Final Project Report

Se Hwan Jeon

December 15, 2019

1 Theory

Complex partial differential equations (PDEs) are often responsible for determining the behavior of natural systems, from heat transfer to gravitation. Generally speaking, few analytical solutions exist for these PDEs even with a number of simplifying assumptions. For these complex systems, a variety of methods are available to numerically approximate such as finite volume and finite element methods. In this report, the steady-state behaviour of 2D square lid-driven cavity flow will be investigated for fluid as shown in Figure 1. The dimensions of the domain are detailed in the subsequent sections.

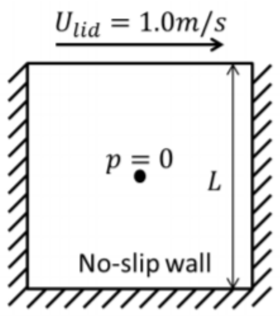

Figure 1: Geometry of lid-driven cavity flow.

1.1 Governing Equations

The governing equations for fluid flow are the Navier-Stokes equations. Constant density, viscosity, and temperature assumptions were made to simplify the analysis, and consequently, the energy equation was decoupled from the the continuity and momentum equations so that it would not need to be solved. As a result, the resulting governing equations can be given as

$$\frac{1}{\beta^2}\frac{\partial p}{\partial t} + \rho\frac{\partial u}{\partial x} + \rho\frac{\partial v}{\partial y} = S, \tag{1.1}$$

$$\rho\frac{\partial u}{\partial t} + \rho u\frac{\partial u}{\partial x} + \rho v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} = \mu\frac{\partial^2 u}{\partial x^2} + \mu\frac{\partial^2 u}{\partial y^2}, \tag{1.2}$$

$$\rho\frac{\partial v}{\partial t} + \rho u\frac{\partial v}{\partial x} + \rho v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} = \mu\frac{\partial^2 v}{\partial x^2} + \mu\frac{\partial^2 v}{\partial y^2}, \tag{1.3}$$

where p is pressure, u is velocity in the x-direction, v is velocity in the y-direction, ρ is the density of the fluid, and μ is the viscosity of the fluid.

β^2 is a time-derivative preconditioning term, and S is the artificial viscosity, given as

$$\beta^2 = \max(u^2+v^2, \kappa U_{lid}^2), \tag{1.4}$$

$$S = -\frac{|\lambda_x|_{max} C^{(4)} \Delta x^3}{\beta^2}\frac{\partial^4 p}{\partial x^4} - \frac{|\lambda_y|_{max} C^{(4)} \Delta y^3}{\beta^2}\frac{\partial^4 p}{\partial y^4}, \tag{1.5}$$

where $|\lambda_x|_{max}$ is the magnitude of the largest eigenvalue in the (x,t) space, $|\lambda_y|_{max}$ is the magnitude of the largest eigenvalue in the (y,t) space, κ is a constant between 0.001 and 0.09, and $C^{(4)}$ is a constant between $\frac{1}{128}$ and $\frac{1}{16}$. The significance of the artificial compressibility term $\frac{1}{\beta^2}\frac{\partial p}{\partial t}$ and the ariticial viscosity term S will be further discussed in Section 1.5 and 1.6.

1.2 Boundary Conditions

The area of interest was defined to have lengths of 0.05 m both horizontally and vertically, and no slip conditions were enforced on each wall. This implied that at the boundaries, $u = v = 0$. Along the top surface, the horizontal velocity u was assumed to be equal to the velocity of the moving lid, U_{lid}, while the vertical velocity was also set to be zero. The density of the fluid was defined to be 1.0 kg/m^3.
For the pressure boundary conditions, linear interpolation was used from the interior of the system to obtain values at the edges of the domain. This can be expressed as

$$p_{i,j}^n = 2p_{i\pm1,j\pm1}^n - p_{i\pm2,j\pm2}^n, \tag{1.6}$$

where n is the time index, i is the index in the x-direction, and j is the index in the y-direction. For each wall, the pressure terms normal to the wall at the point of interest were used to determine the value at the boundary.

1.3 Discretization

A simple explicit scheme was used to discretize the governing equations outlined in Section 1.1. A first order upwind scheme was used for the time derivataives, and second order central differences were used to discretize the spatial derivatives. Solving explicitly for the $n+1$ time step in u, v, and p, the resulting system of equations is given as

$$p_{i,j}^{n+1} = p_{i,j}^n - \beta_{i,j}^2 \Delta t\left[\rho\frac{u_{i+1,j}^n - u_{i-1,j}^n}{2\Delta x} + \rho\frac{v_{i,j+1}^n - v_{i,j-1}^n}{2\Delta y} - S_{i,j} - f_{mass}(x,y)\right] \tag{1.7}$$

$$\begin{aligned} u_{i,j}^{n+1} = u_{i,j}^n + \frac{\Delta t}{\rho}\Bigg[&\rho u_{i,j}^n\frac{u_{i+1,j}^n - u_{i-1,j}^n}{2\Delta x} + \rho v_{i,j}^n\frac{u_{i,j+1}^n - u_{i,j-1}^n}{2\Delta y} + \frac{p_{i+1,j}^n - p_{i-1,j}^n}{2\Delta x} \\ &-\mu\frac{u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n}{\Delta x^2} - \mu\frac{u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n}{\Delta y^2} - f_{xmtm}(x,y)\Bigg] \end{aligned} \tag{1.8}$$

$$\begin{aligned} v_{i,j}^{n+1} = v_{i,j}^n + \frac{\Delta t}{\rho}\Bigg[&\rho u_{i,j}^n\frac{v_{i+1,j}^n - v_{i-1,j}^n}{2\Delta x} + \rho v_{i,j}^n\frac{v_{i,j+1}^n - v_{i,j-1}^n}{2\Delta y} + \frac{p_{i,j+1}^n - p_{i,j-1}^n}{2\Delta y} \\ &-\mu\frac{v_{i+1,j}^n - 2v_{i,j}^n + v_{i-1,j}^n}{\Delta x^2} - \mu\frac{v_{i,j+1}^n - 2v_{i,j}^n + v_{i,j-1}^n}{\Delta y^2} - f_{ymtm}(x,y)\Bigg], \end{aligned} \tag{1.9}$$

where f_{mass}, f_{xmtm} and f_{ymtm} are source terms for the manufactured solution, as will be further discussed in Section 3. For the purposes of analyzing the cavity flow problem, these terms are equal to zero.

1.4 Stability

The convective stability limit for the time step is given as

$$\Delta t_c \leq \frac{\min(\Delta x, \Delta y)}{|\lambda|_{max}}, \tag{1.10}$$

where $|\lambda|_{max}$ is defined as

$$|\lambda|_{max} = \max(\lambda_x, \lambda_y), \tag{1.11}$$

$$\lambda_x = \frac{1}{2}(|u| + \sqrt{u^2 + 4\beta^2}), \tag{1.12}$$

$$\lambda_y = \frac{1}{2}(|v| + \sqrt{v^2 + 4\beta^2}). \tag{1.13}$$

Note that this time step is based not only on the mesh size, but also on the local velocity in space.
The diffusive stability limit for the time step is defined as

$$\Delta t_d \leq \frac{\Delta x \Delta y}{4\nu}, \tag{1.14}$$

where ν is the kinematic viscosity defined as $\nu = \mu/\rho$.
From these definitions, the global minimum time step can be defined as

$$\Delta t \leq \min(\Delta t_c, \Delta t_d) = CFL \min(\Delta t_c, \Delta t_d), \tag{1.15}$$

where the CFL number varies between zero and one. For the analysis of the lid-driven flow, the local time step was used to step forward Equations 1.7-1.9.

1.5 Artificial Viscosity

As shown in Equation 1.1 and 1.5, an artificial viscosity term is added to the Navier-Stokes equations. When using centered-difference schemes, oscillating solutions can satisfy finite difference equation, despite clear errors in the overall solution. This could be resolved by using upwind or downwind schemes, or by introducing this artificial viscosity term, which prevents odd-even decoupling from affecting the solution. This "JST" scheme (James, Schmidt, and Tucker, AIAA, 1991) serves to smooth out oscillations in the solution in areas where the pressure gradient of the flow is sharp. Because the order of the term is Δx^3, it does not affect the second-order accurate estimation of the solution. An example of this odd-even decoupling error is shown below in Figure 2.

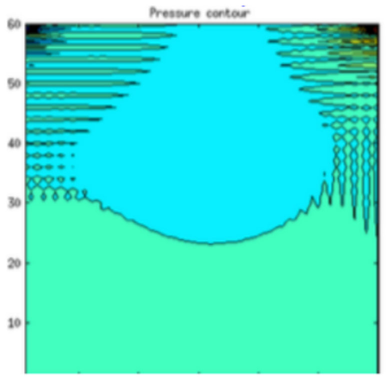

Figure 2: Odd-even decoupling in pressure field.

1.6 Artificial Compressibility

Under the assumptions outlined in Section 1.1, the incompressible Navier-Stokes equations are an ill-conditioned system difficult to resolve in a finite difference scheme. One method of overcoming this is to precondition the system of equations with a vanishing time derivative. This is done by introducing an artificial compressibility term, $\frac{1}{\beta^2}\frac{\partial P}{\partial t}$. Adding this term to the pressure equation allows the system to be fully defined. Because the term is artificial however, the solution will only be valid at steady-state when the pressure gradient in time is equal to zero.

1.7 Point Jacobi/SGS Algorithm

While matrices of unknowns can be directly solved by using methods such as the Thomas algorithm, it is generally far more efficient to use relaxation methods to resolve domains as the mesh size becomes finer. Instead of directly solving the system, the solutions can be iterated until they satisfy the defined finite difference equations (FDEs). For an iteration level k, the Point Jacobi method involves calculating the value of the solution at the $k+1^{th}$ level using values at the k^{th} level until the value satisfies the FDE to some specified tolerance.
The Symmetric Gauss-Seidel (SGS) method is similar, but instead uses the updated information at the $k+1^{th}$ levels to solve for other values at the k^{th} level as well. If swept forward across a domain, the method is more efficient at transferring information forward in space. To overcome this, the method is simply swept across the system again in the opposite direction to ensure consistent convergence to a solution. These methods were coded into the MATLAB environment to compute numerical solutions for u, v, and p.

2 Convergence Analysis

Using the methods outlined previously, a manufactured solution was used to investigate the convergence history for u, v, and p. The manufactured solution was tested at $Re = 100$, $CFL = 0.8$ for a variety of mesh sizes and its residuals were plotted as a function of iteration number, as shown in Figure 3.

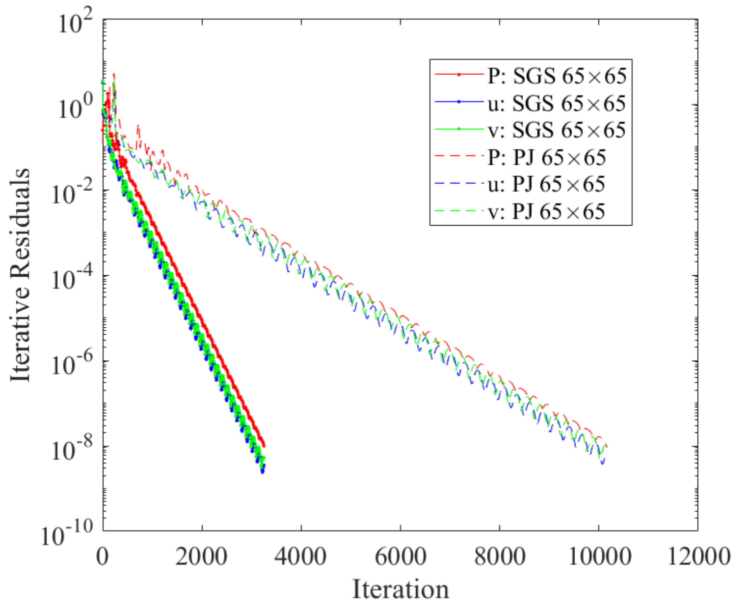

Figure 3: Plot of residual convergence history.

A tolerance of 10^-8 was set to determine convergence. As shown in the plot, the SGS method converges significantly faster than the point Jacobi method for the same mesh size. It was expected that as the mesh size was refined, the number of iterations needed to converge would increase, which Figure 3 confirms.

3 Code Verification and Discretization Error

To verify that the finite difference schemes were implemented correctly in code, the discretization error and order of accuracy was determined for the manufactured solutions provided. Both error estimations were done with $Re = 10, CFL = 0.5$. Because the defined scheme was second order accurate in space in both x and y directions, it was expected that the L_1, L_2, and L_∞ norms of the discretization error plotted as a function of mesh size h would exhibit a slope of 2 when plotted on a log scale. $h = 1$ was defined as the finest mesh with 129 nodes in both directions across the cavity, $h = 2$ was half of this space with 65 nodes across the cavity directions, and so on. The plot is shown in Figure 4.

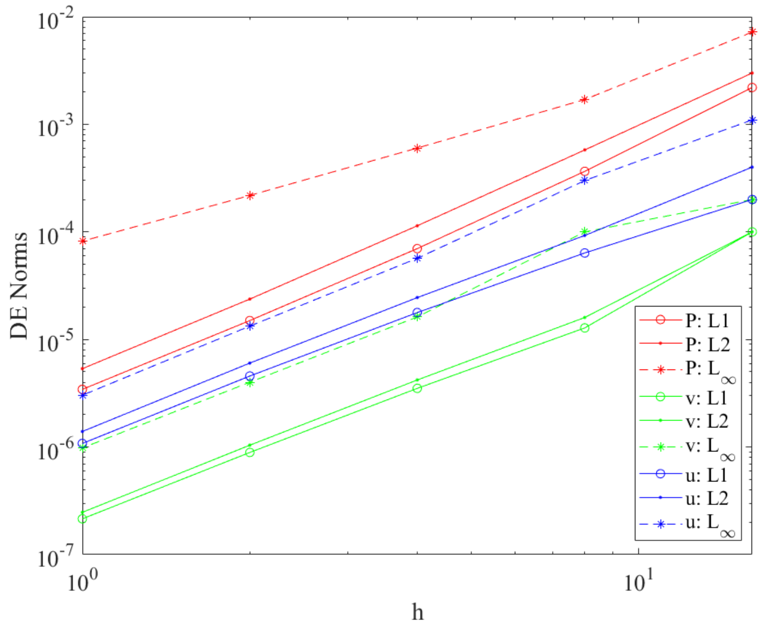

Figure 4: Plot of discretization error norms as a function of mesh size.

To estimate the order of accuracy, an expression for the observed order of accuracy can be given as

$$\hat{p} = \frac{\ln \frac{DE_1}{DE_2}}{\ln(r)}, \tag{1.16}$$

where DE_i is the discretization error at the i^{th} mesh level, $\hat{p}$ is the observed order of accuracy, and r is the grid refinement factor. In this case, $\hat{p}$ is expected to be near 2, as the FDE scheme chosen was second-order accurate in space. The grid refinement factor is also 2, because for each value of h, Δx and Δy would be half the size of the previous mesh. The plot of the observed order of accuracy as a function of mesh size is shown in Figure 5.

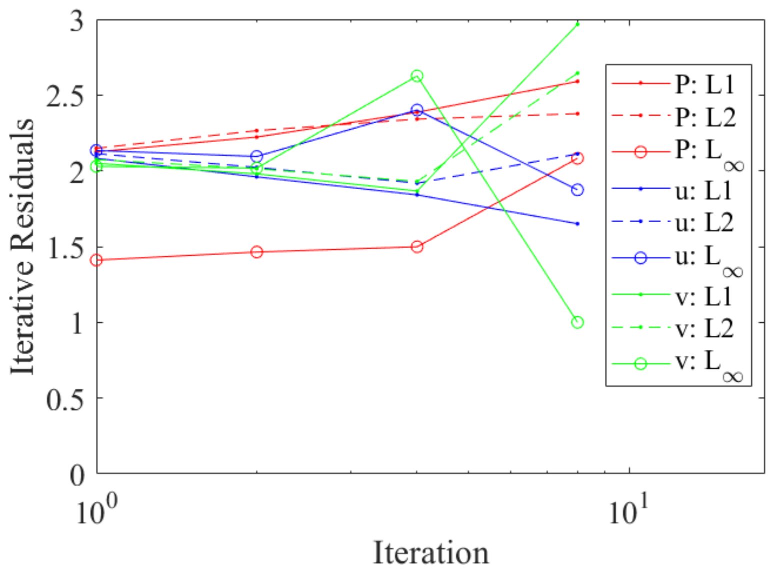

Figure 5: Plot of observed order of accuracy as a function of mesh size.

As can be seen from the plot, as the mesh is refined, the order of accuracy approaches the expected value of 2 for all three defined norms.

4 Flow Analysis

With the verified code, contour plots for u, v, and p were generated for the cavity at different Reynolds number configurations, as shown in Figures 6, 7, 8, and 9. The CFL number and mesh size were also adjusted as was appropriate so that the solution would converge.

4.1 $Re = 10$

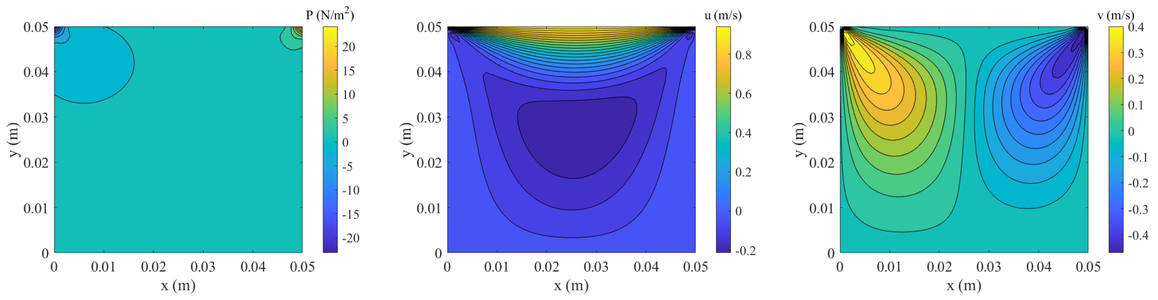

Figure 6: Plots of pressure, u, and v for $Re = 10$.

For low Reynolds number flow, the horizontal velocity is fairly symmetric across the cavity. Little variation is seen across the pressure field except for small differences at the corners.

4.2 $Re = 100$

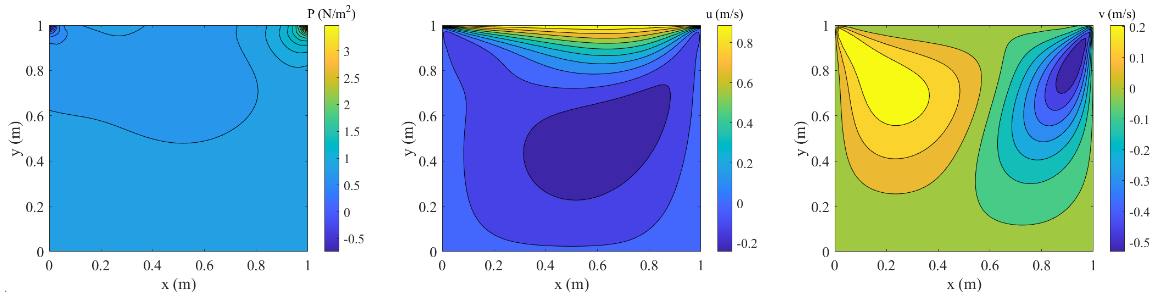

Figure 7: Plots of pressure, u, and v for $Re = 100$.

As the Reynold number is increased, the flow is more fully defined and asymmetric in the horizontal direction. The vertical velocity field also has higher values overall, more defined pressure regions have begun to form.

4.3 $Re = 500$

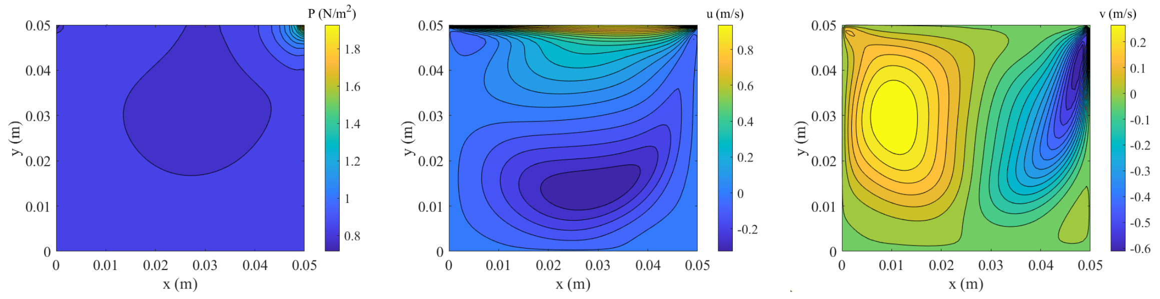

Figure 8: Plots of pressure, u, and v for $Re = 500$.

As before, the influence of the driven lid becomes more apparent and exaggerated as the Reynolds number is increased. Vortices can begin to be seen near the bottom corners of the cavity.

4.4 $Re = 1000$

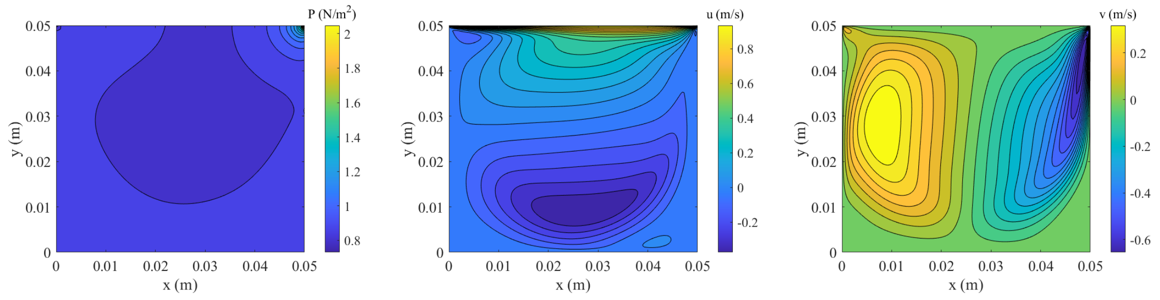

Figure 9: Plots of pressure, u, and v for $Re = 1000$.

When the Reynolds number is increased to 1000, these vortices grow larger and the flow more exaggerated. The vertical velocity of the flow is significantly affected by the moving lid, and more substantial pressure gradients begin to form.

5 Parameter Study

The effect of a number of parameters that control significant inputs to the discretization scheme are studied in this section.

5.1 Effect of Varying κ

As defined in Equation 1.4, κ influences the value of β^2, which in turn influences the artificial viscosity. The effect of varying κ was investigated on the convergence time of the mesh for the manufactured solution. At a constant 65×65 mesh, convergence history was plotted as a function of κ, as can be seen in Figure 10.

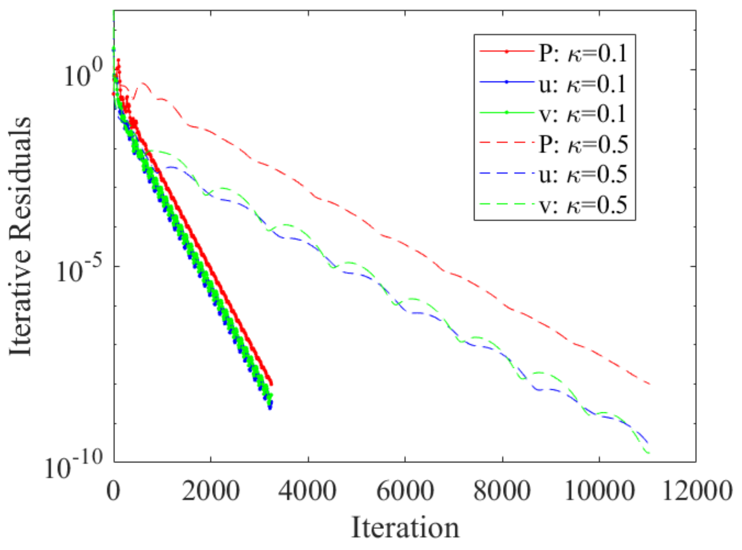

Figure 10: Plot of convergence history for varied κ.

As can be seen in the figure, varying κ seems to significantly affect the convergence of the FDE to the specified tolerance. Because the time derivative preconditioning term is affected by κ, the number of iterations to convergence would be affected by this parameter as well.

5.2 Effect of Varying $C^{(4)}$

$C^{(4)}$ controls the artificial viscosity term, and the effect of varying this parameter on the discretization error of the manufactured solution was investigated. A slice plot of the pressure field at $y = 0.025$ m for $Re = 10$ and $CFL = 0.8$ was taken to see the effect of the damping term in space, as shown in Figure 11.

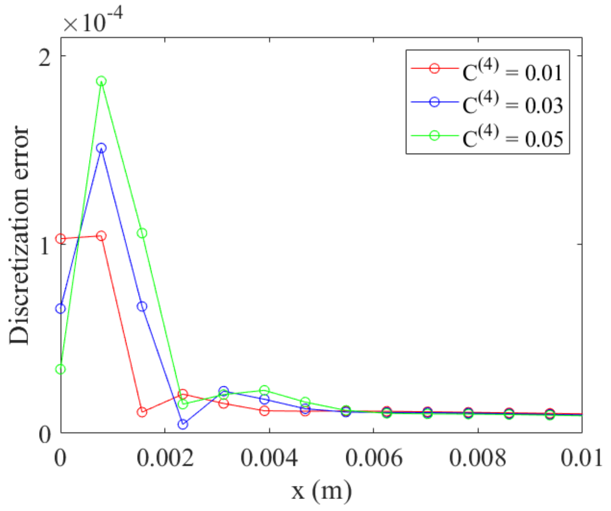

Figure 11: Plot of discretization error of pressure in space as $C^{(4)}$ constant is varied for manufactured solution.

As the constant is increased, the damping effect of the artificial viscosity term is increased as well. While the initial discretization is large from inconsistencies with the boundary, the largest $C^{(4)}$ term is damped most quickly to converge to the solution. If this constant was equal to zero, oscillating discretization error would be seen across the domain due to odd-even decoupling.

5.3 Effect of Varying Mesh Size

Lastly, the effect of the mesh refinement was investigated. For the lid-driven cavity flow at $Re = 10$, the contour u was plotted for a number of mesh sizes, as shown in Figure 12.

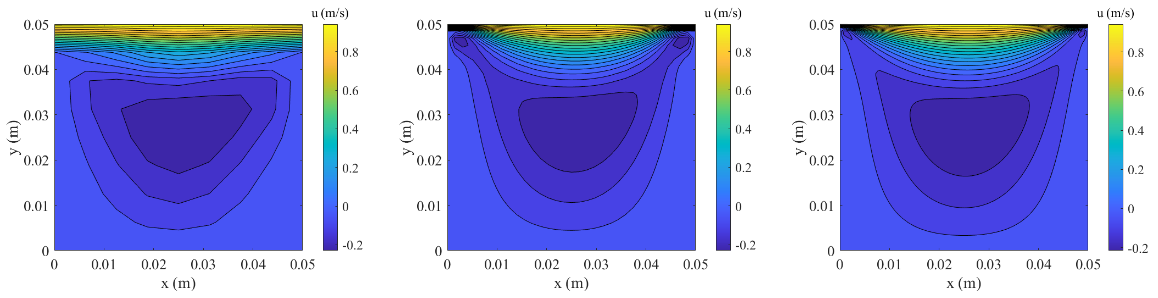

Figure 12: Contour plots of u for $Re = 10$ with mesh sizes 9×9 (left), 33×33 (middle), and 65×65 (right).

As mesh size is refined, more interstices and finer edges are resolved to develop a more accurate solution for the cavity flow. Depending on the level of fidelity needed for the analysis, the mesh could be adjusted considering factors such as computational resources, time, and necessary level of detail. For a relatively simple system like the one presented, even coarse meshes generally capture the overall behaviour of the system, but for more complex geometries or flow conditions, finer meshes may be required.

Appendix

The coded subroutines from the given `cavity_solver.m` file template are attached below.

```
function bndry(~)
%
%Uses global variable(s): zero, one, two, half, imax, jmax, uinf
%To modify: u

% i                        % i index (x direction)
% j                        % j index (y direction)

global zero two half imax jmax uinf
global u

% This applies the cavity boundary conditions

% left and right walls
for i=1:imax
    j=1; % left wall
    u(i,j,1) = 2*u(i,j+1,1)-u(i,j+2,1);
    u(i,j,2) = 0;
    u(i,j,3) = 0;

    j=jmax;
    u(i,j,1) = 2*u(i,jmax-1,1)-u(i,jmax-2,1);
    u(i,j,2) = 0;
    u(i,j,3) = 0;
end

% top and bottom walls
for j=1:jmax
    i=imax;    % top wall
    u(i,j,1) = 2*u(i-1,j,1)-u(i-2,j,1);
    u(i,j,2) = uinf;
    u(i,j,3) = 0;

    i=1; % bottom wall
    u(i,j,1) = 2*u(i+1,j,1)-u(i+2,j,1);
    u(i,j,2) = 0;
    u(i,j,3) = 0;
end
end

function [dtmin] = compute_time_step(dtmin)
%
%Uses global variable(s): one, two, four, half, fourth
%Uses global variable(s): vel2ref, rmu, rho, dx, dy, cfl, rkappa, imax, jmax
%Uses: u
%To Modify: dt, dtmin

% i                        % i index (x direction)
% j                        % j index (y direction)

% dtvisc       % Viscous time step stability criteria (constant over domain)
% uvel2        % Local velocity squared
% beta2        % Beta squared paramete for time derivative preconditioning
% lambda_x     % Max absolute value eigenvalue in (x,t)
% lambda_y     % Max absolute value eigenvalue in (y,t)
% lambda_max   % Max absolute value eigenvalue (used in convective time step computation)
% dtconv       % Local convective time step restriction

global four half fourth
global vel2ref rmu rho dx dy cfl rkappa imax jmax
global u dt

delta_t_max = 1.0e99;
dtvisc = (dx*dy*fourth)/(rmu/rho);
for i=2:imax-1
    for j=2:jmax-1
        uvel2 = u(i,j,2)^2+u(i,j,3)^2;
        beta2 = max(uvel2,rkappa*vel2ref);
        lambda_x = half*(abs(u(i,j,2))+sqrt(u(i,j,2)^2+four*beta2));
        lambda_y = half*(abs(u(i,j,3))+sqrt(u(i,j,3)^2+four*beta2));
        lambda_max = max(lambda_x,lambda_y);
        dtconv = min(dx,dy)/(lambda_max);
        dt(i,j) = cfl*min(dtconv,dtvisc);
        delta_t_max = min(dt(i,j),delta_t_max);
    end
end
dtmin = delta_t_max;
%dtmin = cfl*min(delta_t_max,dtvisc);


end

function Compute_Artificial_Viscosity(~)
%
%Uses global variable(s): zero, one, two, four, six, half, fourth
%Uses global variable(s): imax, jmax, lim, rho, dx, dy, Cx, Cy, Cx2, Cy2, fsmall, vel2ref, rkappa
%Uses: u
%To Modify: artviscx, artviscy

% i                        % i index (x direction)
% j                        % j index (y direction)

% uvel2        % Local velocity squared
% beta2        % Beta squared parameter for time derivative preconditioning
% lambda_x     % Max absolute value e-value in (x,t)
% lambda_y     % Max absolute value e-value in (y,t)
% d4pdx4       % 4th derivative of pressure w.r.t. x
% d4pdy4       % 4th derivative of pressure w.r.t. y
```

```
global two four six half
global imax jmax lim rho dx dy Cx Cy Cx2 Cy2 fsmall vel2ref rkappa
global u
global artviscx artviscy


% artviscx loop
for i=1:imax
    for j=3:jmax-2
        uvel2 = u(i,j,2)^two + u(i,j,3)^two;
        beta2 = max(uvel2,rkappa*vel2ref);
        lambda_x = half*(abs(u(i,j,2))+sqrt(u(i,j,2)^2+four*beta2));
        d4pdx4 = (u(i,j+2,1)-four*u(i,j+1,1)+six*u(i,j,1)-four*u(i,j-1,1)+u(i,j-2,1))/dx;
        artviscx(i,j) = -abs(lambda_x)*Cx*d4pdx4/(beta2);
    end
    artviscx(i,2) = artviscx(i,3);
    artviscx(i,jmax-1) = artviscx(i,jmax-2);
end


% artviscy loop
for j=1:jmax
    for i=3:imax-2
        uvel2 = u(i,j,2)^two + u(i,j,3)^two;
        beta2 = max(uvel2,rkappa*vel2ref);
        lambda_y = half*(abs(u(i,j,3))+sqrt(u(i,j,3)^2+four*beta2));
        d4pdy4 = (u(i+2,j,1)-four*u(i+1,j,1)+six*u(i,j,1)-four*u(i-1,j,1)+u(i-2,j,1))/dy;
        artviscy(i,j) = -abs(lambda_y)*Cy*d4pdy4/(beta2);
    end
    artviscy(2,j) = artviscy(3,j);
    artviscy(imax-1,j) = artviscy(imax-2,j);
end

end


function SGS_forward_sweep(~)
%
%Uses global variable(s): two, three, six, half
%Uses global variable(s): imax, imax, jmax, ipgorder, rho, rhoinv, dx, dy, rkappa, ...
%                       xmax, xmin, ymax, ymin, rmu, vel2ref
%Uses: artviscx, artviscy, dt, s
%To Modify: u

% i                         % i index (x direction)
% j                         % j index (y direction)

% dpdx         % First derivative of pressure w.r.t. x
% dudx         % First derivative of x velocity w.r.t. x
% dvdx         % First derivative of y velocity w.r.t. x
% dpdy          % First derivative of pressure w.r.t. y
% dudy          % First derivative of x velocity w.r.t. y
% dvdy          % First derivative of y velocity w.r.t. y
% d2udx2        % Second derivative of x velocity w.r.t. x
% d2vdx2        % Second derivative of y velocity w.r.t. x
% d2udy2        % Second derivative of x velocity w.r.t. y
% d2vdy2        % Second derivative of y velocity w.r.t. y
% beta2         % Beta squared parameter for time derivative preconditioning
% uvel2         % Velocity squared

global two half
global imax jmax rho rhoinv dx dy rkappa rmu vel2ref
global artviscx artviscy dt s u uold

% Symmetric Gauss-Siedel: Forward Sweep

for j=2:jmax-1
    for i=2:imax-1
        uvel2 = norm([u(i,j,2),u(i,j,3)])^two;
        beta2 = max(uvel2,rkappa*vel2ref);

        dudx = half*(uold(i,j+1,2)-u(i,j-1,2))/dx;
        dvdx = (uold(i,j+1,3)-u(i,j-1,3))/(2*dx);
        dpdx = (uold(i,j+1,1)-u(i,j-1,1))/(2*dx);

        dudy = (uold(i+1,j,2)-u(i-1,j,2))/(2*dy);
        dvdy = (uold(i+1,j,3)-u(i-1,j,3))/(2*dy);
        dpdy = (uold(i+1,j,1)-u(i-1,j,1))/(2*dy);

        d2udx2 = (uold(i,j+1,2)-2*uold(i,j,2)+u(i,j-1,2))/dx^2;
        d2vdx2 = (uold(i,j+1,3)-2*uold(i,j,3)+u(i,j-1,3))/dx^2;
        d2udy2 = (uold(i+1,j,2)-2*uold(i,j,2)+u(i-1,j,2))/dy^2;
        d2vdy2 = (uold(i+1,j,3)-2*uold(i,j,3)+u(i-1,j,3))/dy^2;


        u(i,j,1) = uold(i,j,1)-beta2*dt(i,j)*(rho*dudx+rho*dvdy-(artviscx(i,j)
        +artviscy(i,j))-s(j,i,1));
        u(i,j,2) = uold(i,j,2)-dt(i,j)*rhoinv*(rho*uold(i,j,2)*dudx+rho*uold(i,j,3)*dudy
        +dpdx-rmu*d2udx2-rmu*d2udy2-s(j,i,2));
        u(i,j,3) = uold(i,j,3)-dt(i,j)*rhoinv*(rho*uold(i,j,2)*dvdx+rho*uold(i,j,3)*dvdy
        +dpdy-rmu*d2vdx2-rmu*d2vdy2-s(j,i,3));

    end
end


end


function SGS_backward_sweep(~)
%
```

```
%Uses global variable(s): two, three, six, half
%Uses global variable(s): imax, imax, jmax, ipgorder, rho, rhoinv, dx, dy, rkappa, ...
%                        xmax, xmin, ymax, ymin, rmu, vel2ref
%Uses: artviscx, artviscy, dt, s
%To Modify: u

% i                         % i index (x direction)
% j                         % j index (y direction)

% dpdx         % First derivative of pressure w.r.t. x
% dudx         % First derivative of x velocity w.r.t. x
% dvdx         % First derivative of y velocity w.r.t. x
% dpdy         % First derivative of pressure w.r.t. y
% dudy         % First derivative of x velocity w.r.t. y
% dvdy         % First derivative of y velocity w.r.t. y
% d2udx2       % Second derivative of x velocity w.r.t. x
% d2vdx2       % Second derivative of y velocity w.r.t. x
% d2udy2       % Second derivative of x velocity w.r.t. y
% d2vdy2       % Second derivative of y velocity w.r.t. y
% beta2        % Beta squared parameter for time derivative preconditioning
% uvel2        % Velocity squared

global two half
global imax jmax rho rhoinv dx dy rkappa rmu vel2ref
global artviscx artviscy dt s u uold

% Symmetric Gauss-Siedel: Backward Sweep
for j=jmax-1:-1:2
    for i=imax-1:-1:2
        uvel2 = norm([u(i,j,2),u(i,j,3)])^two;
        beta2 = max(uvel2,rkappa*vel2ref);

        dudx = half*(u(i,j+1,2)-u(i,j-1,2))/dx;
        dvdx = (u(i,j+1,3)-u(i,j-1,3))/(2*dx);
        dpdx = (u(i,j+1,1)-u(i,j-1,1))/(2*dx);

        dudy = (u(i+1,j,2)-u(i-1,j,2))/(2*dy);
        dvdy = (u(i+1,j,3)-u(i-1,j,3))/(2*dy);
        dpdy = (u(i+1,j,1)-u(i-1,j,1))/(2*dy);

        d2udx2 = (u(i,j+1,2)-2*u(i,j,2)+u(i,j-1,2))/dx^2;
        d2vdx2 = (u(i,j+1,3)-2*u(i,j,3)+u(i,j-1,3))/dx^2;
        d2udy2 = (u(i+1,j,2)-2*u(i,j,2)+u(i-1,j,2))/dy^2;
        d2vdy2 = (u(i+1,j,3)-2*u(i,j,3)+u(i-1,j,3))/dy^2;


        u(i,j,1) = u(i,j,1)-beta2*dt(i,j)*(rho*dudx+rho*dvdy-(artviscx(i,j)
        +artviscy(i,j))-s(j,i,1));
        u(i,j,2) = u(i,j,2)-dt(i,j)*rhoinv*(rho*u(i,j,2)*dudx+rho*u(i,j,3)*dudy
        +dpdx-rmu*d2udx2-rmu*d2udy2-s(j,i,2));
        u(i,j,3) = u(i,j,3)-dt(i,j)*rhoinv*(rho*u(i,j,2)*dvdx+rho*u(i,j,3)*dvdy
        +dpdy-rmu*d2vdx2-rmu*d2vdy2-s(j,i,3));

    end
end


end


function point_Jacobi(~)
%
%Uses global variable(s): two, three, six, half
%Uses global variable(s): imax, imax, jmax, ipgorder, rho, rhoinv, dx, dy, rkappa, ...
%                       xmax, xmin, ymax, ymin, rmu, vel2ref
%Uses: uold, artviscx, artviscy, dt, s
%To Modify: u


% i                         % i index (x direction)
% j                         % j index (y direction)

% dpdx          % First derivative of pressure w.r.t. x
% dudx          % First derivative of x velocity w.r.t. x
% dvdx          % First derivative of y velocity w.r.t. x
% dpdy          % First derivative of pressure w.r.t. y
% dudy          % First derivative of x velocity w.r.t. y
% dvdy          % First derivative of y velocity w.r.t. y
% d2udx2        % Second derivative of x velocity w.r.t. x
% d2vdx2        % Second derivative of y velocity w.r.t. x
% d2udy2        % Second derivative of x velocity w.r.t. y
% d2vdy2        % Second derivative of y velocity w.r.t. y
% beta2         % Beta squared parameter for time derivative preconditioning
% uvel2         % Velocity squared
global two half
global imax jmax rho rhoinv dx dy rkappa rmu vel2ref
global u uold artviscx artviscy dt s

% Point Jacobi method
for j=2:jmax-1
    for i=2:imax-1
        uvel2 = norm([u(i,j,2),u(i,j,3)])^two;
        beta2 = max(uvel2,rkappa*vel2ref);

        dudx = half*(uold(i,j+1,2)-uold(i,j-1,2))/dx;
        dvdx = (uold(i,j+1,3)-uold(i,j-1,3))/(2*dx);
        dpdx = (uold(i,j+1,1)-uold(i,j-1,1))/(2*dx);

        dudy = (uold(i+1,j,2)-uold(i-1,j,2))/(2*dy);
        dvdy = (uold(i+1,j,3)-uold(i-1,j,3))/(2*dy);
        dpdy = (uold(i+1,j,1)-uold(i-1,j,1))/(2*dy);
        d2udx2 = (uold(i,j+1,2)-2*uold(i,j,2)+uold(i,j-1,2))/dx^2;
        d2vdx2 = (uold(i,j+1,3)-2*uold(i,j,3)+uold(i,j-1,3))/dx^2;
        d2udy2 = (uold(i+1,j,2)-2*uold(i,j,2)+uold(i-1,j,2))/dy^2;
        d2vdy2 = (uold(i+1,j,3)-2*uold(i,j,3)+uold(i-1,j,3))/dy^2;


        u(i,j,1) = uold(i,j,1)-beta2*dt(i,j)*(rho*dudx+rho*dvdy-(artviscx(i,j)
        +artviscy(i,j))-s(j,i,1) );
        u(i,j,2) = uold(i,j,2)-dt(i,j)*rhoinv*(rho*uold(i,j,2)*dudx+rho*uold(i,j,3)*dudy
        +dpdx-rmu*d2udx2-rmu*d2udy2-s(j,i,2));
        u(i,j,3) = uold(i,j,3)-dt(i,j)*rhoinv*(rho*uold(i,j,2)*dvdx+rho*uold(i,j,3)*dvdy
        +dpdy-rmu*d2vdx2-rmu*d2vdy2-s(j,i,3));

    end
end

end


function [res, resinit, conv] = check_iterative_convergence...
    (n, res, resinit, ninit, rtime, dtmin)
%
%Uses global variable(s): zero
%Uses global variable(s): imax, jmax, neq, fsmall
%Uses: n, u, uold, dt, res, resinit, ninit, rtime, dtmin
%To modify: conv

% i                        % i index (x direction)
% j                        % j index (y direction)
% k                        % k index (# of equations)

global zero
global imax jmax neq fsmall
global u uold dt fp1

% Compute iterative residuals to monitor iterative convergence
N = imax*jmax;
resM = zeros(imax,jmax,neq);
for i=1:imax
    for j=1:jmax
        for k=1:neq
            resM(i,j,k) = abs((u(i,j,k)-uold(i,j,k))/dt(i,j));
        end
    end
end

for k=1:neq
    % res(k)=sum(resM(:,:,k))/N;        % L1
    res(k) = norm(resM(:,:,k))/(N);       % L2
    % res(k) = max(resM(:,:,k));        % Linf
end
```

```
if ((n>1)&&(n<5))
resinit(1) = max(resinit(1),res(1));
resinit(2) = max(resinit(2),res(2));
resinit(3) = max(resinit(3),res(3));
end
res = res./resinit;



% Write iterative residuals every 10 iterations
if ( (mod(n,10)==0)||(n==ninit) )
    fprintf(fp1, '%d   %e   %e   %e   %e\n',  n, rtime, res(1), res(2), res(3));
end

% Write header for iterative residuals every 200 iterations
if ( (mod(n,200)==0)||(n==ninit) )
    fprintf('Iter.    Time (s)       dt (s)      Continuity      x-Momentum       y-Momentum\n');
    fprintf('%d   %e   %e   %e   %e   %e\n',  n, rtime, dtmin, res(1), res(2), res(3));
end


if(min(res)==0)
    conv=1.0;
else
    conv = max(res);
end




end


function Discretization_Error_Norms(rL1norm, rL2norm, rLinfnorm)
%
%Uses global variable(s): zero
%Uses global variable(s): imax, jmax, neq, imms, xmax, xmin, ymax, ymin, rlength
%Uses: u
%To modify: rL1norm, rL2norm, rLinfnorm


% i                        % i index (x direction)
% j                        % j index (y direction)
% k                        % k index (# of equations)

% x        % Temporary variable for x location
% y        % Temporary variable for y location
% DE    % Discretization error (absolute value)

global zero imax jmax neq imms xmax xmin ymax ymin u ummsArray
```

```
if imms==1
    rL1norm(:) = 0;
    rL2norm(:) = 0;
    rLinfnorm(:) = 0;


N = imax*jmax;
DEmat = zeros(imax,jmax,neq);
    for i=1:imax
        for j=1:jmax
            for k=1:neq
                DEmat(i,j,k) = abs(u(i,j,k)-ummsArray(i,j,k));
            end
        end
    end
    for k=1:neq
        for i=1:imax
            for j=1:jmax
                rL1norm(k) = rL1norm(k)+DEmat(i,j,k)/N;
                rLinfnorm(k) = max(rLinfnorm(k),DEmat(i,j,k));
            end
        end
        rL2norm(k) = norm(DEmat(:,:,k))/sqrt(N);
    end
end
rL1norm
rL2norm
rLinfnorm
end
```