California Polytechnic State University, Pomona

Aerospace Engineering Department

Course: ARO 4180-01 Computational Fluid Dynamics

Lid-Driven Cavity Problem: Incompressible, Viscous, Rotational Flow

By: Eliot Khachi

Instructor: Dr. Frank Chandler

Table of Contents

Problem Statement

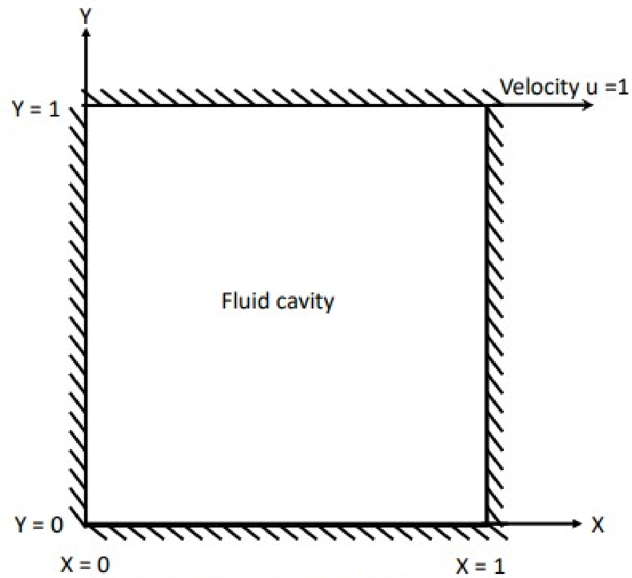
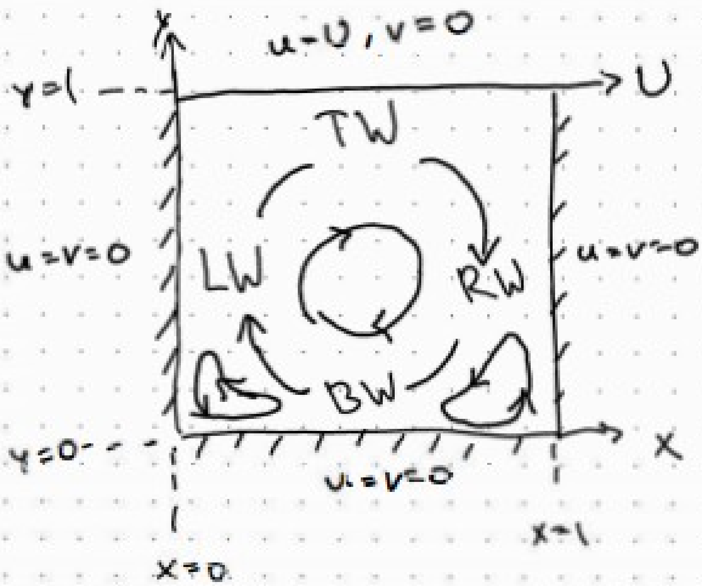


Figure 1. Driven Cavity Problem

A figure of the Lid-Driven Cavity problem is shown above. A square cavity of side-length equal to 1 has walls at locations x = 0, x = 1, y = 0, and y = 1. Its lid is located at the y = 1 wall that is moving to the right with a velocity of u = 1. Fluid inside the cavity is subject to incompressible, viscous, and rotational flow. Consider fluid with Reynold's Numbers of 10, 100, and 400, and iteratively solve an *n* x *n* grid for *n* equal to 8, 16, and 32. Using the Alternating Direction Implicit (ADI) Method solve the parabolic vorticity transport equation, and using the Point Successive Over Relaxation (PSOR) Method solve the elliptic stream function equation.

CFD Final Project

Lid-Driven Cavity Problem



Nomenclature

Top Wall: TW – (x,1)
Bottom Wall: BW – (x,0)
Left Wall: LW – (0,y)
Right Wall: RW – (1,y)

velocity field $\vec{u} = u\hat{i} + v\hat{j}$

Governing Equations:

Vorticity Transport Equation (VTE)

$$\frac{\partial \xi}{\partial t} + u\frac{\partial \xi}{\partial x} + v\frac{\partial \xi}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 \xi}{\partial x^2} + \frac{\partial^2 \xi}{\partial y^2}\right) \quad (8\text{-}94)$$

← Parabolic

Steady-State: $$\underline{u\frac{\partial \xi}{\partial x} + v\frac{\partial \xi}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 \xi}{\partial x^2} + \frac{\partial^2 \xi}{\partial y^2}\right)}$$

← Elliptic

Stream Function Equation (SFE)

$$\xi = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}, \text{ where } v = -\frac{\partial \psi}{\partial x} \ \& \ u = \frac{\partial \psi}{\partial y}$$

$$\Rightarrow \underline{\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\xi} \quad \text{or} \quad \underline{\nabla^2 \psi = -\xi} \quad (8\text{-}95)$$

← Elliptic

Sequential Solution Procedure

1.) Initialize ξ & ψ @ $t=0$ given $\vec{u}$ @ $t=0$, or make an initial guess (as for 3-D problems)

2.) Solve VTE @ each interior grid point for $t=(n+1)\Delta t$
↳ obtain values of ξ using ADI method

3.) Solve SFE @ each interior grid point using ξ obtained in step 2) using PSOR method

4.) Find $u = \frac{\partial \psi}{\partial y}$ & $v = -\frac{\partial \psi}{\partial x}$

5.) Determine ξ on boundaries using ψ & ξ values at interior points.

6.) Check for convergence. If not, return to step 2

MatLab Implementation

While error < ERROR_TOL
- Solve Vorticity transport by ADI
  - X-sweep
  - Y-sweep
- Solve stream function by PSOR
- Solve for velocity fields using central differencing
  - $u = \frac{\partial \psi}{\partial y} \Rightarrow u_{i,j} = \frac{\psi_{i,j+1} - \psi_{i,j-1}}{2\Delta y}$
  - $v = -\frac{\partial \psi}{\partial x} \Rightarrow v_{i,j} = -\frac{\psi_{i+1,j} - \psi_{i-1,j}}{2\Delta x}$

Finite-Difference Derivation

First Solve the Vorticity Transport Eqn by ADI

$$u \frac{\partial \xi}{\partial x} + v \frac{\partial \xi}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 \xi}{\partial x^2} + \frac{\partial^2 \xi}{\partial y^2}\right) \quad (8\text{-}99)$$

ADI Formulation: (8.8.1 p. 340)

$$A_x \xi^{n+\frac{1}{2}}_{i-1,j} + B_x \xi^{n+\frac{1}{2}}_{i,j} + C_x \xi^{n+\frac{1}{2}}_{i+1,j} = D_x \quad (8\text{-}101)$$

$$A_y \xi^{n+1}_{i,j-1} + B_y \xi^{n+1}_{i,j} + C_y \xi^{n+1}_{i,j+1} = D_y \quad (8\text{-}102)$$

where

$$A_x = -\frac{1}{2}\left(\frac{1}{2}c_x + d_x\right) \qquad A_y = -\frac{1}{2}\left(\frac{1}{2}c_y + d_y\right)$$

$$B_x = 1 + d_x \qquad B_y = 1 + d_y$$

$$C_x = \frac{1}{2}\left(\frac{1}{2}c_x - d_x\right) \qquad C_y = \frac{1}{2}\left(\frac{1}{2}c_y - d_y\right)$$

and

$$D_x = \frac{1}{2}\left(\frac{1}{2}c_y + d_y\right)\xi^{n}_{i,j-1} + \left(1 - d_y\right)\xi^{n}_{i,j} + \frac{1}{2}\left(-\frac{1}{2}c_y + d_y\right)\xi^{n}_{i,j+1}$$

$$D_y = \frac{1}{2}\left(\frac{1}{2}c_x + d_x\right)\xi^{n+\frac{1}{2}}_{i-1,j} + \left(1 - d_x\right)\xi^{n+\frac{1}{2}}_{i,j} + \frac{1}{2}\left(-\frac{1}{2}c_x + d_x\right)\xi^{n+\frac{1}{2}}_{i+1,j}$$

where

$$c_x = u\frac{\Delta t}{\Delta x}, \quad c_y = v\frac{\Delta t}{\Delta y}$$

$$d_x = \frac{1}{Re}\frac{\Delta t}{(\Delta x)^2}, \quad d_y = \frac{1}{Re}\frac{\Delta t}{(\Delta y)^2}$$

Solve Stream Function By PSOR

$$\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} = -\xi$$

Point Successive Over-Relaxation Formulation:

$$\psi_{i,j}^{k+1} = (1-\omega)\psi_{i,j}^{k} + \frac{\omega}{2(1+\beta^2)}\left[\xi_{i,j}^{n+1}(\Delta x)^2 + \psi_{i+1,j}^{k} + \psi_{i-1,j}^{k+1} + \beta^2\left(\psi_{i,j+1}^{k} + \psi_{i,j-1}^{k+1}\right)\right]$$

Where ω is the relaxation parameter: $0<\omega<1$ - under-relaxed
$1<\omega<2$ - over-relaxed

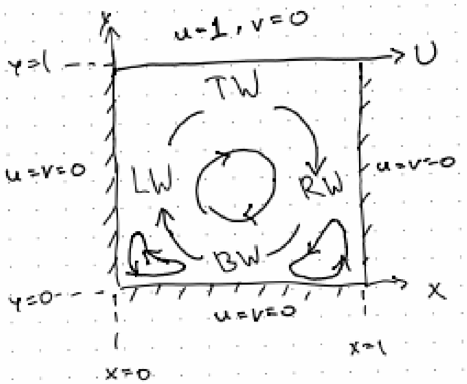
Central-Difference Formulations For Velocity

$$u = \frac{\partial\psi}{\partial y} \Rightarrow u_{i,j} = \frac{\psi_{i,j+1} - \psi_{i,j-1}}{2\Delta y}$$

$$v = -\frac{\partial\psi}{\partial x} \Rightarrow v_{i,j} = -\frac{\psi_{i+1,j} - \psi_{i-1,j}}{2\Delta x}$$

Initial and Boundary Conditions Derivation

Boundary Conditions:



Taylor Series Expansions

BW: $\psi_{i,j+1} = \psi_{i,j} + \Delta y \left.\frac{\partial \psi}{\partial y}\right|_{i,j} + \frac{(\Delta y)^2}{2} \left.\frac{\partial^2 \psi}{\partial y^2}\right|_{i,j} + O[(\Delta y)^3]$

TW: $\psi_{i,j-1} = \psi_{i,j} - \Delta y \left.\frac{\partial \psi}{\partial y}\right|_{i,j} + \frac{(\Delta y)^2}{2} \left.\frac{\partial^2 \psi}{\partial y^2}\right|_{i,j} + O[(\Delta y)^3]$

LW: $\psi_{i+1,j} = \psi_{i,j} + \Delta x \left.\frac{\partial \psi}{\partial x}\right|_{i,j} + \frac{(\Delta x)^2}{2} \left.\frac{\partial^2 \psi}{\partial x^2}\right|_{i,j} + O[(\Delta x)^3]$

RW: $\psi_{i-1,j} = \psi_{i,j} - \Delta x \left.\frac{\partial \psi}{\partial x}\right|_{i,j} + \frac{(\Delta x)^2}{2} \left.\frac{\partial^2 \psi}{\partial x^2}\right|_{i,j} + O[(\Delta x)^3]$

* Lid-Driven Cavity boundary velocities *

1st Order Approx.

Bottom Wall (BW): $\partial^2\psi/\partial x^2 = 0 \Rightarrow \xi_{i,j} = -\left.\frac{\partial^2 \psi}{\partial y^2}\right|_{i,j} = \frac{2(\psi_{i,j} - \psi_{i,j+1})}{\Delta y^2} + \frac{2\cancelto{0}{u_{i,j}}}{\Delta y}$

Top Wall (TW): $\partial^2\psi/\partial x^2 = 0 \Rightarrow \xi_{i,j} = -\left.\frac{\partial^2 \psi}{\partial y^2}\right|_{i,j} = \frac{2(\psi_{i,j} - \psi_{i,j-1})}{\Delta y^2} \; \frac{2\cancelto{1}{u_{i,j}}}{\Delta y}$

Left Wall (LW): $\partial^2\psi/\partial y^2 = 0 \Rightarrow \xi_{i,j} = -\left.\frac{\partial^2 \psi}{\partial x^2}\right|_{i,j} = \frac{2(\psi_{i,j} - \psi_{i+1,j})}{\Delta x^2} - \frac{2\cancelto{0}{v_{i,j}}}{\Delta x}$

Right Wall (RW): $\partial^2\psi/\partial y^2 = 0 \Rightarrow \xi_{i,j} = -\left.\frac{\partial^2 \psi}{\partial x^2}\right|_{i,j} = \frac{2(\psi_{i,j} - \psi_{i-1,j})}{\Delta x^2} + \frac{2\cancelto{0}{v_{i,j}}}{\Delta x}$

Program Listing


```
clear all;
clc;
N = [8 16 32];
Re = [10 100 400];
for l=1:length(N)
for r =1:length(Re)
%% Given
t_f = 3; % final time
dt = 0.01; % time step
u0 = 1; % velocity of lid
L_x = 1; % x-length of cavity
L_y = 1; % y-length of cavity
N_x = N(l); % # of x-nodes
N_y = N(l); % # of y-nodes
dx = L_x/N_x; % spatial-x step
dy = L_y/N_y; % spatial-y step
beta = dx/dy;
alpha = 1/(2*(1+beta^2));
w_PSOR = 0.9; % relaxation parameter for PSOR
ERROR_TOL = 0.05; % error tolerance
Re_r = Re(r); % Reynold's Number
%% Define Initial Conditions
u = zeros(N_x,N_y); % Define u velocity field
u(1, :) = 1; % initialize lid velocity
v = zeros(N_x,N_y); % Define v velocity field
% Initialize tridiagonal Matrix
tri_x = zeros(N_x,N_y);
tri_y = zeros(N_x,N_y);
% Initialize vorticity matrices
w_x = zeros(N_x, N_y); % X-sweep
w_y = zeros(N_x, N_y); % Y-sweep
w = zeros(N_x, N_y); % final vorticity matrix
% Initialize stream-function matrix
psi = zeros(N_x, N_y);
%% Boundary Conditions
w_x(:,1) = 2*(psi(:,1)-psi(:,2))/dy^2;
w_x(:,end) = 2*(psi(:,end)-psi(:,end-1))/dx^2;
w_x(end,:) = 2*(psi(end,:)-psi(end-1,:))/dx^2;
w_x(1,:) = 2*(psi(1,:)-psi(2,:))/dy^2 - 2*u0/dy;

w_y(:,1) = 2*(psi(:,1)-psi(:,2))/dy^2;
w_y(:,end) = 2*(psi(:,end)-psi(:,end-1))/dx^2;
w_y(end,:) = 2*(psi(end,:)-psi(end-1,:))/dx^2;
w_y(1,:) = 2*(psi(1,:)-psi(2,:))/dy^2 - 2*u0/dy;

w(:,1) = 2*(psi(:,1)-psi(:,2))/dy^2;
w(:,end) = 2*(psi(:,end)-psi(:,end-1))/dx^2;
w(end,:) = 2*(psi(end,:)-psi(end-1,:))/dx^2;
w(1,:) = 2*(psi(1,:)-psi(2,:))/dy^2 - 2*u0/dy;
```

```
% Define coefficents for tridiagonal matrix
d_x = 1/Re_r*dt/dx^2;
d_y = 1/Re_r*dt/dy^2;
B_x = 1 + d_x;
B_y = 1 + d_y;

n = 1; % iteration level
t = 0; % current time
current_error = 1;
while t < t_f%current_error > ERROR_TOL % loop until error is within the tolerance
    %% Boundary Conditions at next iteration level
    % Stream-Function
    psi(:,1, n+1) = psi(:,1, n);
    psi(1,:, n+1) = psi(1,:, n);
    psi(:,end, n+1) = psi(:,end, n);
    psi(end,:, n+1) = psi(end,:, n);

    % Velocity Fields
    u(1,:, n+1) = u(1,:,n);
    u(end,:, n+1) = u(end,:,n);
    u(:,1, n+1) = u(:,1,n);
    u(:,end, n+1) = u(:,end,n);

    v(1,:, n+1) = v(1,:,n);
    v(end,:, n+1) = v(end,:,n);
    v(:,1, n+1) = v(:,1,n);
    v(:,end, n+1) = v(:,end,n);

    % Vorticity Matrices
    w_x(:,1,n+1) = 2*(psi(:,1,n+1)-psi(:,2,n+1))/dy^2;
    w_x(:,end,n+1) = 2*(psi(:,end,n+1)-psi(:,end-1,n+1))/dx^2;
    w_x(end,:,n+1) = 2*(psi(end,:,n+1)-psi(end-1,:,n+1))/dx^2;
    w_x(1,:,n+1) = 2*(psi(1,:,n+1)-psi(2,:,n+1))/dy^2 - 2*u0/dy;

    w_y(:,1,n+1) = 2*(psi(:,1,n+1)-psi(:,2,n+1))/dy^2;
    w_y(:,end,n+1) = 2*(psi(:,end,n+1)-psi(:,end-1,n+1))/dx^2;
    w_y(end,:,n+1) = 2*(psi(end,:,n+1)-psi(end-1,:,n+1))/dx^2;
    w_y(1,:,n+1) = 2*(psi(1,:,n+1)-psi(2,:,n+1))/dy^2 - 2*u0/dy;

    w(:,1,n+1) = 2*(psi(:,1,n+1)-psi(:,2,n+1))/dy^2;
    w(:,end,n+1) = 2*(psi(:,end,n+1)-psi(:,end-1,n+1))/dx^2;
    w(end,:,n+1) = 2*(psi(end,:,n+1)-psi(end-1,:,n+1))/dx^2;
    w(1,:,n+1) = 2*(psi(1,:,n+1)-psi(2,:,n+1))/dy^2 - 2*u0/dy;

    %% Solve Vorticity-Transport Equation by ADI Method
    % 1st define tridiagonal matrices' elements
    for i = 1:N_y
        for j = 1:N_x
```

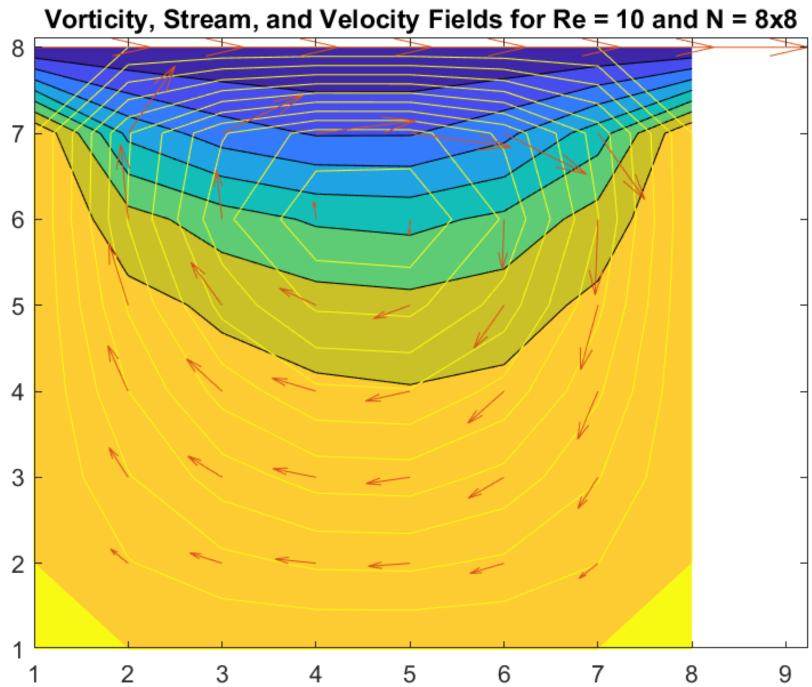
```
            c_x(i,j) = u(i,j,n)*dt/dx;
            A_x(i,j) = -1/2*(1/2*c_x(i,j) + d_x);
            C_x(i,j) = 1/2*(1/2*c_x(i,j) - d_x);
            c_y(i,j) = v(i,j,n)*dt/dy;
            A_y(i,j) = -1/2*(1/2*c_y(i,j) + d_y);
            C_y(i,j) = 1/2*(1/2*c_y(i,j) - d_y);
            if i == j + 1 % Lower Diagonal
                tri_x(i,j) = A_x(i,j);
                tri_y(i,j) = A_y(i,j);
            elseif i == j % Main Diagonal
                tri_x(i,j) = B_x;
                tri_y(i,j) = B_y;
            elseif i == j - 1 % Upper Diagonal
                tri_x(i,j) = C_x(i,j);
                tri_y(i,j) = C_y(i,j);
            end
        end
    end
    % Perform X-SWEEP
    for i = 2:N_y-1 % Iterate bottom to top
        D_x = [];
        for j = 2:N_x-1 % Iterate left to right
            % initialize D matrix of "Ax = D"
            if j == 2
                D_ij = (c_y(i,j)/2+d_y)/2 * w(i,j-1,n) + (1-d_y)*w(i,j,n) + (-c_y(i,
j)/2+d_y)/2*w(i,j+1,n) - A_x(i,j)*w(i,j-1,n);
            elseif j == N_x-1
                D_ij = (c_y(i,j)/2+d_y)/2 * w(i,j-1,n) + (1-d_y)*w(i,j,n) + (-c_y(i,
j)/2+d_y)/2*w(i,j+1,n) - C_x(i,j)*w(i,j+1,n);
            else
                D_ij = (c_y(i,j)/2+d_y)/2 * w(i,j-1,n) + (1-d_y)*w(i,j,n) + (-c_y(i,
j)/2+d_y)/2*w(i,j+1,n);
            end
            D_x = [D_x D_ij];
        end
        % Solve for vorticity at ith row using Matlab '/' matrix operator
        w_i_x = D_x/tri_x(2:end-1, 2:end-1);
        w_x(i, 2:end-1, n+1) = w_i_x;
    end
    % Y-SWEEP
    % 1st define tridiagonal matrices' elements
    % Perform Y-SWEEP
    for j= 2:N_x-1 % Iterate left to right
        D_y = [];
        for i = 2:N_y-1 % Iterate bottom to top
            % initialize D matrix of "Ax = D"
            if i == 2
                D_ij = (c_x(i,j)/2+d_x)/2 * w_x(i-1,j,n) + (1-d_x)*w_x(i,j,n) + (-c_x
(i,j)/2+d_x)/2*w_x(i+1,j,n) - A_y(i,j)*w_x(i-1,j,n+1);
```

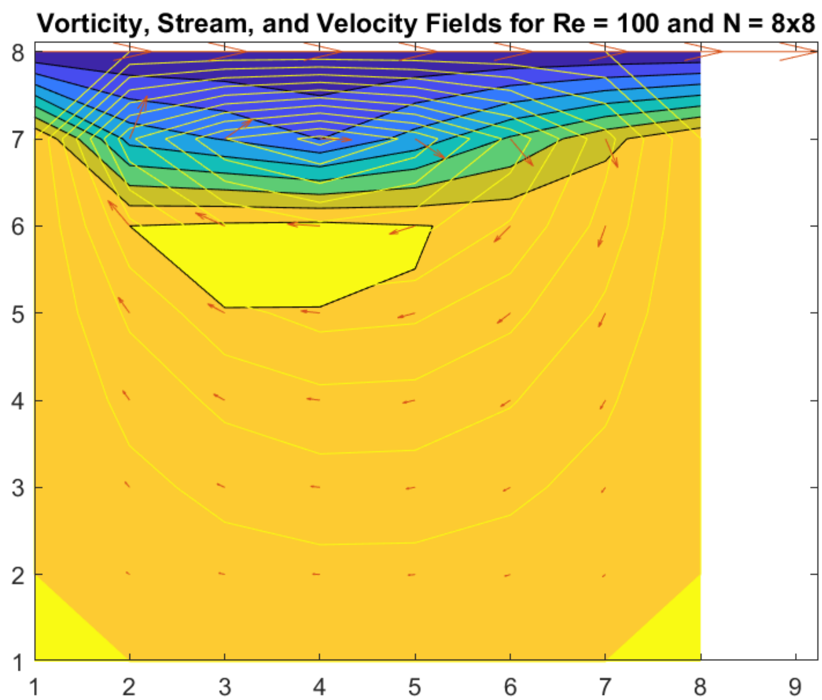
```
            elseif i == N_y-1
                D_ij = (c_x(i,j)/2+d_x)/2 * w_x(i-1,j,n) + (1-d_x)*w_x(i,j,n) + (-c_x
(i,j)/2+d_x)/2*w_x(i+1,j,n) - C_y(i,j)*w_x(i+1,j,n+1);
            else
                D_ij = (c_x(i,j)/2+d_x)/2 * w_x(i-1,j,n) + (1-d_x)*w_x(i,j,n) + (-c_x
(i,j)/2+d_x)/2*w_x(i+1,j,n);
            end
            D_y = [D_y D_ij];
        end
        % Solve for vorticity at jth column using Matlab '/' matrix operator
        w_j_y = D_y/tri_y(2:end-1, 2:end-1);
        w_y(2:end-1, j, n+1) = w_j_y;
        w = w_y;
    end
    %% Solve Stream Function by PSOR
    PSOR_error = 1; % initialize current PSOR error
    count = n;
    while PSOR_error > ERROR_TOL % iterate until PSOR error is within the tolerance
        psi_old = psi(:,:,end-1);
        for i = 2:N_y-1
            for j = 2:N_x-1
                psi(i,j,n+1) = (1-w_PSOR)*psi(i,j,n) + w_PSOR*alpha*(w(i,j,n+1)*dx^2+
psi(i+1,j,n) + psi(i-1,j,n+1)+beta^2*(psi(i,j+1,n)+psi(i,j-1,n+1)));
            end
        end
        psi_old = psi(:,:,end-1);
        PSOR_error = max(max(abs(psi(:,:,end) - psi_old)));
    end
    n = count;
    %% Initialize new velocity fields
    for i = 2:N_y-1
        for j = 2:N_x-1
            u(i,j,n+1) = (psi(i,j+1,n)-psi(i,j-1,n))/(2*dy);
            v(i,j,n+1) = -(psi(i+1,j,n)-psi(i-1,j,n))/(2*dx);
        end
    end
    %% Update Counters and Error to Check for Convergence
    current_error = max(max(abs(w_y(2:end-1,2:end-1,n+1) - w_y(2:end-1,2:end-1,n))));
    n = n + 1; % increment iteration level
    t = t + dt; % increment time
end
w_flipped = flipud(w(:,:,end));
psi_flipped = flipud(psi(:,:,end));
u_flipped = flipud(u(:,:,end));
v_flipped = flipud(v(:,:,end));


%% Plots
Re_str = num2str(Re(r));
```

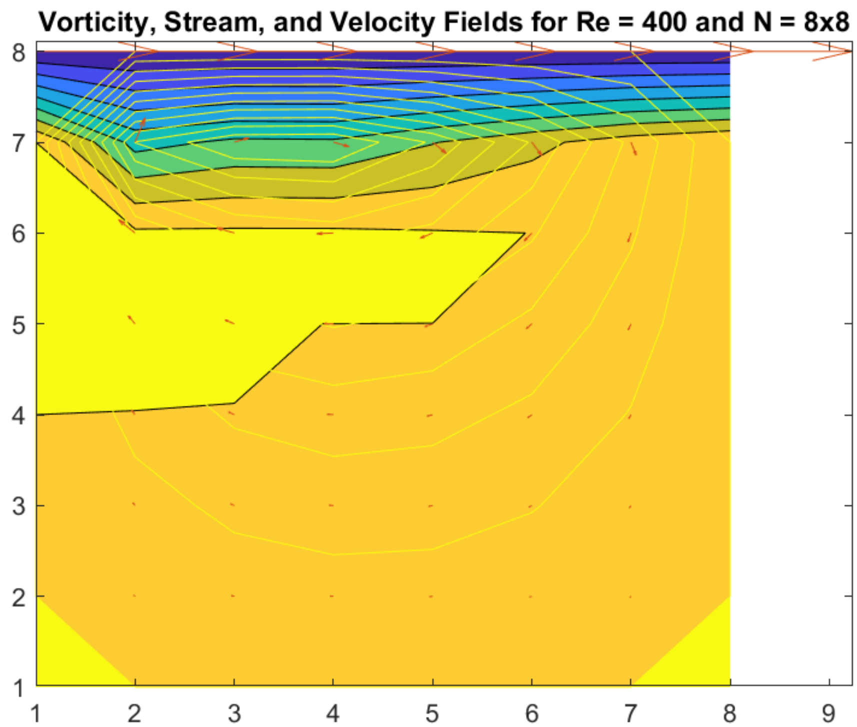
```
N_str = num2str(N(l));
figure(3*(l-1)+r) % Vorticity Plot
contourf(w_flipped)
hold on
%title(['Vorticity for Re = ', Re_str, ' and N = ', N_str, 'x', N_str]);
%figure(3*(l-1)+r+1) % Stream-function Plot
contour(psi_flipped)
hold on
%title(['Stream Function for Re = ', Re_str, ' and N = ', N_str, 'x', N_str]);
%figure(3*(l-1)+r+2) % Velocity-field Plot
quiver(u_flipped(2:end-1,2:end-1), v_flipped(2:end-1,2:end-1), 1)
title(['Vorticity, Stream, and Velocity Fields for Re = ', Re_str, ' and N = ',↙
N_str, 'x', N_str]);
end
end
```

Program Results

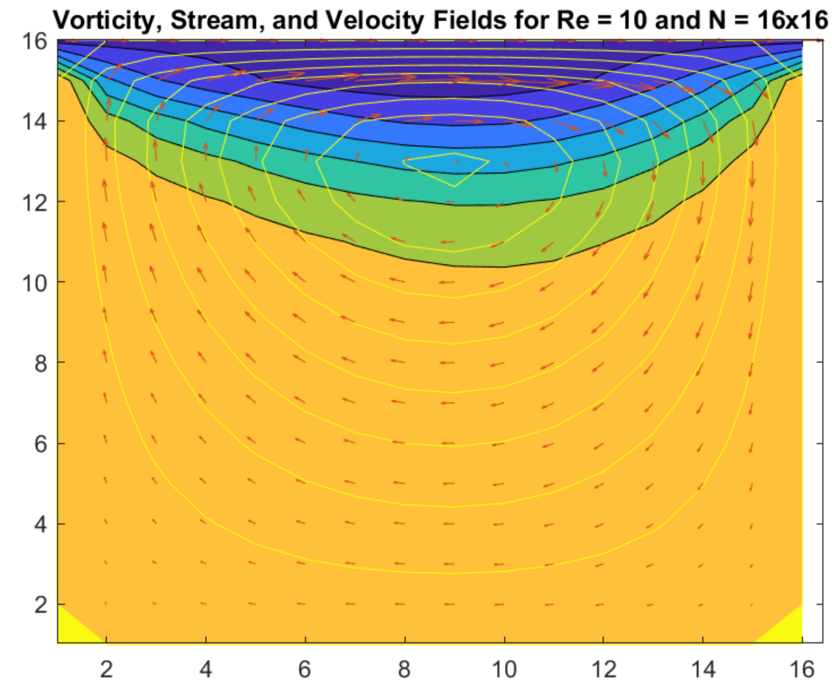
8 x 8 Grid Results

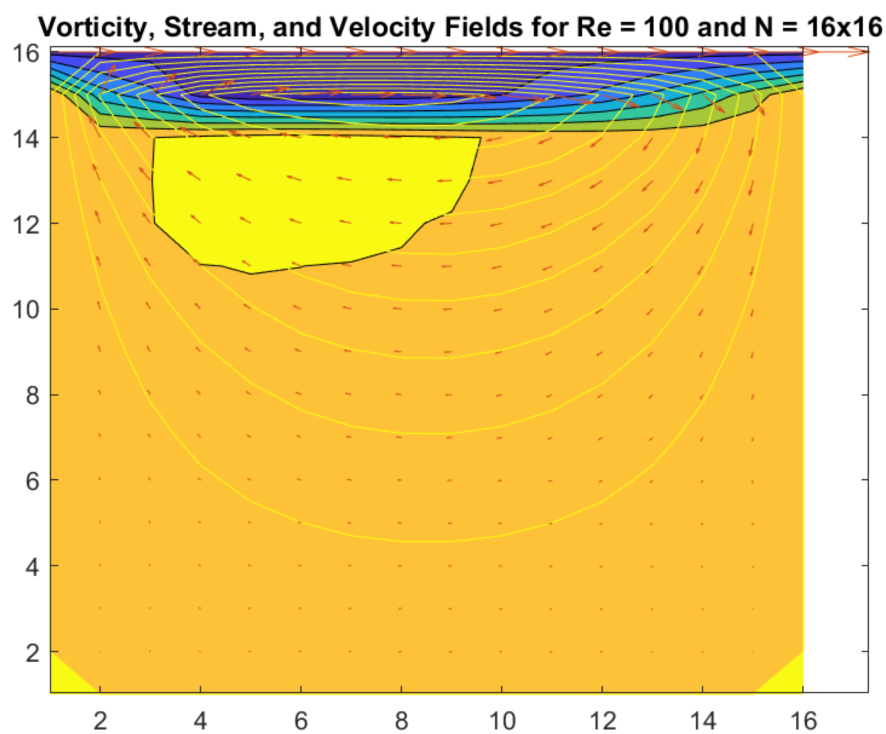


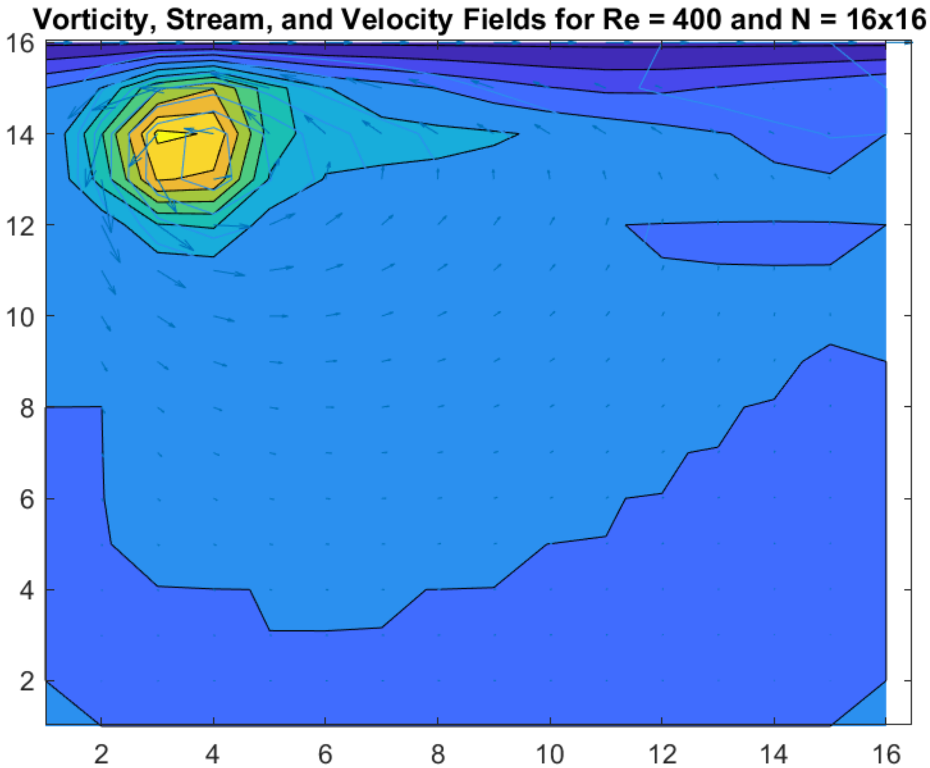



Vorticity, Stream, and Velocity Fields for Re = 400 and N = 8x8
1
2
3
4
5
6
7
8
9

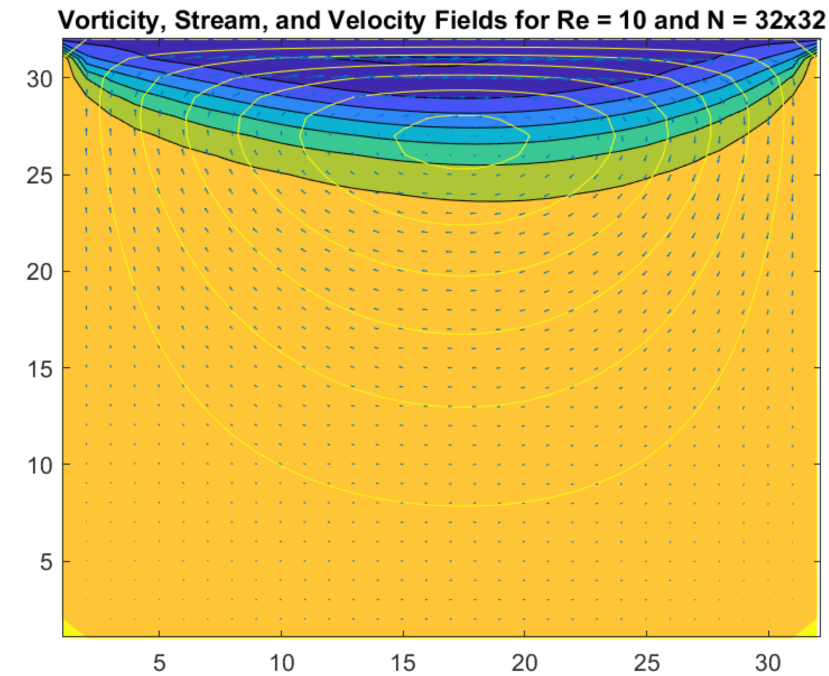
16 x 16 Grid Results

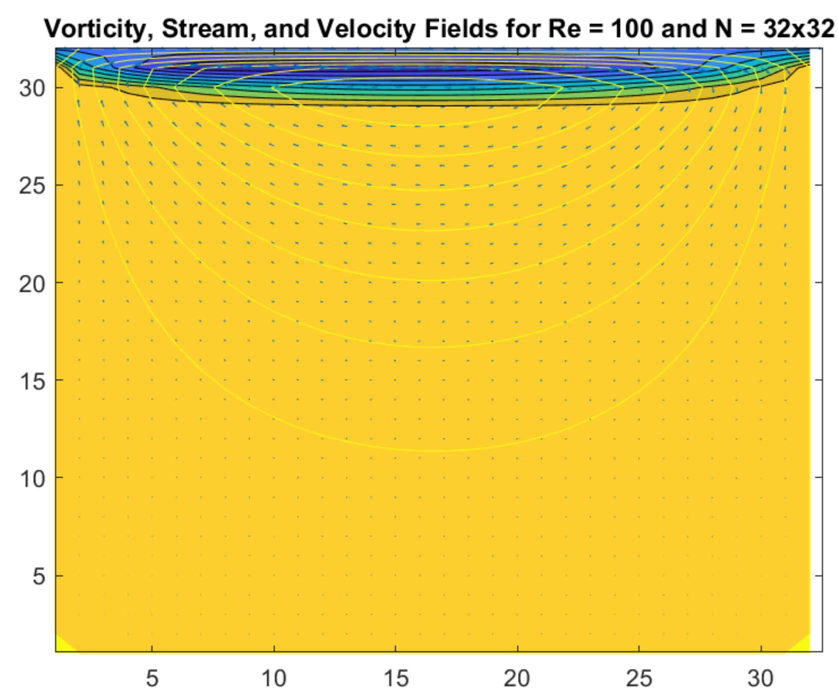





Vorticity, Stream, and Velocity Fields for Re = 400 and N = 16x16
16
14
12
10
8
6
4
2
2
4
6
8
10
12
14
16

32 x 32 Grid Results





Discussion

In each figure, there are separation points (yellow portions of the plot) at the bottom corners of the cavity. For Reynolds numbers of 100 there is additional separation occurring off the left wall and below the location of maximum vorticity, and for Reynolds numbers of 400 there seems to be separation at the locations of high vorticity.

The nature of the results change depending on the number of nodes being solved, which is a sign that the computations were improperly implemented into MATLAB. Rather than seeing entirely different results altogether for higher node-counts, we should expect to see the same results shown in higher resolution.