Lid-Driven Cavity Flow using a Staggered Finite-Volume Pressure Correction Method

Ryan Coe

Department of Aerospace and Ocean Engineering, Virginia Tech, Blacksburg, VA, USA

ABSTRACT

A finite-volume method has been developed to solve the incompressible Navier-Stokes equations with primitive variables in a non-dimensional form. The classical lid-driven cavity problem will be examined at a range of Reynolds numbers and solution method parameters. The pressure-correction method is used in conjunction with a 2nd-order Adams-Bashforth time advancement scheme on a uniform, Cartesian staggered grid.

1 INTRODUCTION

The incompressible viscous Navier-Stokes equations present a mixed elliptic-parabolic problem. Iterative pressure correction methods, originally developed for free surface incompressible flows [1], offer an efficient means of solving these equations.

The assumption of a constant density (ρ) and viscosity (μ) removes the energy equation from the system causing the speed of sound to be theoretically infinite. This in turn makes the explicit schemes often used in similar compressible problems extremely ineffective due to stability constraints (CFL and von Neumann) that severely limit the time-step size.

2 Methods

The incompressible Navier-Stokes equations can be written non-dimensionally as

$$\frac{\partial u}{\partial t}+\frac{\partial u^2}{\partial x}+\frac{\partial \mathrm{uv}}{\partial y}=\frac{-\partial p}{\partial x}+\frac{1}{\mathrm{Re}}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)$$

$$\frac{\partial v}{\partial t}+\frac{\partial v^2}{\partial y}+\frac{\partial \mathrm{uv}}{\partial x}=\frac{-\partial p}{\partial y}+\frac{1}{\mathrm{Re}}\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right)$$

$$\frac{\partial \mathrm{u}}{\partial x}+\frac{\partial \mathrm{v}}{\partial y}=0$$

with the non-dimensional variables formulated as follows.

$$u=\frac{u'}{U_{\mathrm{lid}}} \qquad v=\frac{v'}{U_{\mathrm{lid}}} \qquad t=\frac{t'U_{\mathrm{lid}}}{L_{\mathrm{lid}}}$$

$$x=\frac{x'}{L_{\mathrm{lid}}} \qquad y=\frac{y'}{L_{\mathrm{lid}}}$$

$$\rho=\frac{\rho'}{\rho_{\mathrm{ref}}} \qquad \mu=\frac{\mu'}{\mu_{\mathrm{ref}}}$$

$$p=\frac{p'-p_{\mathrm{ref}}}{\rho' U_{\mathrm{lid}}^2} \qquad \mathrm{Re}=\frac{\rho' U_{\mathrm{lid}} L_{\mathrm{lid}}}{\mu'}$$

To prevent decoupling of the pressure and velocity at adjacent grid points a staggered grid was used. Standard 2nd order central difference schemes allow for "checkerboard" solutions, in which the equations are satisfied by unphysical fluctuations in the fluid parameters. By staggering the location each variable (p, x-velocity and y-velocity) to different locations of the cell, the finite-volume discretizations can be based on adjacent points, thus eliminating issues of irrational oscillations.

The momentum equations can be represented by finite-volume central difference discretizations, where i and j are cell indices on the staggered grid in the x and y-directions respectively.

$$\frac{\partial u}{\partial t}=\frac{u_{i,j}^{n+1}-u_{i,j}^{n}}{\Delta \mathrm{t}}+\mathcal{O}(\Delta \mathrm{t})$$

$$\frac{\partial u^2}{\partial x}=\frac{(u^2)_e-(u^2)_w}{\Delta \mathrm{x}_s}+\mathcal{O}(\Delta \mathrm{x}_s^2)$$

$$(u)_e=\frac{1}{2}\left(u_{i,j}+u_{i+1,j}\right)$$

$$(u)_w=\frac{1}{2}\left(u_{i-1,j}+u_{i,j}\right)$$

$$\frac{\partial \mathrm{uv}}{\partial y}=\frac{(\mathrm{uv})_n-(\mathrm{uv})_s}{\Delta \mathrm{y}_v}+\mathcal{O}(\Delta \mathrm{y}_v^2)$$

$$(u)_n=\frac{1}{2}\left(u_{i,j}+u_{i,j+1}\right)$$

$$(u)_s=\frac{1}{2}\left(u_{i,j-1}+u_{i,j}\right)$$

$$(v)_n=\frac{1}{2}\left(v_{i-1,j+1}+v_{i,j+1}\right)$$

$$(v)_s=\frac{1}{2}\left(v_{i-1,j}+v_{i,j}\right)$$

$$\frac{\partial p}{\partial x} = \frac{p_{i,j} - p_{i-1,j}}{\Delta x_s} + \mathcal{O}(\Delta x_s^2)$$

$$\frac{\partial^2 u}{\partial x^2} = \frac{\left(\frac{\partial u}{\partial x}\right)_e - \left(\frac{\partial u}{\partial x}\right)_w}{\Delta x_s} + \mathcal{O}(\Delta x_s^2)$$

$$\left(\frac{\partial u}{\partial x}\right)_e = \frac{u_{i+1,j} - u_{i,j}}{\Delta x_u} + \mathcal{O}(\Delta x_u^2)$$

$$\left(\frac{\partial u}{\partial x}\right)_w = \frac{u_{i,j} - u_{i-1,j}}{\Delta x_u} + \mathcal{O}(\Delta x_u^2)$$

$$\frac{\partial^2 u}{\partial y^2} = \frac{\left(\frac{\partial u}{\partial y}\right)_n - \left(\frac{\partial u}{\partial y}\right)_s}{\Delta y_v} + \mathcal{O}(\Delta y_v^2)$$

$$\left(\frac{\partial u}{\partial y}\right)_n = \frac{u_{i,j+1} - u_{i,j}}{\Delta y_s} + \mathcal{O}(\Delta y_s^2)$$

$$(\frac{\partial u}{\partial y})_s = \frac{u_{i,j} - u_{i,j-1}}{\Delta y_s} + \mathcal{O}(\Delta y_s^2)$$

The approximations for the y-momentum equation can be formed in a similar manner. The pressure is found by forming a Poisson problem using the continuity equation.

$$\frac{\partial u}{\partial x} = \frac{u_{i+1,j} - u_{i,j}}{\Delta x_u} + \mathcal{O}(\Delta x_u^2)$$

$$u_{i+1,j}^{n+1} = \tilde{u}_{i+1,j} - \Delta t \frac{p_{i+1,j} - p_{i,j}}{\Delta x_s}$$

$$u_{i,j}^{n+1} = \tilde{u}_{i,j} - \Delta t \frac{p_{i,j} - p_{i-1,j}}{\Delta x_s}$$

$$\frac{\partial v}{\partial y} = \frac{v_{i,j+1} - \mathrm{uv}_{i,j}}{\Delta y_v} + \mathcal{O}(\Delta y_v^2)$$

$$v_{i,j+1}^{n+1} = \tilde{v}_{i,j+1} - \Delta t \frac{p_{i,j+1} - p_{i,j}}{\Delta y_s}$$

$$v_{i,j}^{n+1} = \tilde{v}_{i,j} - \Delta t \frac{p_{i,j} - p_{i,j-1}}{\Delta y_s}$$

$$\frac{1}{\Delta x_u}\left(\left(\tilde{u}_{i+1,j} - \Delta t \frac{p_{i+1,j} - p_{i,j}}{\Delta x_s}\right) - \left(\tilde{u}_{i,j} - \Delta t \frac{p_{i,j} - p_{i-1,j}}{\Delta x_s}\right)\right) + \frac{1}{\Delta y_v}\left(\left(\tilde{v}_{i,j+1} - \Delta t \frac{p_{i,j+1} - p_{i,j}}{\Delta y_s}\right) - \left(\tilde{v}_{i,j} - \Delta t \frac{p_{i,j} - p_{i,j-1}}{\Delta y_s}\right)\right) = 0$$

$$A_p p_p + \sum_\ell A_\ell p_\ell = Q_{i,j}$$

$$A_N = A_S = \frac{\Delta x_u}{\Delta y_s}$$

$$A_E = A_W = \frac{\Delta y_v}{\Delta x_s}$$

$$A_P = -\sum_\ell A_\ell = -2\left(\frac{\Delta x_u}{\Delta y_s} + \frac{\Delta y_v}{\Delta x_s}\right)$$

$$Q_{i,j} = \frac{1}{\Delta t}\left((\tilde{u}_{i+1,j} - \tilde{u}_{i,j})\Delta y_v + (\tilde{v}_{i,j+1} - \tilde{v}_{i,j})\Delta x_u\right)$$

The terms $\tilde{u}$ and $\tilde{v}$ are intermediate values, yet to be corrected to reflect the current pressure gradient.

The pressure correction method is an iterative, fractional-stepping scheme in which the momentum equations and pressure equations are decoupled. While a number of variations of the method exist, the scheme used in this study is as follows (the equations below show the steps used for the x-momentum equation, the y-momentum equation uses an analogous process).

1. Solve for intermediate velocities by ignoring the pressure terms in the momentum equations

$$\tilde{u} = u^n + \frac{1}{2}\Delta t(3H^n - H^{n-1})$$

Here, H represents the right-hand side (RHS) of the unsteady discretization momentum equation. A second order Adams-Bashforth scheme is used to advance this intermediate velocity solution.

$$\int_{t_n}^{t_n+\Delta t} f\, dt = \Delta t(\frac{3}{2}f^n - \frac{1}{2}f^{n-1})$$

2. Solve for a pressure correction to satisfy continuity (a residual level of 1E-5 was required at each time-step)

$$A_p p_p + \sum_\ell A_\ell p_\ell = Q_{i,j}$$

3. Correct the intermediate velocity values with the newly established pressure gradient

$$u^{n+1} = \tilde{u} - \Delta t \nabla p$$

This iterative process is repeated until the errors in the solution have been sufficiently reduced. A reduction to 1E-8 was required for convergence during this study.

Solving the pressure Poisson equation presents the most significant task during the solution process. An alternating direction implicit (ADI) solver with SOR was used in conjunction with the Taylor algorithm, which is used to solve the resulting tridiagonal systems. The functions used

to assemble the ADI systems and solve them using the Thomas algorithm can be found in the Appendix.

Boundary conditions for the pressure equation at the West wall are applied as follows (the other boundaries are treated in a similar manner).

$$A_p p_p + \sum_{\ell} A_\ell p_\ell = Q_{i,j}$$

$$A_N = A_S = \frac{\Delta x_u}{\Delta y_s}$$

$$A_E = \frac{\Delta y_v}{\Delta x_s}$$

$$A_W = 0$$

$$A_P = -\sum_{\ell} A_\ell = -2\frac{\Delta x_u}{\Delta y_s} + \frac{\Delta y_v}{\Delta x_s}$$

$$Q_{i,j} = \frac{1}{\Delta t}((\tilde{u}_{i+1,j} - \tilde{u}_{i,j})\Delta y_v + (\tilde{v}_{i,j+1} - \tilde{v}_{i,j})\Delta x_u)$$

Here, the term $\tilde{u}_{i,j}$ is given by the boundary condition as

$$\tilde{u}_{i,j} = 0$$

The v-velocity at the West (and East) wall is not directly given but can be applied through the use of a ghost cell.

$$\frac{1}{2}(v_{1,j} + v_{2,j}) = 0 \qquad v_{1,j} = -v_{2,j}$$

At the top wall (the "lid" of the lid-driven cavity), a non-zero velocity (U_{lid}) is imparted to the fluid.

$$\frac{1}{2}(u_{i,n} + u_{i,n-1}) = U_{\text{lid}} \qquad u_{i,n} = 2U_{\text{lid}} - u_{i,n-1}$$

The Matlab program used to implement the methods described in this section is in Appendix of this report.

3 Results

A representative solution (Re=100, on a 128x128 grid), containing the fundamental flow structures, is shown in Figure 1.

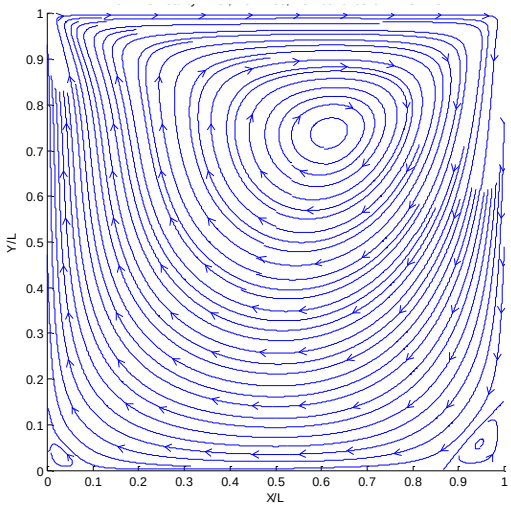

Figure 1 – Streamlines showing solution for Re = 100 on a 128x128 grid

3.1 Convergence Behavior

A series of simulations were run in order to explore the convergence characteristics of the system for a Reynolds number of 100. The error convergence history for each grid arrangement is shown in Figure 2, Figure 3 and Figure 4. The $L2$ norm of the change in solution of the u-velocity is plotted against non-dimensional solution time. Each plot shows the convergence history using three different time-step sizes.

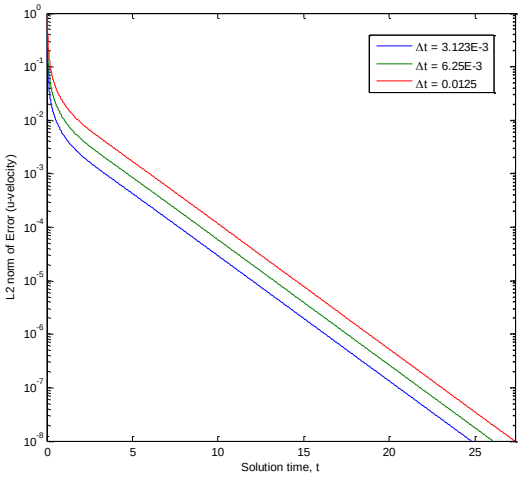

Figure 2 – Error history during convergence (Re = 100) on a 32x32 grid

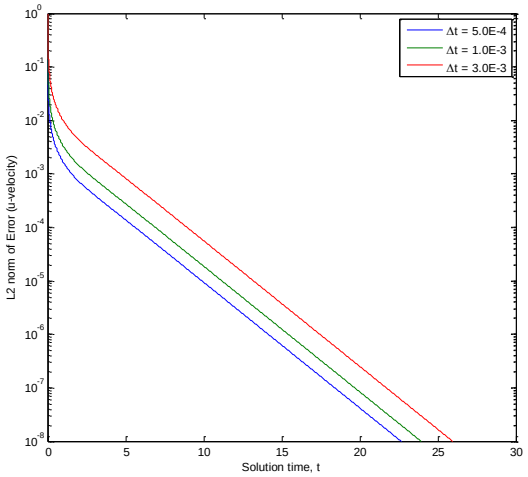

Figure 3 - Error history during convergence (Re = 100) on a 64x64 grid

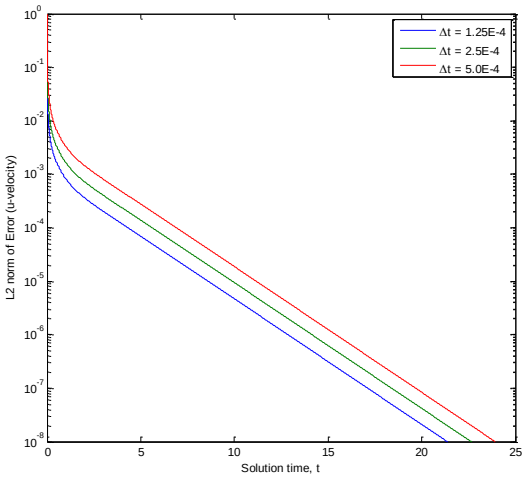

Figure 4 - Error history during convergence (Re = 100) on a 128x128 grid

The solution time required for convergence appears to reduce along with the grid spacing (Δx). The larger time-step values (Δt) require large solution times to converge.

CPU time required for convergence was heavily dependent on the number of grid points. Figure 5 shows the wall-time needed to converge solutions with grid sizes of 32x32 and 64x64. Convergence times for solutions with grids of 128x128 cells were not recorded due to a programing error, but were on the order of 6 hours (2E4 s).

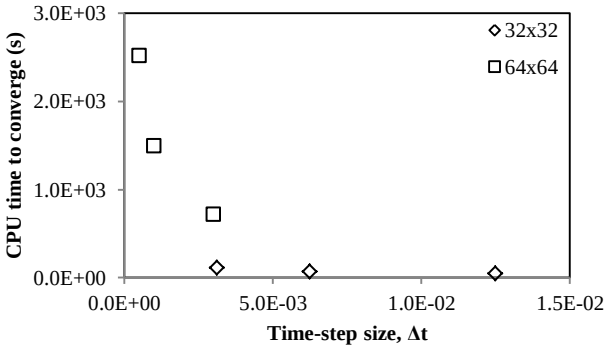

Figure 5 – CPU time to convergence of solution

3.2 Velocity Comparisons

Figure 6 and Figure 7 show computed velocity distributions at the midsections of the domain for a flow at Re = 100. The u-velocity is shown a $x = \frac{L}{2}$ and the v-velocity is shown at $y = \frac{L}{2}$. Data from a 129x129 grid presented by Ghia et al. is also shown for comparison [2]. All solutions (except from Ghia et al.) use a time-step of 5.0E-4.

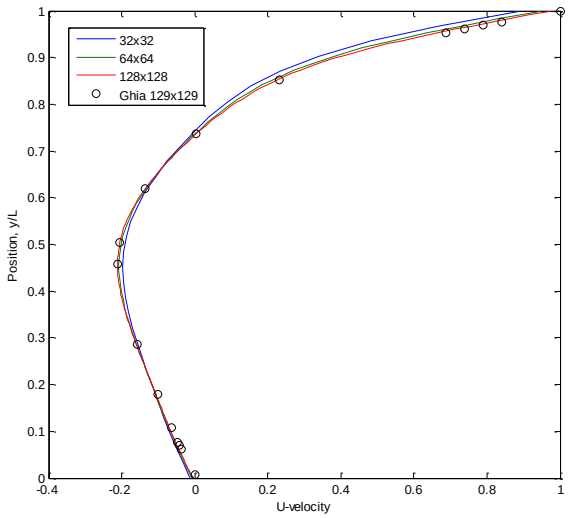

Figure 6 – U-velocity along a vertical bisector (Re = 100) with comparison to Ghia et al. [2]

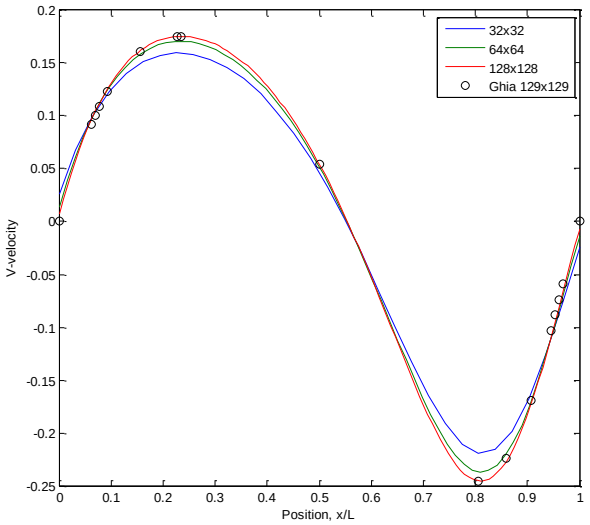

Figure 7 - V-velocity along a horizontal bisector (Re = 100) with comparison to Ghia et al. [2]

The u-velocity comparison, shown in Figure 6, shows only minor variation between the different solutions. The v-velocity, shown in Figure 7, shows a stronger grid dependence. The solution appears to become asymptotic as the grid spacing decreases.

3.3 Higher Reynolds Number Flows

A number of solutions were also computed at a Reynolds number of 1000, Figure 8 shows the solution computed using a 128x128 grid using a time-step of 0.00625.

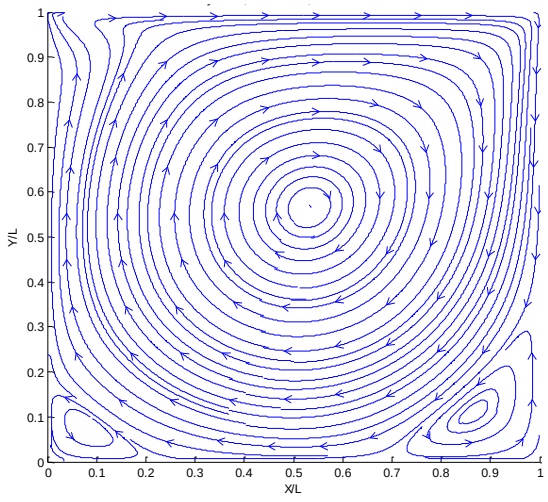

Figure 8 - Streamlines showing solution for Re = 1000 on a 128x128 grid

The 0.00625 time-step is an order of magnitude large than the largest time-step used for solutions with a Reynolds number of 100 on the same grid. This results from an effective lowering of the fluid viscosity, a term appears in the von Neumann stability condition. The solution required only 26618 time-steps to converge. This is significantly fewer than solutions on the same grid for Re = 100. This may be due to the large time-step's ability to quickly move past transient flow features.

Figure 9 and Figure 10 show computed velocity distributions at the midsections of the domain in the same manner as Figure 6 and Figure 7, but for Re = 1000. Again, the results from this study show good agreement with those of Ghia et al. [2].

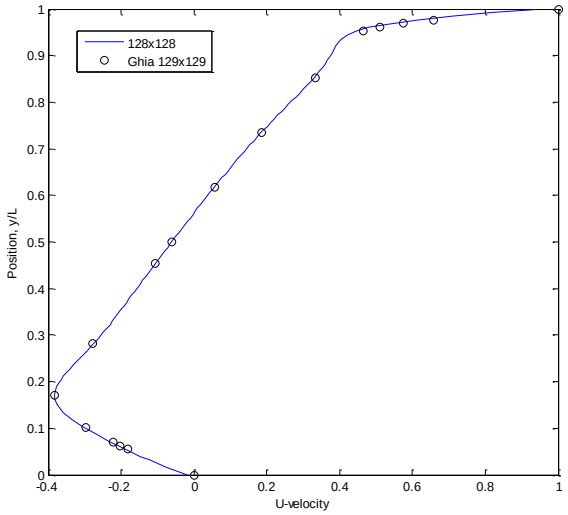

Figure 9 - U-velocity along a vertical bisector (Re = 1000) with comparison to Ghia et al. [2]

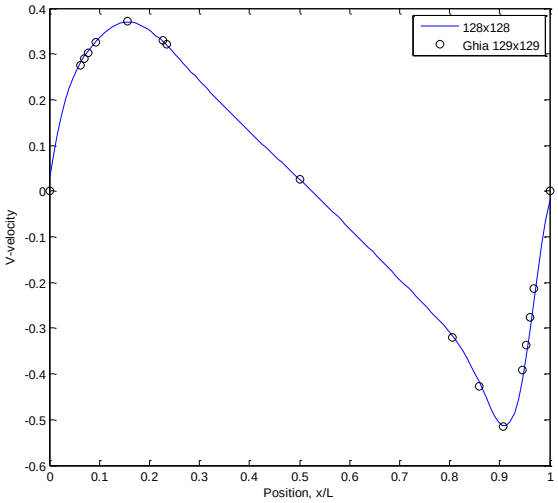

Figure 10 - V-velocity along a horizontal bisector (Re = 1000) with comparison to Ghia et al. [2]

4 Possible Improvements

The code for this study was written as part of a class project within a limited time-period. While it is fully capable of providing accurate solutions, significant upgrades could be made to improve the computational speed.

While Matlab offers a user-friendly environment for program development, it cannot compete with the computational efficiency of a language such a Fortran. The relatively simple task of translating the code for this project from Matlab to Fortran would provide a significant performance improvement (orders of magnitude increase in speed).

As far as the actual algorithms of the code are concerned, the bulk of the computational time is spent to solve the pressure Poisson equation (on the order of 75%). The function which assembles each ADI system has been vectorized to improve speed (a roughly 50% improvement was witnessed). The Thomas algorithm cannot be further vectorized or parallelized.

Multigrid methods have been shown to significantly accelerate the solutions of elliptic equations. During the process of an iterative solution, errors of different wavelengths are reduced at drastically different rates. Wavelengths which are small (compared to the grid spacing) are reduced significantly early in the solution process while low-frequency errors require many iterations to reduce. Errors, which were of a low frequency on one grid, can become high frequency errors on a grid with twice the spacing. Multigrid methods take advantage of this property by solving the solution (or reducing residuals) on grids of different densities.

Works Cited

[1] F. H. Harlow and J. E. Welch, "Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface," *THE PHYSICS OF FLUIDS*, vol. 3, pp. 2182-2189, 1965.

[2] U. Ghia, *et al.*, "High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method," *JOURNAL OF COMPUTATIONAL PHYSICS*, vol. 48, pp. 387-411, 1982.

Appendix

LidDrivenCavity.m

```
% Solves incompressible Navier-Stokes equations
for lid-driven cavity using
% finite volume discretization with staggered
grid.  Pressure-correction
% method (MAC) with Adams-Bashforth 2-step time
integration, ADI and Thomas
% algorithm to solve the Poisson pressure
equation.
% INPUTS
% Re        Reynolds number
% dt        Timestep size(s)
% w         Over-relaxation factor
% nc        Number of finite volume (area) cells
% update    Update Heads Up Display (HUD) every x
timesteps
% tmax      Maximum solution time
% ipmax     Maximum inner iterations to solve the
Poisson pressure eq.

clc
clf
clear
wt0 = cputime;

Re = 100;
dt = 0.000125;
w = 1.1;
nc = 128;
update = 100;
tmax = 1000;
ipmax = 5000;

Ulid = 1;
n = nc+2;
L = 1;
nx = nc;
ny = nc;
hx = 1/nx;
hy = 1/ny;
delta = L/nc;
dxu = delta;dxs = delta;dyv = delta;dys = delta;
% constant grid spacing
x = 0:1/(nc-1):1;
y = 0:1/(nc-1):1;

% x(1:nx+1)=(0:nx);
% y(1:ny+1)=(0:ny);

[X,Y] = meshgrid(x,y);
tstring = ['Re = ', num2str(Re), ...
    ', Number of cells = ',
num2str(nc),'x',num2str(nc)];


%% Initialize the solution
% points used:
%   scalar/pressure
%       i = 2:(nc+2)-1
%       j = 2:(nc+2)-1
%   x-momentum
%       i = 3:(nc+2)-1
%       j = 2:(nc+2)-1
%   y-momentum
%       i = 2:(nc+2)-1
%       j = 3:(nc+2)-1

p = zeros(nc+2,nc+2);
u = zeros(nc+2,nc+2);
v = zeros(nc+2,nc+2);
us = u;
vs = v;
ps = p;
p_old = p;
u_old = u;
v_old = v;

S(1:n-1,1:n-1) = 0;

%% Solution
```

```
% Coefficients for poisson pressure equation
As = zeros(n,n);
As(:,3:n-1) = dxu/dys;
An = zeros(n,n);
An(:,2:n-2) = dxu/dys;
Aw = zeros(n,n);
Aw(3:n-1,:) = dyv/dxs;
Ae = zeros(n,n);
Ae(2:n-2,:) = dyv/dxs;
Ap = -(Aw + Ae + As + An);


iter.p = 2;
iter.o = 1;

res.p(1) = 0;
tp = 0;

for t = 0:dt:tmax

    %   PREDICTOR
    [u,v] = setBCs(u,v,Ulid);
    [Hx,Hy] = HHP(u,v,dxu,dxs,dyv,dys,Re);
    if iter.o == 1
        us(2:n-1,1:n-1) = u(2:n-1,1:n-1) +
dt*Hx(2:n-1,1:n-1);
        vs(1:n-1,2:n-1) = v(1:n-1,2:n-1) +
dt*Hy(1:n-1,2:n-1);
    else
        [Hx_old,Hy_old] =
HHP(u_old,v_old,dxu,dxs,dyv,dys,Re);
        us(2:n-1,1:n-1) = u(2:n-1,1:n-1) ...
            + dt/2*(3*Hx(2:n-1,1:n-1) -
Hx_old(2:n-1,1:n-1));
        vs(1:n-1,2:n-1) = v(1:n-1,2:n-1) ...
            + dt/2*(3*Hy(1:n-1,2:n-1) -
Hy_old(1:n-1,2:n-1));
    end
    u_tp = us;
    v_tp = vs;

    % CORRECTOR
    for i = 2:n-1
        for j = 2:n-1
            S(i,j) = 1/dt*((us(i+1,j) -
us(i,j))*dyv ...
                + (vs(i,j+1) - vs(i,j))*dxu);
        end
    end

    tpp = cputime;
    for c = 1:ipmax
        iter.p = iter.p + 1;
        [p,res.p(iter.p)] =
PoissonADI2P(p,n,Ap,An,As,Ae,Aw,S,w);
        if res.p(iter.p) < 1e-5
            break
        elseif isnan(res.p(iter.p)) == 1
            disp('SOLUTION DIVERGED')
            beep
            return
        end
    end
    if c == ipmax
        disp('Warning: Pressure unconverged')
        disp((res.p(iter.p)))
    end
    tp = tp + (cputime - tpp);

    % CORRECTION
    u(2:n-1,2:n-1) = us(2:n-1,2:n-1) ...
        - dt/dxs*(p(2:n-1,2:n-1) - p(1:n-2,2:n-
1));
    v(2:n-1,2:n-1) = vs(2:n-1,2:n-1) ...
        - dt/dys*(p(2:n-1,2:n-1) - p(2:n-1,1:n-
2));


    err.p(iter.o) = norm(p - p_old);
    err.u(iter.o) = norm(u - u_old);
    err.v(iter.o) = norm(v - v_old);

    if err.u(iter.o) < 1E-8 && err.v(iter.o) < 1E-
8
        beep
        wt = cputime - wt0;
        break
    end

    p = p - p(5,5); % normalize pressure

    p_old = p;
    u_old = u;
    v_old = v;


    % Heads-up display
    if rem(t,dt*update) == 0 && t ~= 0
        HUD(t,wt0,iter,err,tstring)
        disp(tp)
    end
    iter.o = iter.o + 1;
end

uc = 0.5*(u(2:end-1,2:end-1) + u(3:end,2:end-1));
vc = 0.5*(v(2:end-1,2:end-1) + v(2:end-1,3:end));

figure(2)
% uu = sqrt(u.^2 + v.^2);
% contourf(X,Y,uu(2:end-1,2:end-1)',20)
streamslice(X,Y,uc',vc',2)
axis([0,L,0,L])
title(['Lid Driven Cavity Error, ',tstring])
xlabel('X/L')
ylabel('Y/L')

wt = cputime - wt0;

fname =
['Re',num2str(Re),'nc',num2str(nc),'dt',num2str(dt
)];
fname = strrep(fname,'.','p');
save(fname)
```

HHP.m

```
 function [Hx,Hy] = HHP(u,v,dxu,dxs,dyv,dys,Re)

n = length(u);

% X-Momentum
ue2 = (1/2.*(u(4:n,2:n-1)+u(3:n-1,2:n-1))).^2;
uw2 = (1/2.*(u(2:n-2,2:n-1)+u(3:n-1,2:n-1))).^2;
du2dx = (ue2-uw2)./dxs;

un = 1/2.*(u(3:n-1,2:n-1) + u(3:n-1,3:n));
vn = 1/2.*(v(2:n-2,3:n) + v(3:n-1,3:n));
us = 1/2.*(u(3:n-1,1:n-2) + u(3:n-1,2:n-1));
vs = 1/2.*(v(2:n-2,2:n-1) + v(3:n-1,2:n-1));
```

```
duvdy = (un.*vn - us.*vs)./dyv;

d2udx2 = (u(4:n,2:n-1) - 2.*u(3:n-1,2:n-1) +
u(2:n-2,2:n-1))./(dxu*dxs);
d2udy2 = (u(3:n-1,3:n) - 2.*u(3:n-1,2:n-1) +
u(3:n-1,1:n-2))./(dys*dyv);

Hx(3:n-1,2:n-1) = -du2dx - duvdy + (1/Re).*(d2udx2
+ d2udy2);


% Y-Momentum
vn2 = (1/2.*(v(2:n-1,4:n) + v(2:n-1,3:n-1))).^2;
vs2 = (1/2.*(v(2:n-1,3:n-1) + v(2:n-1,2:n-2))).^2;
dv2dy = (vn2 - vs2)./dys;

ue = 1/2.*(u(3:n,3:n-1) + u(3:n,2:n-2));
ve = 1/2.*(v(2:n-1,3:n-1) + v(3:n,3:n-1));
uw = 1/2.*(u(2:n-1,3:n-1) + u(2:n-1,2:n-2));
vw = 1/2.*(v(1:n-2,3:n-1) + v(2:n-1,3:n-1));
duvdx = (ue.*ve - uw.*vw)./dxu;

d2vdx2 = (v(3:n,3:n-1) - 2.*v(2:n-1,3:n-1) +
v(1:n-2,3:n-1))./(dxs*dxu);
d2vdy2 = (v(2:n-1,4:n) - 2.*v(2:n-1,3:n-1) +
v(2:n-1,2:n-2))./(dyv*dys);

Hy(2:n-1,3:n-1) = -dv2dy - duvdx + (1/Re).*(d2vdx2
+ d2vdy2);
```

PoissonADI2P.m

```
function [phi,res] =
PoissonADI2P(phi,n,Ap,An,As,Ae,Aw,S,w)



%% Perform ADI
for j = 2:n-1
    RHS1(2:n-1) = (1-w).*Ap(2:n-1,j).*phi(2:n-1,j)
...
        + w.*(-(An(2:n-1,j).*phi(2:n-1,j+1) ...
        + As(2:n-1,j).*phi(2:n-1,j-1))+ S(2:n-
1,j));
    phi(2:n-1,j) = Tri2(w.*Aw(2:n-1,j),Ap(2:n-
1,j),w.*Ae(2:n-1,j),RHS1(2:n-1));
end
for i = 2:n-1
    RHS2(2:n-1) = (1-w).*Ap(i,2:n-1).*phi(i,2:n-1)
...
        + w.*(-(Ae(i,2:n-1).*phi(i+1,2:n-1) ...
        + Aw(i,2:n-1).*phi(i-1,2:n-1)) + S(i,2:n-
1));
    phi(i,2:n-1) = Tri2(w.*As(i,2:n-1),Ap(i,2:n-
1),w.*An(i,2:n-1),RHS2(2:n-1));
end

%% Find Residual
rho(2:n-1,2:n-1) = S(2:n-1,2:n-1) ...
    - ( Ap(2:n-1,2:n-1).*phi(2:n-1,2:n-1) ...
    + An(2:n-1,2:n-1).*phi(2:n-1,3:n) ...
    + As(2:n-1,2:n-1).*phi(2:n-1,1:n-2) ...
    + Ae(2:n-1,2:n-1).*phi(3:n,2:n-1) ...
    + Aw(2:n-1,2:n-1).*phi(1:n-2,2:n-1));

res = sqrt(sum(sum(rho.^2))/n);
```

Tri2.m

```
function x = Tri2(a,b,c,f)

n = length(b);

beta(1) = b(1);
gamma(1) = f(1)/beta(1);
for q = 2:n
    beta(q) = b(q) - a(q)*c(q-1)/beta(q-1);
    gamma(q) = (-a(q)*gamma(q-1) +f(q))/beta(q);
end

x(n) = gamma(n);
for q = n-1:-1:1
    x(q) = gamma(q) - x(q+1)*c(q)/beta(q);
end
```

HUD.m

```
function [] = HUD(t,wt0,iter,err,tstring)

io = 1:iter.o;
semilogy(io,err.p,io,err.u,io,err.v)
title(['Lid Driven Cavity Error, ',tstring])
legend('pressure','u-velocity','v-velocity')
xlabel('Iteration')
ylabel('Error')
drawnow expose

wt = cputime - wt0;
clc
fprintf('Solution time (s): %4.3e\n',t);
fprintf('Wall time (s)    : %4.3e\n',wt);
fprintf('Speed            : %4.3e\n\n',t/wt);
fprintf('Errors\n')
fprintf('                 p:
%4.3e\n',err.p(iter.o));
fprintf('                 u:
%4.3e\n',err.u(iter.o));
fprintf('                 v:
%4.3e\n',err.v(iter.o));
```

SetBCs.m

```
function [u,v] = setBCs(u,v,Ulid)

% West
v(1,:) = -v(2,:);
% u(1,:) = 0;
u(2,:) = 0;
% East
v(end,:) = -v(end-1,:);
u(end,:) = 0;
% South
u(:,1) = - u(:,2);
% v(:,1) = 0;
v(:,2) = 0;
% North
u(:,end) = (2*Ulid - u(:,end-1));
v(:,end) = 0;
```