CFD Coding Project Technical Memo
EMEC 435: Computational Fluid Dynamics
Authors: Montana Marks
Instructor: Erick Johnson

Introduction

Computational Fluid Dynamics is an ever-changing and lively field. With the recent and continuous advancements in computing technology, engineers are able to test and analyze more complex and difficult problems. For this reason, it is important for students to understand the fundamental principles of this complex and unique field.

In this project we were tasked with solving the 2D Incompressible Navier-Stokes equations. This was achieved using an explicit, time marching, predictor corrector method paired with a computational mesh to solve the governing equations. The pressure, and velocities are stored on a staggered grid and are predicted, corrected and then updated for every time step. The code allows for the solving of a Lid Driven cavity flow problem. Written with the intention of allowing the user to change some parameters, the code allows for the selection of fluid density, dynamic viscosity, grid size, side length, simulation time, and lid velocity. Once the user runs the code, a pressure and velocity field will be displayed and updated every time step, this diagram will also be saved as a .avi movie file for later examination.

Procedure

There are several methods you can use to solve this problem; the following procedure is just one way to do so. The resulting code will be broken into 10 steps, with steps 3-9 being performed inside of the time loop. For a more detailed explanation of the various discretizations refer to Anderson's "Computational Fluid Dynamics".

Step 1: Define constants, create mesh, define mesh sizes, pre-allocate, and create initial conditions.

Step 2: Create Laplacian Matrix; this is done outside of the time loop near the beginning of the code.

Step 3: Calculate the u and v momentum predictors

Step 4: Calculate the right hand side of the Poisson equation using the previously calculated predictor velocities.

Step 5: Find the pressure solution using matrix inversion of the Laplacian and right hand side matrix calculated in step 4.

Step 6: Convert the pressure solution into the mesh.

Step 7: Correct for the x & y velocities using the predictor velocities and the previously calculated pressure.

Step 8: Set the boundary conditions.

Step 9: Post processing setup,

Step 10: Open, Write, and Close the video from post processing.

Results

As we can see in the figures below, the lid driven cavity simulation performed as expected. Shortly after the simulation was initialized a small vortex was created on top right wall. It is believed that this vortex is the result of the transfer of momentum from the top layer of moving fluid to the lower layers. As time goes on we see that the small vortex slowly grows into a large eddy located near the center of the cavity.

Overall the code performed well, however there are several limitations. For one the code has a rather high computation time. Converting the Laplacian operator into a sparse matrix could reduce computation time. Furthermore, the code is only useful for very specific problems; although more complex problems could be solved it would require large amounts of modifications to the boundary conditions in order to solve them.

Appendix A: Tables and Figures

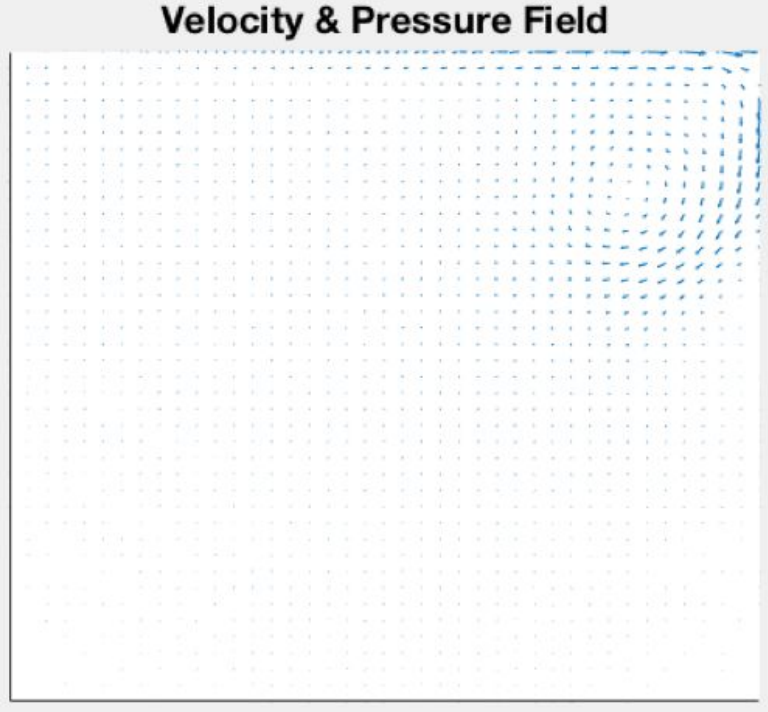

Figure 1.A: Velocity field near the beginning of the simulation

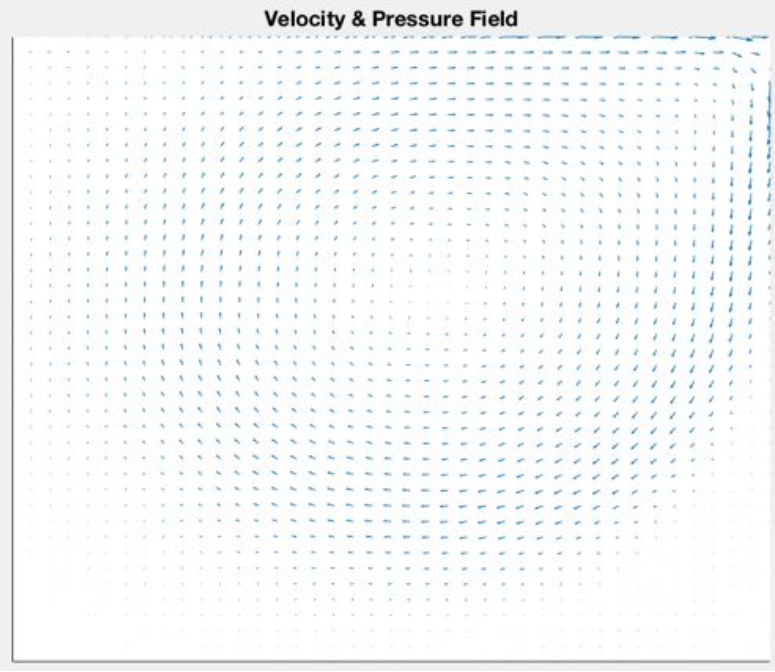

Figure 2.A: Velocity field near the end of the simulation.

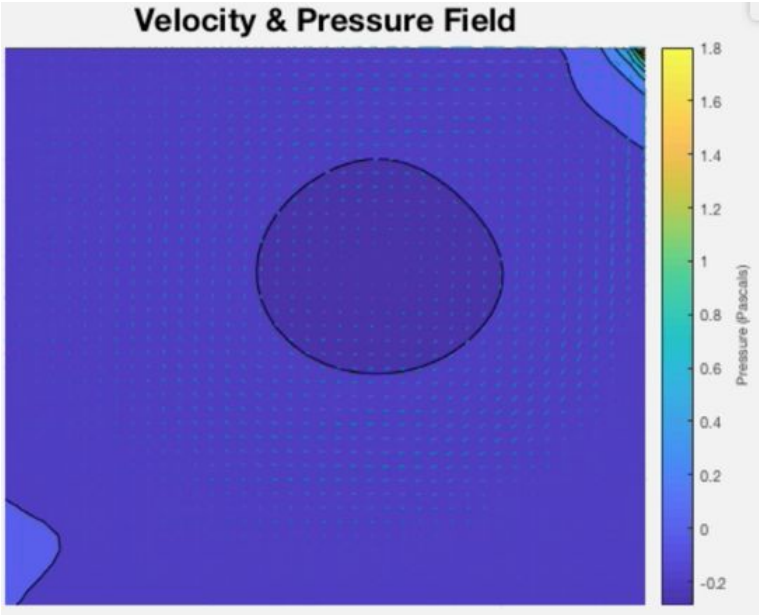

Figure 3.A: Pressure and Velocity field near end of simulation.

Appendix B: Entire Code Used for Solution

```
%                          **** READ ME ****

% This is a Compuational Fluid Dynamics Solver.
% This code utilizes time marching central differenced Navier Stokes equations
% combined with the pressure correnction method and a staggered grid to solve
% a lid driven incompressible or compressible cavity flow problem. The solution will
% output into a .avi formatted file. It is neccesary for the user to
% define the fluid properties, such as Density, Kinematic Viscosity,
% Number of grid divisions in both x & y, the desired time step,
% the total simulation time and the side lengths.
% All of the various parameters will effect the solution.


clear; clc;

% Constants

nu=0.001; % Kinematic Viscosity of water @ 40C
rho=1.225; % Density of water @ 40c

% Define Specified Parameters

nx=41; % Number of Grid points in x
ny=41; % Number of Grid Points in y
Lx=1; % Length in x Direction
Ly=1; % Length in y Direction
dt=0.001; % Time Step Size
t_final=10; % Simulation Time (s)


% Index extents
imin=2;
imax=imin+nx-1;
jmin=2;
jmax=jmin+ny-1;

% Create mesh
```

```
x(imin:imax+1)=linspace(0,Lx,nx+1);
y(jmin:jmax+1)=linspace(0,Ly,ny+1);

%Create Mesh Middle spaces
xm(imin:imax)= 0.5*(x(imin:imax)...
    +x(imin+1:imax+1));
ym(jmin:jmax)=0.5*(y(jmin:jmax)...
    +y(jmin+1:jmax+1));

% Create mesh sizes
dx=x(imin+1)-x(imin);
dy=y(jmin+1)-y(jmin);
dxi=1/dx;
dyi=1/dy;

% Number of timesteps
Nt=t_final/dt;

% Preallocate
p=zeros(imax,jmax);
us=zeros(imax+1,jmax+1);
vs=zeros(imax+1,jmax+1);
R=zeros(imax,1);
u=zeros(imax+1,jmax+1);
v=zeros(imax+1,jmax+1);
t=zeros(1,Nt);
Z=peaks(nx);
L=zeros(nx*ny,nx*ny);


% Innitial conditions
t(1)=0; % Innitial Time
u_bot(1)=0; % Innitial Velocity for Bottom wall
u_top(1)=2; % Innitial Velocity for Top Wall
v_lef(1)=0; % Innitial Velocity for Left Wall
v_rig(1)=0; % Innitial Velocity for right wall
fr=1; % Frame Rate

% Creat Laplacian operator for solving pressure Poisson equation
```

```
for j=1:ny
   for i=1:nx
      L(i+(j-1)*nx, i+(j-1)*nx)=2*dxi^2+2*dyi^2;
      for ii=i-1:2:i+1
         if (ii>0 && ii<=nx) % Interior points
            L(i+(j-1)*nx,ii+(j-1)*nx)=-dxi^2;
         else               % Neuman conditions on boundary
            L(i+(j-1)*nx,i +(j-1)*nx)= ...
               L(i+(j-1)*nx,i +(j-1)*nx)-dxi^2;
         end
      end
      for jj=j-1:2:j+1
         if (jj>0 && jj<=ny) % Interior points
            L(i+(j-1)*nx,i+(jj-1)*nx)=-dyi^2;
         else               % Neuman conditions on boundary
            L(i+(j-1)*nx,i +(j-1)*nx)= ...
               L(i+(j-1)*nx,i +(j-1)*nx)-dyi^2;
         end
      end
   end
end

L(1 ,:)=0;
L(1 ,1)=1;

% solver
 while t <= t_final

       % update time
      t = t + dt;

      u_top(1)=2;
   % u Momentum Predictor
   for j = jmin:jmax

      for i = imin+1:imax

         A = (nu*(u(i-1,j)-2*u(i,j)+u(i+1,j))*dxi^2 ...
            +nu*(u(i,j-1)-2*u(i,j)+u(i,j+1))*dyi^2 ...
```

```
            -u(i,j)*(u(i+1,j)-u(i-1,j))*0.5*dxi ...
            -(0.25*(v(i-1,j)+v(i-1,j+1)+v(i,j)+v(i,j+1)))...
            *(u(i,j+1)-u(i,j-1))*0.5*dyi);

        us(i,j)=u(i,j)+dt*A;

    end
end

% v Momentum predictor
for j = jmin+1:jmax

    for i = imin:imax

        B = (nu*(v(i-1,j)-2*v(i,j)+v(i+1,j))*dxi^2 ...
            +nu*(v(i,j-1)-2*v(i,j)+v(i,j+1))*dyi^2 ...
            -(0.25*(u(i,j-1)+u(i+1,j-1)+u(i,j)+u(i+1,j)))...
            *(v(i+1,j)-v(i-1,j))*0.5*dxi...
            -v(i,j)*(v(i,j+1)-v(i,j-1))*0.5*dyi);

        vs(i,j)=v(i,j)+dt*B;
    end
end

% Compute right-hand side (R) of the Pressure Poisson Equation using
% the predictor velocities (vs) & (us).

n=0;

for j=jmin:jmax
    for i=imin:imax
        n=n+1;
        R(n)=-rho/dt* ...
            ((us(i+1,j)-us(i,j))*dxi ...
            +(vs(i,j+1)-vs(i,j))*dyi);
    end
end

% Find pressure solution for matrix inversion, Using Laplacian
```

```
% Operator (L) and Right hand side of Pressure Poisson Equations (R)

pv=L\R;

n=0;

p=zeros(imax,jmax);

% Convert pressure into mesh

for j=jmin:jmax
   for i=imin:imax
      n=n+1;
      p(i,j)=pv(n);
   end
end

% Corrector of x^(n+1) & y^(n+1) velocity

for j=jmin:jmax
   for i=imin+1:imax
      u(i,j)=us(i,j)-dt/rho*(p(i,j)-p(i-1,j))*dxi;
   end
end
for j=jmin+1:jmax
   for i=imin:imax
      v(i,j)=vs(i,j)-dt/rho*(p(i,j)-p(i,j-1))*dyi;
   end
end

% Set Boundary Conditions
u(:,jmin-1)=u(:,jmin)-2*(u(:,jmin)-u_bot);
u(:,jmax+1)=u(:,jmax)-2*(u(:,jmax)-u_top);
v(imin-1,:)=v(imin,:)-2*(v(imin,:)-v_lef);
v(imax+1,:)=v(imax,:)-2*(v(imax,:)-v_rig);

% post-processing... can also include within each time-step

figure(1); clf(1);
```

```
    title('Velocity & Pressure Field','fontsize',20);
  xlim([0,Lx]);
  ylim([0,Ly]);
  hold on
  %contourf(x(imin:imax),y(jmin:jmax),p(imin:imax,jmin:jmax)');
  quiver(x(imin:imax),y(jmin:jmax),...
    u(imin:imax,jmin:jmax)',v(imin:imax,jmin:jmax)','filled');
  xmax=Lx-dx;
  ymax=Ly-dy;
  % Set Limits on X & Y Axis
  xlim([0 xmax]);
  ylim([0 ymax]);
  % Remove Tick Marks on X & Y Axis
  set(gca,'YTick',[]);
  set(gca,'XTick',[]);
  hold on
  cb = colorbar;
  cb.Label.String = 'Pressure (Pascals)';
  drawnow

  % Update Frame Rate for Video
  Y(fr) = getframe(gcf);
  fr=fr+1;

end

video=VideoWriter('CFD_Lid_Driven.avi','Uncompressed AVI');
open(video);
writeVideo(video,Y);
close(video);
```

Appendix C: References

Anderson, J. (1995) “Computational Fluid Dynamics”. New York, NY: McGraw-Hill

Owkes M. (2015) “A Guide to writing your first CFD solver”. Bozeman, MT