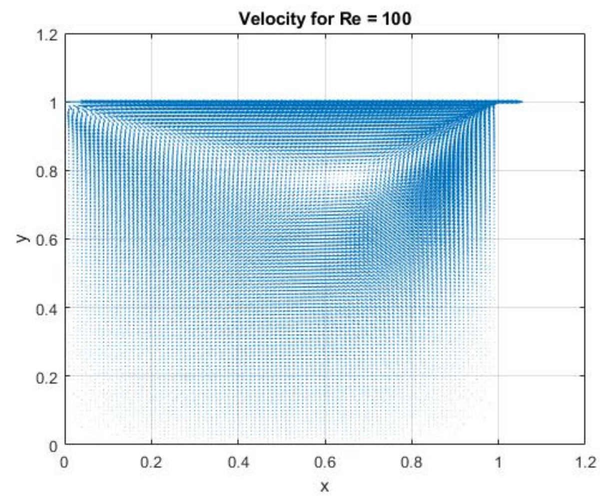

Figure 10: Velocity Field for $Re_L = 100$

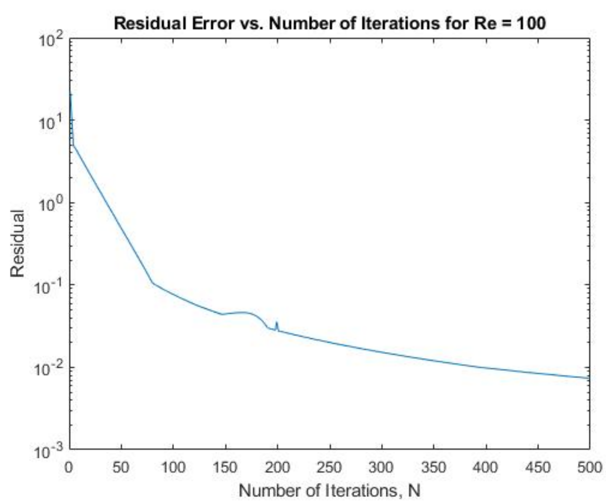

Figure 11: Residual Behavior for $Re_L = 100$

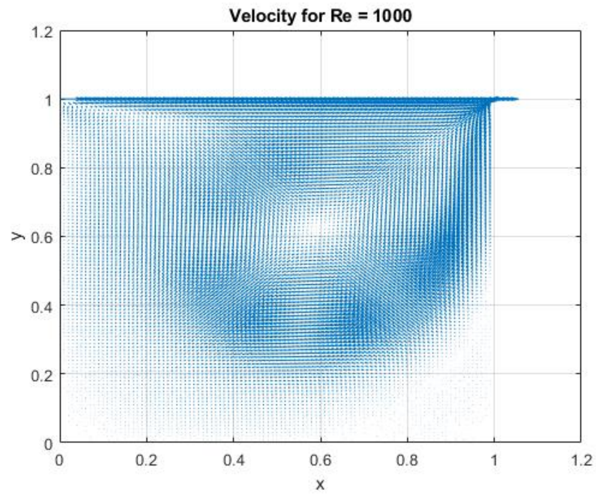

Figure 12: Velocity Field for $Re_L = 1000$

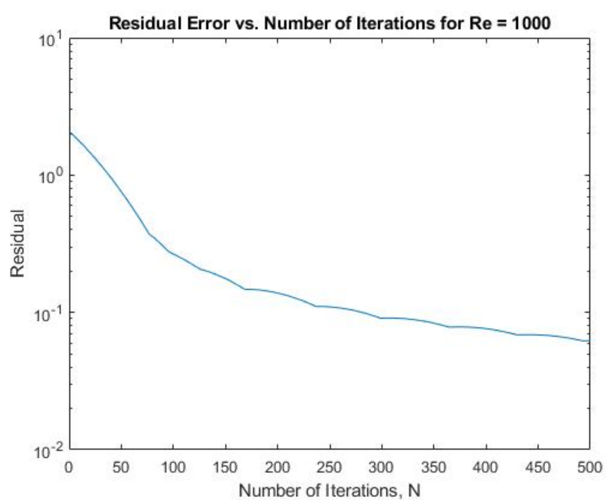

Figure 13: Residual Behavior for $Re_L = 1000$

5.2 Code Listings

```
% Seth Strayer
% MEMS1055, Computer Aided Analysis in Transport Phenomena
% Project 5
% Dr. Peyman Givi
% 4/19/19

% This code is used to calculate a numerical solution to the Lid-Driven
% Square Cavity Flow Problem.

clear;  clc;

% defining parameters
L = 1;                  % length of any side
U = 1;                  % plate velocity
nu = 1e-3;              % viscosity
Re = (U*L)/nu;          % Reynolds Number
h = 0.01;               % grid spacing
dt = 0.001;             % time spacing
M = L/h + 1;            % total grid points in x
N = L/h + 1;            % total grid points in y
xi = linspace(0, L, M); % array of x grid points
yi = linspace(0, L, N); % array of y grid points
tol = 1e-3;             % tolerance
converge = 0;           % convergence flag
iteration = 0;          % iteration
Beta = 1.95;            % relaxation factor
CFL = (U*dt)/h;         % CFL number
F = (nu*dt)/h^2;        % Fourier number

if CFL^2 <= 2*F && CFL <= 1 && F<= 1/2
    stable = 1;
else
    stable = 0;
end

% creating output file used to write residual data
fid = fopen('it_data.txt', 'wt');

% defining grid matrix
A = zeros(M, N+1);

% assigning values to grid matrix
for i = 1:M
    for j = 1:N
        A(i, j+1) = yi(j);
    end
    A(i, 1) = xi(i);
end
```

Figure 14: MATLAB Script (1)

```

% declaring X and Y matrices used to plot grid points
X = zeros(M, N);
Y = zeros(M, N);

% assigning values to grid matrix
for i = 1:M
    for j = 1:N
        X(i, j) = A(i, 1);
        Y(i, j) = A(1, j+1);
    end
end

% plotting grid points
plot(X, Y, '.k');

% definining stream function, voricity, and velocity matrices
sf = zeros(M, N);
w = zeros(M, N);
u = zeros(M, N);
u(:, N) = 1;
v = zeros(M, N);

% declaring dummy matrices to store old values
oldsf = sf;
oldw = w;

% starting iterative solving loop
while converge == 0
    % update interior streamfunction values through SOR iteration
    for i = 2:M-1
        for j = 2:N-1
            sf(i, j) = 1/4*Beta*(sf(i+1, j) + sf(i-1, j) + sf(i, j+1) + ...
                sf(i, j-1) + h^2*w(i, j)) + (1-Beta)*(sf(i, j));
        end
    end

    % update boundary conditions for vorticity
    for i = 2:M-1
        for j = 2:N-1
            w(i, 1) = -2.0*sf(i, 2)/h^2;     % bottom wall
            w(i, N) = -2.0*sf(i, N-1)/h^2 - (2*U)/(h);  % top wall
            w(1, j) = -2.0*sf(2, j)/h^2;     % left wall
            w(M, j) = -2.0*sf(M-1, j)/h^2;  % right wall
        end
    end

    % update vorticity matrix and solve for maximum residual error
```

Figure 15: MATLAB Script (2)

```matlab
for i = 2:M-1
    for j = 2:N-1
        % solving for the vorticity at all interior grid points
        w(i, j) = w(i, j) - (dt/(4*h^2))*((sf(i, j+1) - sf(i, j-1)) ...
            *(w(i+1, j) - w(i-1, j))) + (dt/(4*h^2))*((sf(i+1, j) - ...
            sf(i-1, j)) * (w(i, j+1) - w(i, j-1))) + (dt/(Re*h^2)) ...
            *(w(i+1, j) + w(i-1, j) + w(i, j+1) + w(i, j-1) - 4*w(i, j));


        % calculating residual error for each point
        R(i, j) = abs(w(i, j) - oldw(i, j));

        % finding maximum residual
        max_R_array = max(R);

    end
end

% calculate velocity field
for i = 2:M-1
    for j = 2:N-1
        u(i,j)=(sf(i,j+1)-sf(i,j-1))/(2*h);
        v(i,j)=(sf(i+1,j)-sf(i-1,j))/(2*h);
    end
end

% finding the maximum residual error present at any given point
max_R = max_R_array(1);
for j = 2:N-1
    if max_R_array(j) > max_R
        max_R = max_R_array(j);
    end
end

% if tolerance is met, exit the loop
if max_R < tol
    converge = 1;
    continue;
end

% redefining old matrices and updating iteration
oldsf = sf;
oldw = w;
iteration = iteration + 1;

% printing results to file
fprintf(fid, '%d   %d\n', [iteration; max_R]);

```

Figure 16: MATLAB Script (3)

```
    % printing the maximum residual error for each iteration
    max_R

end

% plotting stream function
hold on;
figure(1);
contourf(X, Y, sf, 10);view(2);
shading interp;
grid on;
xlabel('x');
ylabel('y');
title(['Stream Function for Re = ', num2str(Re)]);
colorbar;

% plotting vorticity
hold on;
figure(2);
contourf(X, Y, w, 100);view(2);
shading interp;
grid on;
xlabel('x');
ylabel('y');
title(['Vorticity for Re = ', num2str(Re)]);
colorbar;

% plotting velocity field
hold on;
figure(3);
scale = 4;
quiver(X, Y, u, v, scale);view(2);
shading interp;
grid on;
xlabel('x');
ylabel('y');
title(['Velocity for Re = ', num2str(Re)]);

% extracting residual data from file
load it_data.txt
iter_data = it_data(:, 1);
R_data = it_data(:, 2);

% defining parameters for plotting
startit = 1;
numit = 500;

% creating plot of residual error vs. number of iterations
```

Figure 17: MATLAB Script (4)

```
% creating plot of residual error vs. number of iterations
figure(4);
semilogy(iter_data(startit:numit), R_data(startit:numit));
title(['Residual Error vs. Number of Iterations for Re = ', num2str(Re)]);
xlabel('Number of Iterations, N');
ylabel('Residual');
hold on;
```

Figure 18: MATLAB Script (5)

5.3 References

[1] Tannehill, John C., et al. Computational Fluid Mechanics and Heat Transfer. 2nd ed., Taylor & Francis, 1997.

[2] "Lid-Driven Cavity Problem." CFD Online, Various Sponsors, 9 Apr. 2013, www.cfd-online.com/Wiki/Lid-driven_cavity_problem.

[3] Ghia, U, et al. "High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method." Journal of Computational Physics, vol. 48, no. 3, 1982, pp. 387–411., doi:10.1016/0021-9991(82)90058-4.

List of Figures