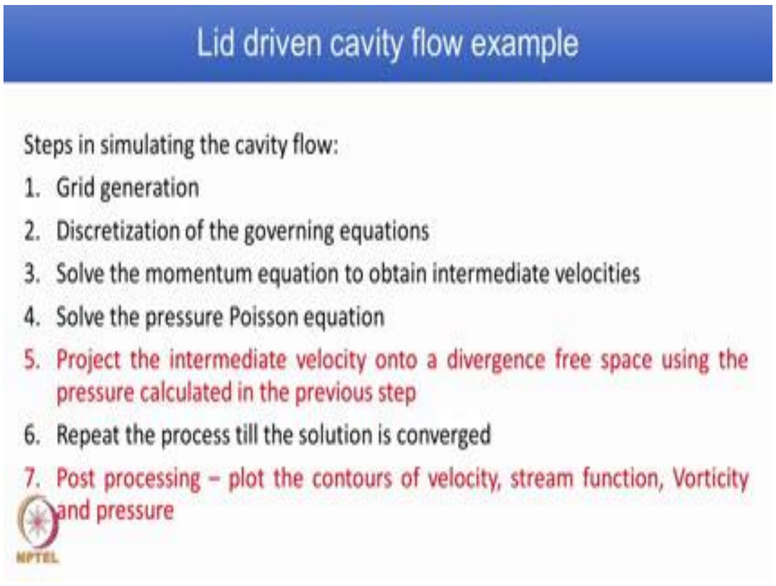
Foundation of Computational Fluid Dynamics

Dr. S. Vengadesan
Department of Applied Mechanics
Indian Institute of Technology, Madras

Lecture - 42

Greetings and it is my pleasure to welcome you again to this course on CFD. This is the last module for this week. In this module, we mentioned, we will take example problem of flow in a lid driven cavity, and explained step-by-step steps involved in CFD, discretization procedure, corresponding display of working code. In the last module, we are going to complete this procedure and in addition we are also going to see different post processing and corresponding display of code.

(Refer Slide Time: 01:02)



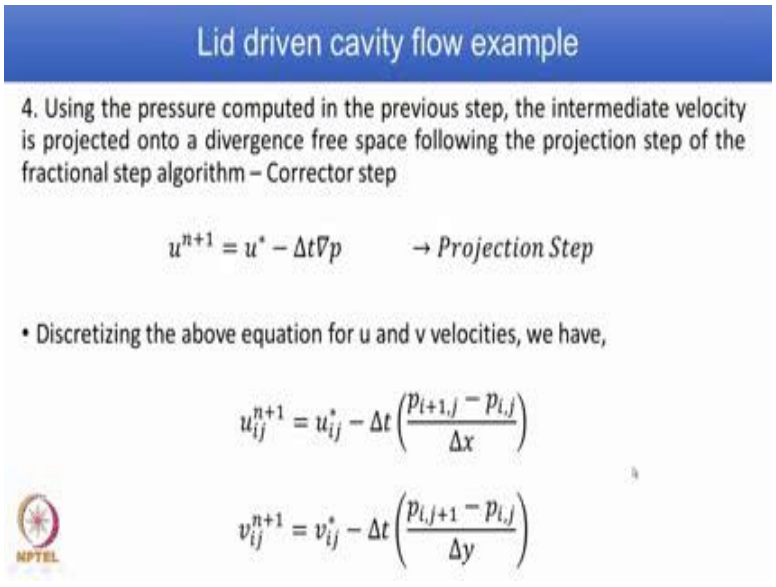
Lid driven cavity flow example

Steps in simulating the cavity flow:

1. Grid generation
2. Discretization of the governing equations
3. Solve the momentum equation to obtain intermediate velocities
4. Solve the pressure Poisson equation
5. Project the intermediate velocity onto a divergence free space using the pressure calculated in the previous step
6. Repeat the process till the solution is converged
7. Post processing – plot the contours of velocity, stream function, Vorticity and pressure

Steps involved in doing the simulation of flow in lid driven cavity are listed here. We already mentioned about this in other module as well. Hence the fifth step is the one we are going to see in particularly in this module. Fifth step is project the intermediate velocity onto a divergence free space; using the pressure calculated in the projection step. So this particular step is otherwise also called correction step. Then you repeat this process until the solution has reached the convergence condition. Once you obtained the velocity stream then you do post processing to get different information about the flow.

(Refer Slide Time: 02:00)



Lid driven cavity flow example

4. Using the pressure computed in the previous step, the intermediate velocity is projected onto a divergence free space following the projection step of the fractional step algorithm – Corrector step

$$u^{n+1} = u^{*} - \Delta t \nabla p \qquad \rightarrow \textit{Projection Step}$$

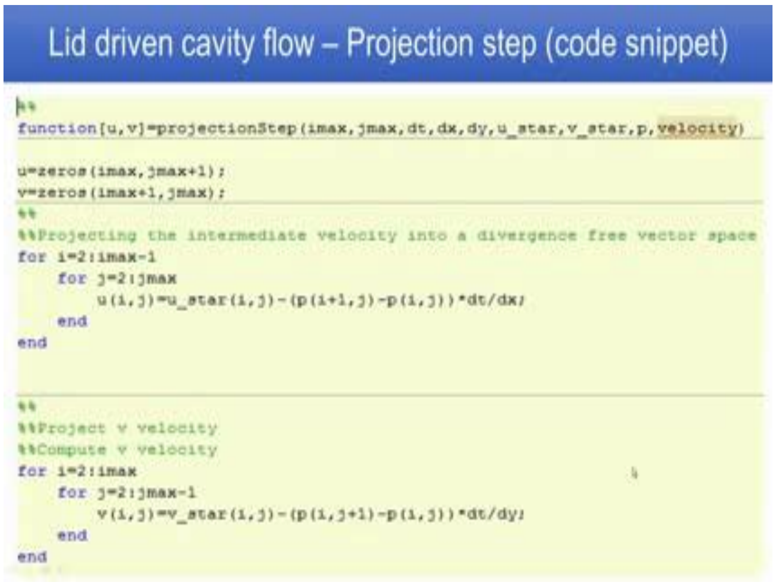
- Discretizing the above equation for u and v velocities, we have,

$$u_{ij}^{n+1} = u_{ij}^{*} - \Delta t\left(\frac{p_{i+1,j} - p_{i,j}}{\Delta x}\right)$$

$$v_{ij}^{n+1} = v_{ij}^{*} - \Delta t\left(\frac{p_{i,j+1} - p_{i,j}}{\Delta y}\right)$$

So, in the projection step method, we already seen predicted step solving the pressure Poisson equation to get pressure everywhere. Using this pressure computed in the pressure Poisson equation, the intermediate velocity which are calculated without considering the pressure term, and they are also referred as u star, v star; these two velocities are projected onto the divergence free space using the fractional step algorithm. And this particular step is called correction step and it is listed here. So, u at n plus 1 that is a new velocity equal to u star minus delta t into del p; similarly for v star as well. And if you discretize the above equation, for u as well as v velocity, then we have as shown here that is u at i comma j new value at n plus 1 equal to u i comma j star minus delta t p i plus 1comma j minus p i comma j by delta x. Similarly for v, so v at i comma j n plus 1 that is the new value equal to v i comma j star that is the intermediate velocity at the same i comma j location minus delta t p i comma j plus 1 minus p i comma j by delta y.

(Refer Slide Time: 03:50)

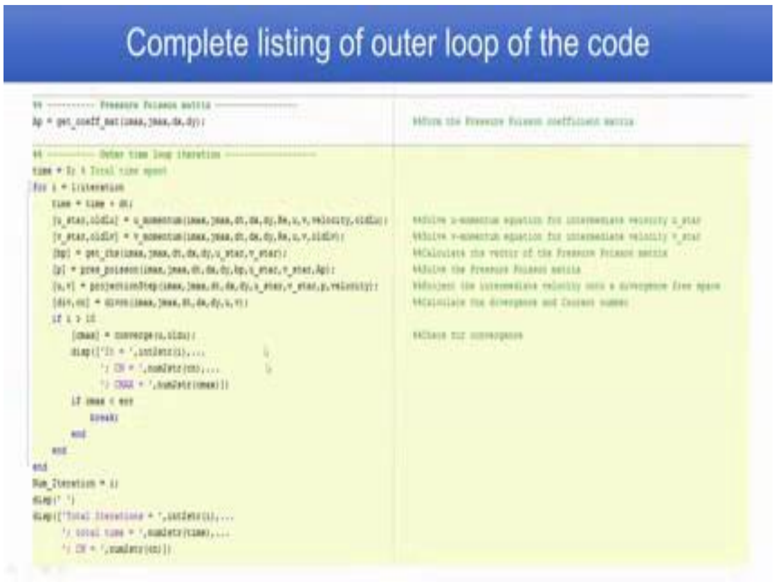


Lid driven cavity flow – Projection step (code snippet)

```
%%
function[u,v]=projectionStep(imax,jmax,dt,dx,dy,u_star,v_star,p,velocity)

u=zeros(imax,jmax+1);
v=zeros(imax+1,jmax);
%%
%%Projecting the intermediate velocity into a divergence free vector space
for i=2:imax-1
    for j=2:jmax
        u(i,j)=u_star(i,j)-(p(i+1,j)-p(i,j))*dt/dx;
    end
end


%%
%%Project v velocity
%%Compute v velocity
for i=2:imax
    for j=2:jmax-1
        v(i,j)=v_star(i,j)-(p(i,j+1)-p(i,j))*dt/dy;
    end
end
```

You will see the corresponding code here. So, function u v projection step i max, j max, dt, dx, u star, v star, p, velocity; then u is already initialized, and v is also initialized with corresponding number of required array size; for u, it is i max j max plus 1; for v, it us i max plus 1comma j max. Now we do the projecting the intermediate velocity onto a divergence free vector space first for u velocity next this lines are for. So, for i equal to 2 to i max minus 1, and for j 2 to j max u i ,j equal to u star i comma j minus p i plus 1 comma j minus p i j into delta t by delta x. In the form of code, it is dt by dx. These are same as what we have seen in previous slide in the form of equation. Similarly, now for the v velocity, so for i 2 to i max, and for j 2 to j max minus 1 v i comma j equal to v star i comma j minus p i comma j plus 1 minus p i comma j multiplying by dt by dy.

(Refer Slide Time: 05:22)



Complete listing of outer loop of the code

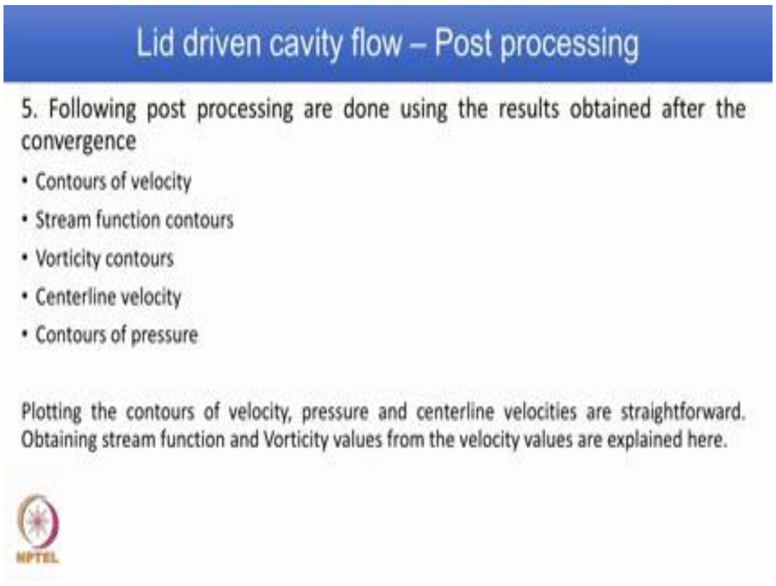
```
%% ---------- Pressure Poisson matrix ----------------
Ap = get_coeff_mat(imax,jmax,dx,dy);                                        %%Form the Pressure Poisson coefficient matrix

%% ---------- Outer time loop iteration ------------------
time = 0; % Total time spent
for i = 1:iteration
    time = time + dt;
    [u_star,olddu] = u_momentum(imax,jmax,dt,dx,dy,Re,u,v,velocity,oldu);  %%Solve u-momentum equation for intermediate velocity u_star
    [v_star,oldv] = v_momentum(imax,jmax,dt,dx,dy,Re,u,v,oldv);             %%Solve v-momentum equation for intermediate velocity v_star
    [bp] = get_rhs(imax,jmax,dt,dx,dy,u_star,v_star);                      %%Calculate the vector of the Pressure Poisson matrix
    [p] = pres_poisson(imax,jmax,dt,dx,dy,bp,u_star,v_star,Ap);            %%Solve the Pressure Poisson matrix
    [u,v] = projectionStep(imax,jmax,dt,dx,dy,u_star,v_star,p,velocity);   %%Project the intermediate velocity onto a divergence free space
    [div,cn] = divcn(imax,jmax,dt,dx,dy,u,v);                              %%Calculate the divergence and Courant number
    if i > 10
        [cmax] = converge(u,oldu);                                          %%Check for convergence
        disp(['It = ',int2str(i),...
              '; CN = ',num2str(cn),...
              '; CMAX = ',num2str(cmax)])
        if cmax < err
            break;
        end
    end
end
Num_Iteration = i;
disp(' ')
disp(['Total Iterations = ',int2str(i),...
    '; total time = ',num2str(time),...
    '; CN = ',num2str(cn)])
```

And what is shown here is complete listing of outer loop code. So, we start from here, in right pressure Poisson coefficient matrix. So, A p equal to get coefficient matrix i max comma j max comma dx comma dy. Then outer loop iteration, time is equal to zero which is, you start the iteration by initializing time. So, first it is time equal to zero then for i equal to one to iteration that is number of iteration you can decide or the convergence criteria you can decide. So, time is equal to time plus dt which is also defined u star comma old u equal to u momentum and corresponding argument; similarly v star , old v equal to v momentum and corresponding arguments. So, these two lines represent solving the u momentum equation for respective velocity that is u star and v star then we know pressure Poisson equation has right side term and left side term.

So, this bp is calculating the right side term of the pressure Poisson equation, we have seen them independently in detail in previous modules. What you are seen here is the complete listing of outer loop. So, bp equal to get R.H.S i max comma j max comma dt comma dx comma du comma dy comma u star and v star. So, this line calculate the R.H.S vector of pressure Poisson matrix then you solve the pressure Poisson equation that is what shown here. Then you project the step, this we are just now seen to get u and v. Once you get u and v at the new time level after the final correction step, you calculate for the divergence and Courant number. So, divergence, cn, Cn represents Courant number, and there is separate argument for the divergence div, cn, i max, j max, dt, dx, dy, u, v.

So, you can decide if the number of iteration is more than the desire number of iteration or you can decide based on Courant criteria. So, c max is the maximum iteration, so converge u comma old u. And you can also have some display on the screen for checking the progress of the code. S it will display this particular lines will display iteration number, Courant number and maximum value. If the convergence maximum is less than the error that you specified, and it will break; otherwise it will continue, and you have final display again.

(Refer Slide Time: 08:38)



Lid driven cavity flow – Post processing

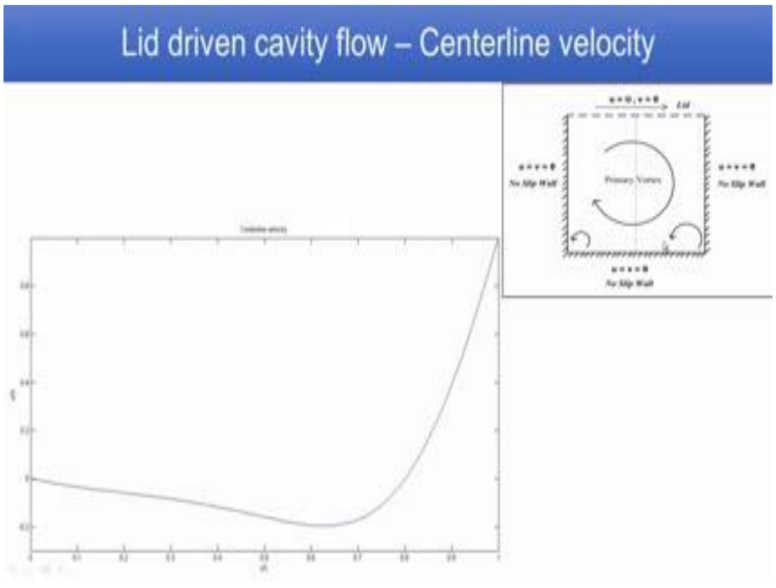
5. Following post processing are done using the results obtained after the convergence

- Contours of velocity
- Stream function contours
- Vorticity contours
- Centerline velocity
- Contours of pressure

Plotting the contours of velocity, pressure and centerline velocities are straightforward. Obtaining stream function and Vorticity values from the velocity values are explained here.

So, once you get solution then we have to do post processing, there are various methods available for post processing. You can look at u, v this are two velocity listed in this two-dimensional simulation. You can look at u and v at every node; you can also look at pressure at every node. You can also look at u and v in different form for examples stream function vorticity contour. And these are listed here. So, you can have a contours of velocity and we have the stream function contours, vorticity contours, centerline velocity you can have a velocity line plotted at any specific x and y grid line then you can have a contours of pressure etcetera. So, plotting the contours of velocity pressure and centerline velocities are straightforward, obtaining the stream function and vorticity values from the velocity values are explained in the subsequent slides.

(Refer Slide Time: 09:46)



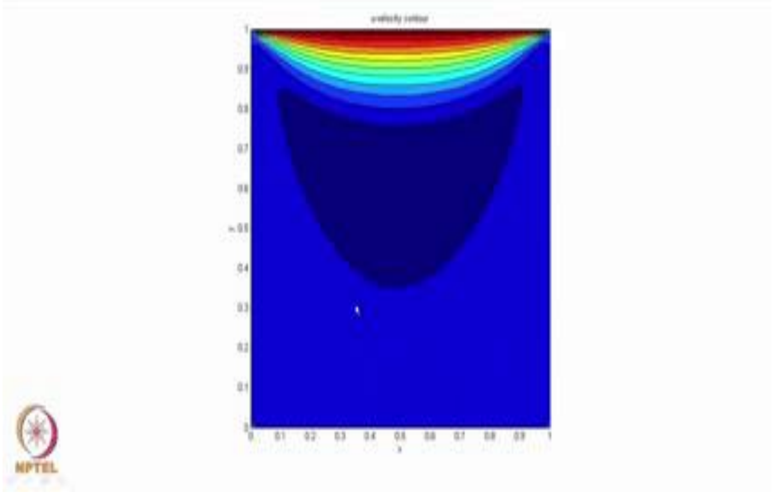
Lid driven cavity flow – Centerline velocity

So, this is the example problem, we had that is flow in a lid driven cavity. And this particular slide, we will try to see what is known as the center line velocity. So, this line that is shown here corresponds to x at half of this length. Please recall, we define the coordinate along this horizontal line is x coordinate line, and along this vertical line is y coordinate line. So, at the center of this face, you have a line mark and we are going to see velocity plotted along this line, and that is what shown here. And we mentioned one of beginning lectures; we have what is known as the non-dimensionalization. So, u that is the local velocity is non-dimensionalized with the driving velocity; the driving velocity is the velocity scale; and in this problem, we have taken driving velocity to be 1.

If you non-dimensionalized any velocity by driving velocity then you get u by u and that is what is plotted on the y-axis, and the length of the cavity in this case it is square and that is taken as l and that is used non-dimensionalized distance. So, we are going to plot along this line that is marked, and this is in y direction that is non-dimensionalized with the length. So, you have y over l going from 0 to 1. So, zero is on this side and one is on the other side. We mentioned that we have boundary condition that is u equal to 1. So, at y by l equal to 1, you have the maximum velocity, because it is non-dimensionalized, you have u by u will be 1. And on this side that is on the bottom, you have the wall and you have the velocity boundary condition, no slip condition u is equal to zero. So, you have u equal to 0 at this point, and it goes down to some negative value at particular value of y by l then it starts showing positive value that is the interpretation of this line. It is also possible to plot velocity variation along any particular location both horizontally as well as vertically.

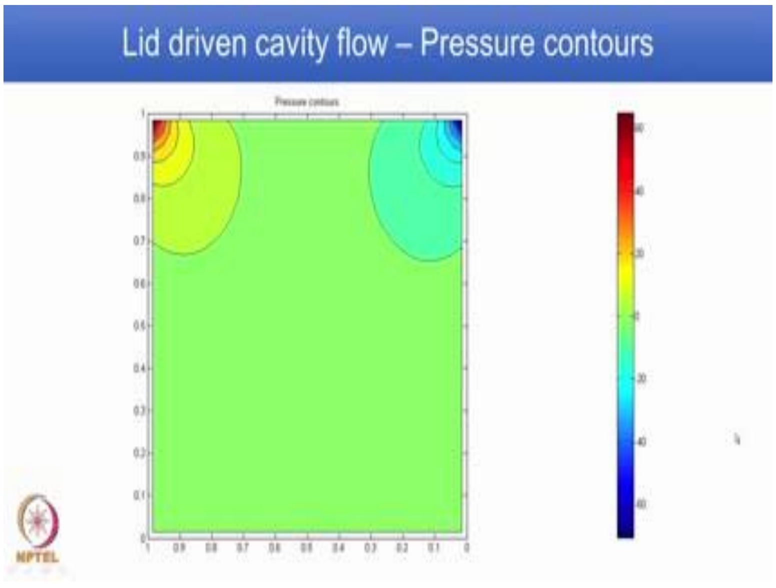
(Refer Slide Time: 12:13)

Lid driven cavity flow – u velocity contours



And you can also look at the velocity in the form of contour. So, what is shown as u velocity contour, so this gives you variation of velocity for the entire domain in the whole. So, this red color corresponds to maximum velocity and each line corresponds to one particular value. So, along this contour line the velocity values remains the same and blue color corresponds to very low velocity, and that is the interpretation of this contour plot.

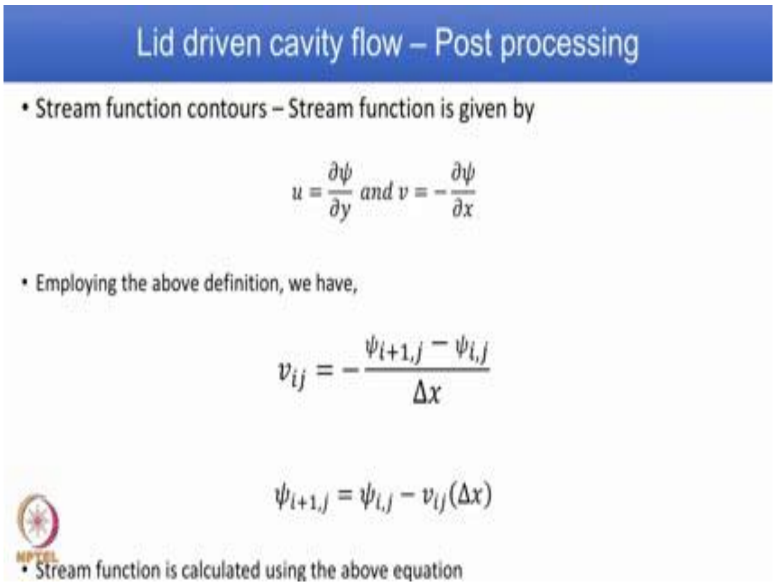
(Refer Slide Time: 12:54)



Lid driven cavity flow – Pressure contours

You can also look at pressure in the form of contour and that is what shown here again for the entire domain and you plot you plot contour along with along with legend and that is what shown here in the form of color bar. So, you can interpret as red color corresponds to value 60 and blue color corresponds to value minus 60, and this picture tells us the flow symmetric and that is why you get the symmetric pressure contour.

(Refer Slide Time: 13:28)



Lid driven cavity flow – Post processing

- Stream function contours – Stream function is given by

$$u = \frac{\partial \psi}{\partial y} \text{ and } v = -\frac{\partial \psi}{\partial x}$$

- Employing the above definition, we have,

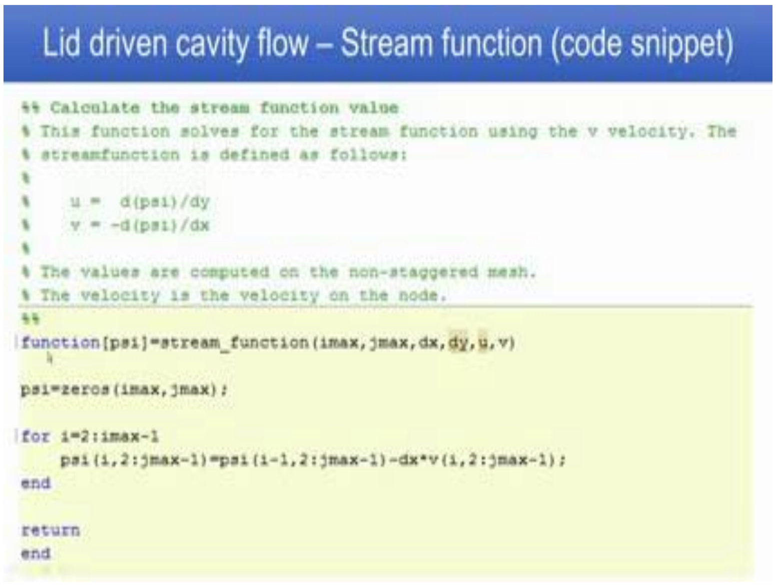
$$v_{ij} = -\frac{\psi_{i+1,j} - \psi_{i,j}}{\Delta x}$$

$$\psi_{i+1,j} = \psi_{i,j} - v_{ij}(\Delta x)$$

- Stream function is calculated using the above equation

Next the stream function contour, we know the definition of the stream function relating to velocity u equal to dou psi by dou y and v equal to minus dou psi by dou x employing the above definition, so to get stream function relating to velocity. So, now, we can take any velocity, for now we take v velocity, so v i comma j equal to minus psi i plus 1 comma j minus psi i comma j by delta x. Please remember stream lines are contours of stream func constant stream function and what we are done here is a discretized form of this particular equation. So, you define any particular line as a reference psi given the value of zero then you can go on integrate based on this velocity and this delta x to get psi at i plus 1. So, psi at i plus 1 comma j equal to psi at i comma j minus v i j into delta x stream function is calculated using the above relation, and you have the corresponding display of the code shown here.

(Refer Slide Time: 14:51)

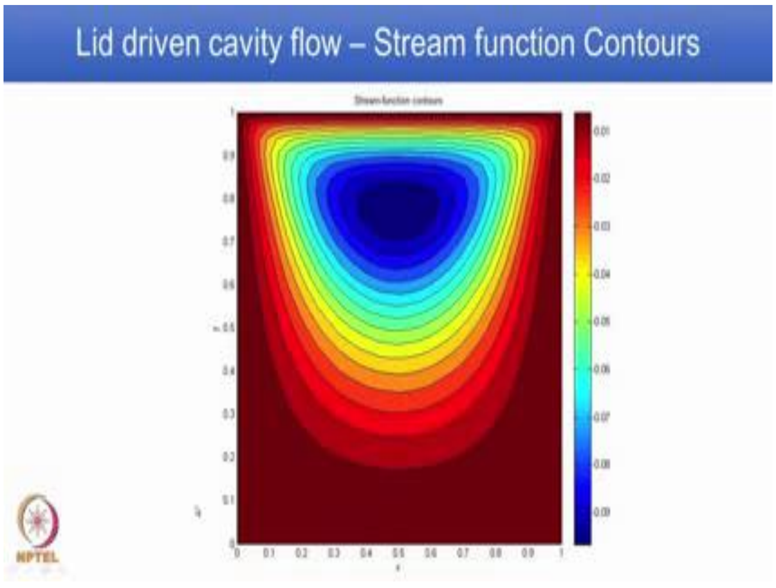


Lid driven cavity flow – Stream function (code snippet)

```
%% Calculate the stream function value
% This function solves for the stream function using the v velocity. The
% streamfunction is defined as follows:
%
%    u =  d(psi)/dy
%    v = -d(psi)/dx
%
% The values are computed on the non-staggered mesh.
% The velocity is the velocity on the node.
%%
function[psi]=stream_function(imax,jmax,dx,dy,u,v)

psi=zeros(imax,jmax);

for i=2:imax-1
    psi(i,2:jmax-1)=psi(i-1,2:jmax-1)-dx*v(i,2:jmax-1);
end

return
end
```

So, function psi equal to stream function i max comma j max comma dx comma dy comma u and v and psi are initialized beginning zeros i max comma j max. And for i 2 to i max minus 1 psi i comma 2 to j max minus 1 equal to psi i minus 1 comma 2 to j max minus one minus dx into v at i comma 2 to j max minus 1. This line is same as what we have seen in the previous slide, it calculate for all i location.

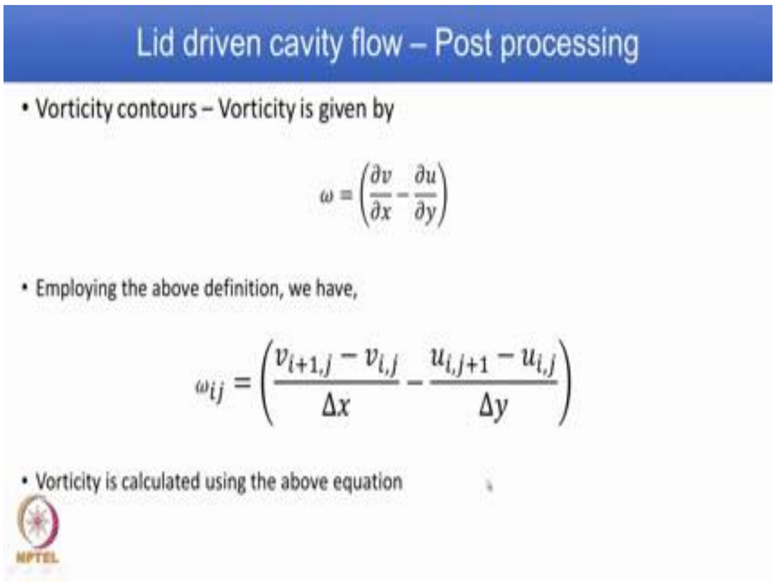
(Refer Slide Time: 15:42)



Lid driven cavity flow – Stream function Contours

And once you get the values of psi you can look at the contour as shown here, again there is a corresponding color definition in the form of the legend shown here. So, color map and corresponding color values indication is also given here. So, the square cavity, so y is going from one to as y is going from zero to one x is going from 0 to 1; and at the center, we have the very low value, and this point we have the very low value, and we have contours connecting all constant value of psi and this is the meaning of stream function contour.

(Refer Slide Time: 16:25)



Lid driven cavity flow – Post processing

- Vorticity contours – Vorticity is given by

$$\omega = \left(\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}\right)$$

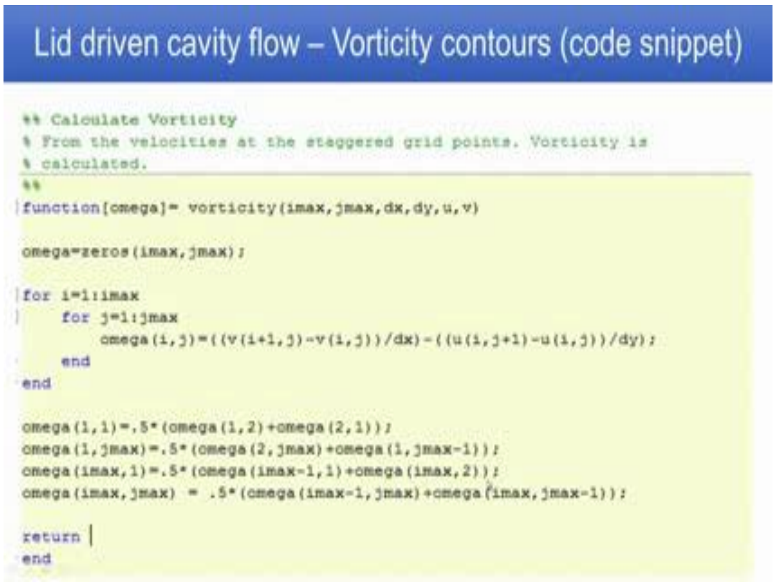
- Employing the above definition, we have,

$$\omega_{ij} = \left(\frac{v_{i+1,j} - v_{i,j}}{\Delta x} - \frac{u_{i,j+1} - u_{i,j}}{\Delta y}\right)$$

- Vorticity is calculated using the above equation

Next we have the vorticity contour. So, vorticity again defined based on the velocity component as shown here, omega is equal to dou v by dou x minus dou u by dou y. And you employing the above definition write the discrete form of the equation omega at i comma j equal to v at i plus 1comma j minus v at i comma j by delta x minus u i comma j plus 1 minus u at i comma j by delta y. So, using this you can calculate the vorticity.

(Refer Slide Time: 17:03)

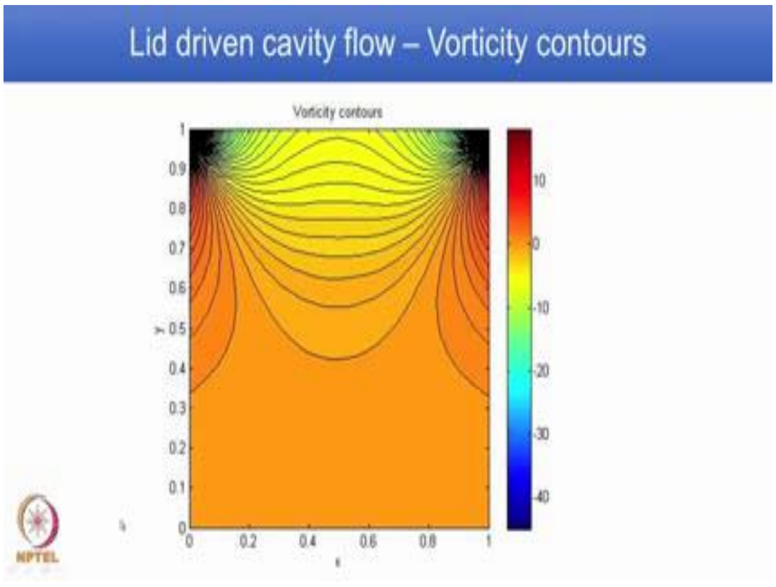


Lid driven cavity flow – Vorticity contours (code snippet)

```
%% Calculate Vorticity
% From the velocities at the staggered grid points. Vorticity is
% calculated.
%%
function[omega]= vorticity(imax,jmax,dx,dy,u,v)

omega=zeros(imax,jmax);

for i=1:imax
    for j=1:jmax
        omega(i,j)=((v(i+1,j)-v(i,j))/dx)-((u(i,j+1)-u(i,j))/dy);
    end
end

omega(1,1)=.5*(omega(1,2)+omega(2,1));
omega(1,jmax)=.5*(omega(2,jmax)+omega(1,jmax-1));
omega(imax,1)=.5*(omega(imax-1,1)+omega(imax,2));
omega(imax,jmax) = .5*(omega(imax-1,jmax)+omega(imax,jmax-1));

return
end
```

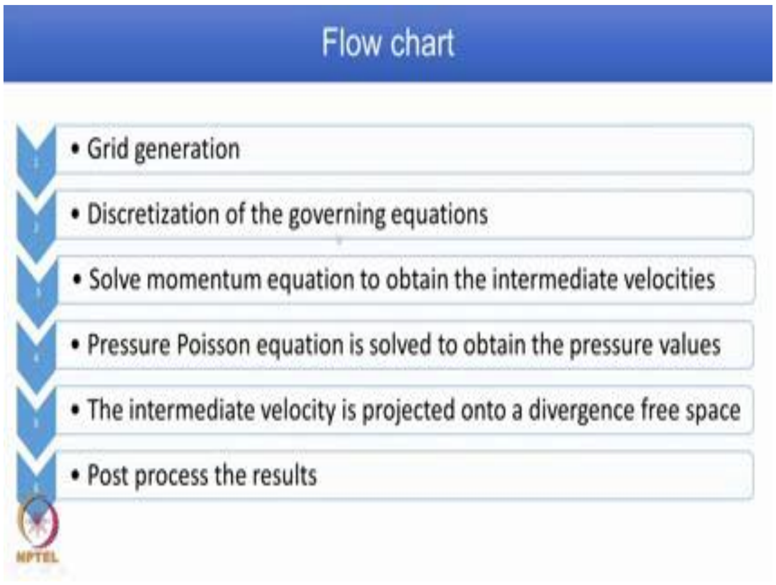
And it is shown in the form of the code. So, we have function omega that is the separate subroutine of function to calculate omega and you define corresponding arguments vorticity i max comma j max comma dx comma dy comma u comma v. So, you give velocity fields u v, you give delta x, delta y, and i max, j max to the function it will give you the vorticity. as usual it initialize the omega array, then you calculate omega as shown here. So, for i equal to 1 to i max, for j equal to 1 to j max, omega at i comma j equal to based on velocity component v and based on velocity component u. And this is again a code written corresponding to the mathematical definition we had in the previous slide. And you also defined a specific location so omega 1 comma 1 is defined as shown here omega 1 comma j max omega i max comma 1 and omega i max comma j max. So, these are values of omega at boundary condition locations.

(Refer Slide Time: 18:28)


Lid driven cavity flow – Vorticity contours

And once you get omega, again you will look at a flow through omega in the form of vorticity contour as shown here. Again you can have a interpretation, color and then corresponding values are given here. So, you can have idea where is the maximum omega, and where is the minimum omega.

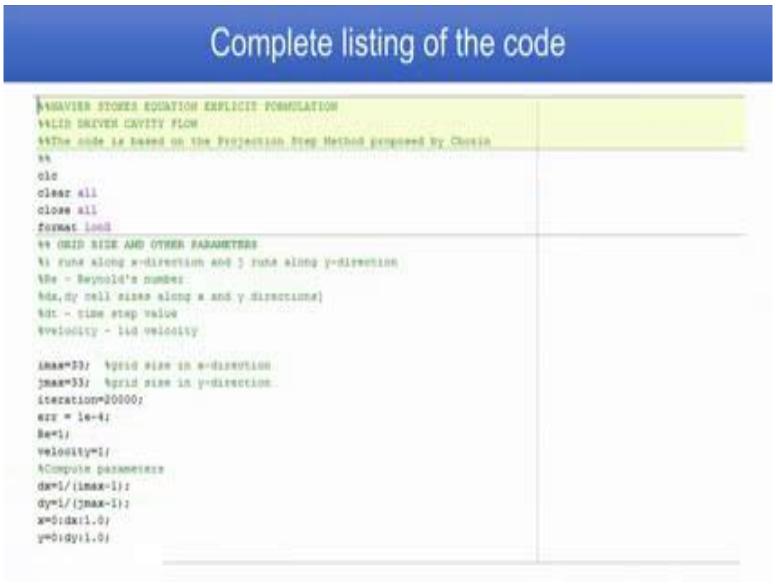
(Refer Slide Time: 18:52)


Flow chart

- Grid generation
- Discretization of the governing equations
- Solve momentum equation to obtain the intermediate velocities
- Pressure Poisson equation is solved to obtain the pressure values
- The intermediate velocity is projected onto a divergence free space
- Post process the results

The complete flow chart for this problem is given here. The first step we have the grid generation, second step discretization of the governing equation, third step to solve momentum equation to obtain the intermediate velocities, we have separately u and v momentum equation, you can solve them separately. Once you get u star and v star then we get pressure Poisson equation, and we solved pressure Poisson equation again separately. Then this intermediate velocity is obtained or corrected by this projection step on the divergence free space. Once you get correct velocities then they are post processed to get results and to look into the flow.

(Refer Slide Time: 19:37)



Complete listing of the code

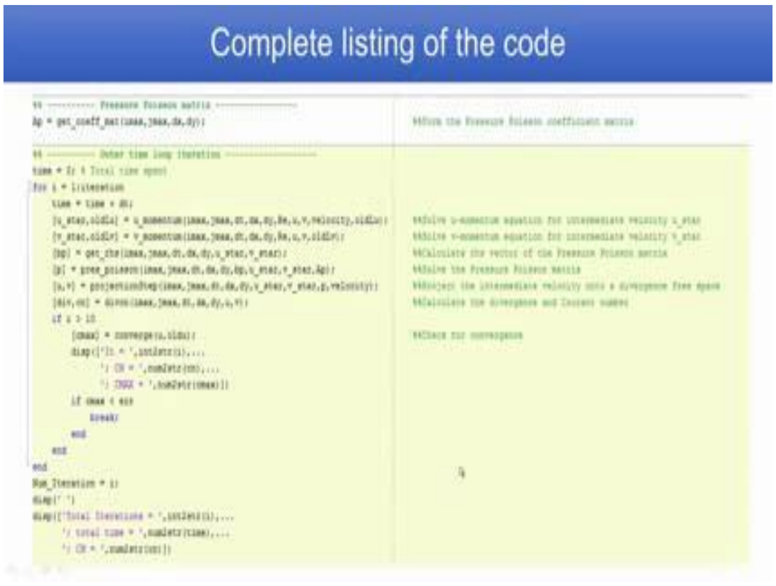
```
%%NAVIER STOKES EQUATION EXPLICIT FORMULATION
%%LID DRIVEN CAVITY FLOW
%%The code is based on the Projection Step Method proposed by Chorin
%%
clc
clear all
close all
format long
%% GRID SIZE AND OTHER PARAMETERS
%i runs along x-direction and j runs along y-direction
%Re - Reynold's number
%dx,dy cell sizes along x and y directions)
%dt - time step value
%velocity - lid velocity

imax=33;  %grid size in x-direction
jmax=33;  %grid size in y-direction
iteration=20000;
err = 1e-4;
Re=1;
velocity=1;
%Compute parameters
dx=1/(imax-1);
dy=1/(jmax-1);
x=0:dx:1.0;
y=0:dy:1.0;
```

Complete listing of the code is given. We have seen the code part-by-part now. We will see the entire code Navier-Stokes equation, explicit formulation, lid driven cavity flow the code is based on projection step method proposed by Chorin. So, we have all this initial lines then you define grid size on another parameters i runs along x direction, j runs along y direction Reynolds number is defined and that is used symbol Re; dx comma dy are cell sizes along x and y direction; dt is a time step value, velocity is the lid velocity that is the boundary condition on the top wall. So these are commands written.

Again we have explained in beginning grid, we are using only for that to get good results we are actually change the grid to 33 that is why you see here i max 33 grid size in x direction; similarly j max equal to 33 grid size in y direction. So, we set the limit of iteration to 20000, and the convergence criteria that is error in the divergence as 1 10 to the power of minus 4, Reynolds number as 1, and velocity boundary condition on the top wall as 1. So, once you define number of grid lines then you calculate dx and dy and that is what shown here dx equal to 1 upon i max minus 1, and dy equal 1 upon j max minus

1. So, x is going from 0 by delta x value upto the value of 1.0. Similarly y is going from 0 to 1 with the step value of dy, dt equal to 0.002

(Refer Slide Time: 21:44)

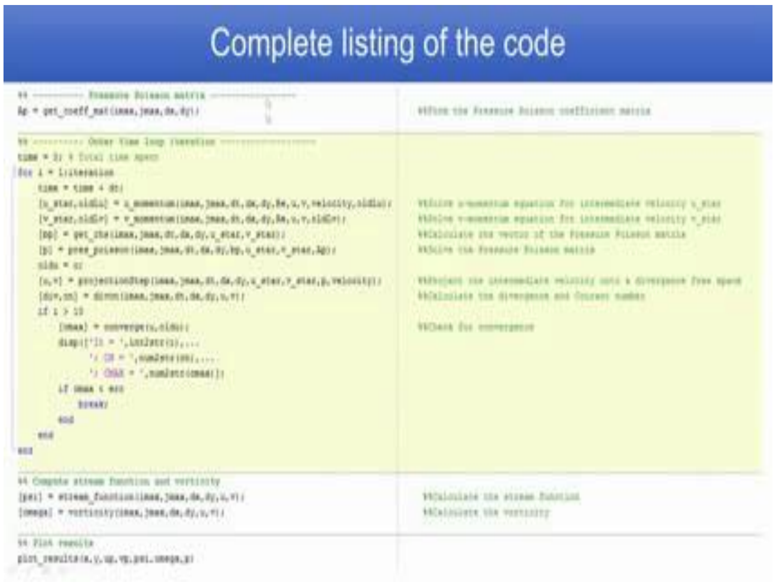


Complete listing of the code

```
%% ---------- Pressure Poisson matrix ----------------
Ap = get_coeff_mat(imax,jmax,dx,dy);                                   %%Form the Pressure Poisson coefficient matrix

%% -------- Outer time loop iteration -----------------
time = 0; % Total time spent
for i = 1:iteration
    time = time + dt;
    [u_star,oldu] = u_momentum(imax,jmax,dt,dx,dy,Re,u,v,velocity,oldu);   %%Solve u-momentum equation for intermediate velocity u_star
    [v_star,oldv] = v_momentum(imax,jmax,dt,dx,dy,Re,u,v,oldv);            %%Solve v-momentum equation for intermediate velocity v_star
    [bp] = get_rhs(imax,jmax,dt,dx,dy,u_star,v_star);                      %%Calculate the vector of the Pressure Poisson matrix
    [p] = pres_poisson(imax,jmax,dt,dx,dy,bp,u_star,v_star,Ap);            %%Solve the Pressure Poisson matrix
    [u,v] = projectionStep(imax,jmax,dt,dx,dy,u_star,v_star,p,velocity);  %%Project the intermediate velocity onto a divergence free space
    [div,cn] = diver(imax,jmax,dt,dx,dy,u,v);                              %%Calculate the divergence and Courant number
    if i > 10
        [cmax] = converge(u,oldu);                                         %%Check for convergence
        disp(['It = ',int2str(i),...
              '; CN = ',num2str(cn),...
              '; INDX = ',num2str(cmax)])
        if cmax < eps
            break;
        end
    end
end
Num_Iteration = i;
disp(' ')
disp(['Total Iterations = ',int2str(i),...
     '; total time = ',num2str(time),...
     '; CN = ',num2str(cn)])
```

And we have the complete listing of the code; we have already seen part-by-part. So this is the first line is for the pressure Poisson coefficient matrix then you have the outer loop time spent then iteration starts. So, we calculate separately u momentum, v momentum and this line corresponds to right side vector of the pressure Poisson matrix then you solve the pressure Poisson equation, you get solution for pressure, then do the projection step to get the correct u and v velocity check for divergence, and also check Courant number then you check whether it is between number of iteration and between Courant number then you break or you continue. Then finally you can have the list of display the number of iteration total iteration Courant number etcetera.

(Refer Slide Time: 22:49)

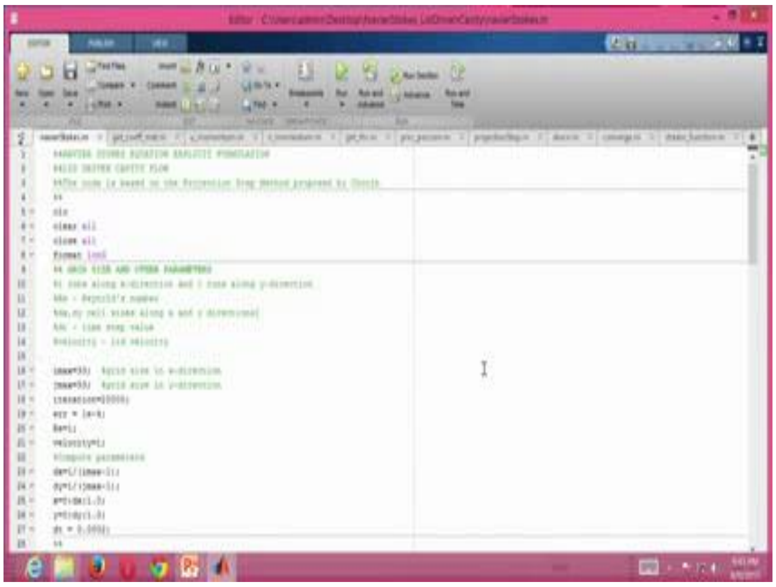


Complete listing of the code

```
%% ---------- Pressure Poisson matrix ----------
Ap = get_coeff_mat(imax,jmax,dx,dy);                              %%Form the Pressure Poisson coefficient matrix

%% ---------- Outer Time loop iteration ------------------
time = 0; % Total time spent
for i = 1:iteration
    time = time + dt;
    [u_star,oldLu] = u_momentum(imax,jmax,dt,dx,dy,Re,u,v,velocity,oldLu);   %%Solve u-momentum equation for intermediate velocity u_star
    [v_star,oldLv] = v_momentum(imax,jmax,dt,dx,dy,Re,u,v,oldLv);            %%Solve v-momentum equation for intermediate velocity v_star
    [bp] = get_rhs(imax,jmax,dt,dx,dy,u_star,v_star);                        %%Calculate the vector of the Pressure Poisson matrix
    [p] = pres_poisson(imax,jmax,dt,dx,dy,bp,u_star,v_star,Ap);              %%Solve the Pressure Poisson matrix
    oldu = u;
    [u,v] = projectionStep(imax,jmax,dt,dx,dy,u_star,v_star,p,velocity);     %%Project the intermediate velocity onto a divergence free space
    [div,cn] = divcn(imax,jmax,dt,dx,dy,u,v);                                %%Calculate the divergence and Courant number
    if i > 10
        [cmax] = converge(u,oldu);                                           %%Check for convergence
        disp(['It = ',int2str(i),...
              '; CN = ',num2str(cn),...
              '; CMAX = ',num2str(cmax)]);
        if cmax < err
            break;
        end
    end
end

%% Compute stream function and vorticity
[psi] = stream_function(imax,jmax,dx,dy,u,v);                     %%Calculate the stream function
[omega] = vorticity(imax,jmax,dx,dy,u,v);                         %%Calculate the vorticity

%% Plot results
plot_results(x,y,up,vp,psi,omega,p)
```

And this same now we have included the post processing lines also, we have stream function and we have omega, and finally, we can also plot results this is the complete working code using Matlab.

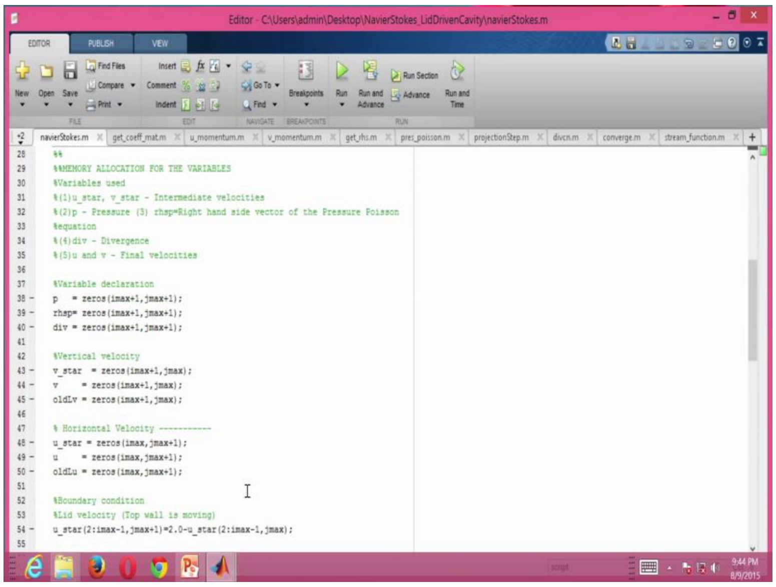
(Refer Slide Time: 23:04)



```
%NAVIER STOKES EQUATION EXPLICIT FORMULATION
%LID DRIVEN CAVITY FLOW
%%The code is based on the Projection Step Method proposed by Chorin
%%
clc
clear all
close all
format long
%% GRID SIZE AND OTHER PARAMETERS
%i rows along x-direction and j rows along y-direction
%Re - Reynold's number
%dx,dy cell sizes along x and y directions)
%dt - time step value
%velocity - lid velocity

imax=33;  %grid size in x-direction
jmax=33;  %grid size in y-direction
iteration=10000;
err = 1e-6;
Re=1;
velocity=1;
%Compute parameters
dx=1/(imax-1);
dy=1/(jmax-1);
x=0:dx:1.0;
y=0:dy:1.0;
dt = 0.0001;
%%
```

We have explained the algorithm and corresponding code part in detail. Now we will put an entire algorithm, and then we will explain how the code is actually running, and also discuss the results obtained. What is shown here is a Matlab code, in Matlab platform itself. Now let us start seeing looking at the lines Navier-Stokes equation, explicit formulation, lid driven cavity flow and this are certain commands. Then we have clear screen, first one we start talking about grid we explain initially with 4 by 4, the actual running of the code we run with 33 by 33. So, i max is 33, and j max is 33. The number of iteration allowed is about 20000, and for convergence criteria we set the limit 10 to the power of minus 4, Reynolds number is 1, and lid driven velocity condition is 1. Then we define grid based on this, so dx equal to one upon i max minus 1, and dy equal to 1 upon j max minus one and dt is also set a 0.0002, then we do the memory allocation of all the variables.

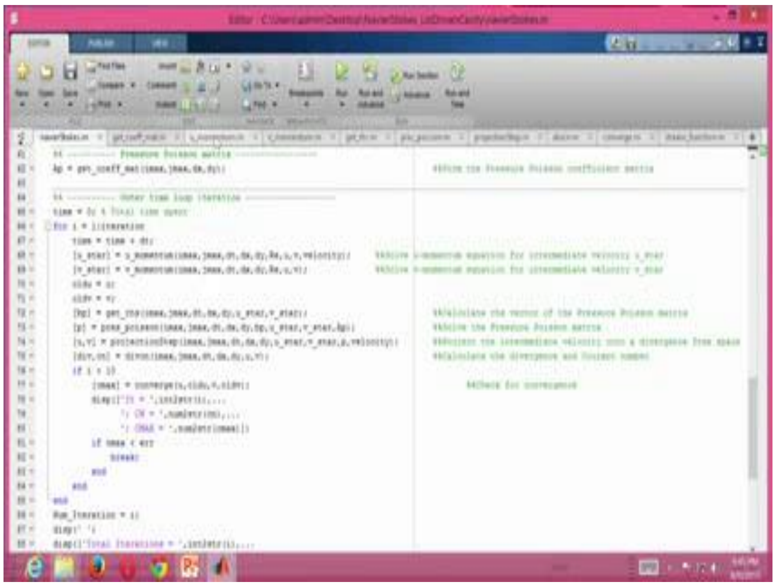
(Refer Slide Time: 24:26)



```
%%
%%MEMORY ALLOCATION FOR THE VARIABLES
%Variables used
%(1)u_star, v_star - Intermediate velocities
%(2)p - Pressure (3) rhsp=Right hand side vector of the Pressure Poisson
%equation
%(4)div - Divergence
%(5)u and v - Final velocities

%Variable declaration
p   = zeros(imax+1,jmax+1);
rhsp= zeros(imax+1,jmax+1);
div = zeros(imax+1,jmax+1);

%Vertical velocity
v_star  = zeros(imax+1,jmax);
v      = zeros(imax+1,jmax);
oldLv = zeros(imax+1,jmax);

% Horizontal Velocity -----------
u_star = zeros(imax,jmax+1);
u      = zeros(imax,jmax+1);
oldLu = zeros(imax,jmax+1);

%Boundary condition
%Lid velocity (Top wall is moving)
u_star(2:imax-1,jmax+1)=2.0-u_star(2:imax-1,jmax);

```

So, pressure then rhs side of the pressure equation and divergence calculation vertical velocity; in that we have predicted velocity v star and actual velocity; similarly for horizontal velocity u star u then boundary condition. U star we define boundary condition for u star as shown here. Then velocity and collocated mesh this is required we solved equations or we stored variables at staggered location. Once we obtained solution then we need to do post processing. To enable post processing we convert the variable from staggered grid to collocated grid arrangement and that is done using this collocated definition of the variables. Then we solve pressure Poisson equation.

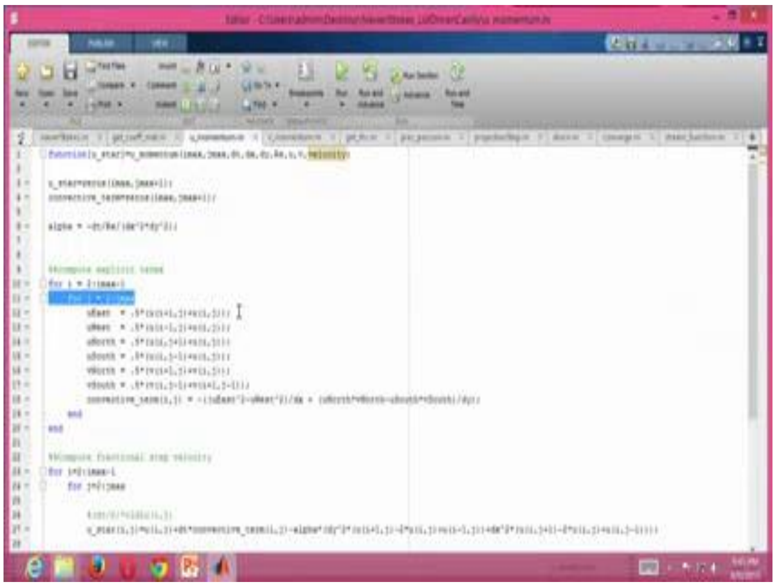
(Refer Slide Time: 25:24)



```
%% ---------- Pressure Poisson matrix ----------
Ap = get_coeff_mat(imax,jmax,dx,dy);                                   %%Form the Pressure Poisson coefficient matrix

%% ---------- Outer time loop iteration ----------
time = 0; % Total time spent
for i = 1:iteration
    time = time + dt;
    [u_star] = u_momentum(imax,jmax,dt,dx,dy,Re,u,v,velocity);     %%Solve u-momentum equation for intermediate velocity u_star
    [v_star] = v_momentum(imax,jmax,dt,dx,dy,Re,u,v);              %%Solve v-momentum equation for intermediate velocity v_star
    uold = u;
    vold = v;
    [bp] = get_rhs(imax,jmax,dt,dx,dy,u_star,v_star);                  %%Calculate the vector of the Pressure Poisson matrix
    [p] = pres_poisson(imax,jmax,dt,dx,dy,bp,u_star,v_star,Ap);        %%Solve the Pressure Poisson matrix
    [u,v] = projectionStep(imax,jmax,dt,dx,dy,u_star,v_star,p,velocity);  %%Project the intermediate velocity into a divergence free space
    [div,cn] = divcn(imax,jmax,dt,dx,dy,u,v);                          %%Calculate the divergence and Courant number
    if i > 10
        [cmax] = converge(u,uold,v,vold);                                  %%Check for convergence
        disp(['It = ',int2str(i),...
              '; CN = ',num2str(cn),...
              '; CMAX = ',num2str(cmax)])
        if cmax < err
            break;
        end
    end
end
Num_Iteration = i;
disp(' ');
disp(['Total Iterations = ',int2str(i),...
```

So, we have the u star u momentum

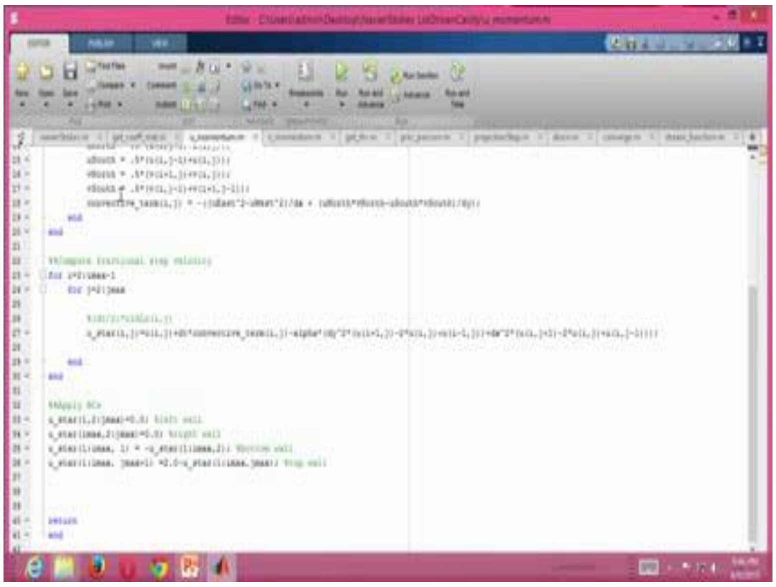
(Refer Slide Time: 25:28)



```
function[u_star]=u_momentum(imax,jmax,dt,dx,dy,Re,u,v,velocity)

u_star=zeros(imax,jmax+1);
convective_term=zeros(imax,jmax+1);

alpha = -dt/Re/(dx^2*dy^2);


%Compute explicit terms
for i = 2:imax-1
    for j = 2:jmax
        uEast  = .5*(u(i+1,j)+u(i,j));
        uWest  = .5*(u(i-1,j)+u(i,j));
        uNorth = .5*(u(i,j+1)+u(i,j));
        uSouth = .5*(u(i,j-1)+u(i,j));
        vNorth = .5*(v(i+1,j)+v(i,j));
        vSouth = .5*(v(i,j-1)+v(i+1,j-1));
        convective_term(i,j) = -((uEast^2-uWest^2)/dx + (uNorth*vNorth-uSouth*vSouth)/dy);
    end
end

%Compute fractional step velocity
for i=2:imax-1
    for j=2:jmax

        %(dt/2)*old(i,j)
        u_star(i,j)=u(i,j)+dt*convective_term(i,j)-alpha*(dy^2*(u(i+1,j)-2*u(i,j)+u(i-1,j))+dx^2*(u(i,j+1)-2*u(i,j)+u(i,j-1)));
```

Now comes the outer loop, so we have u star u momentum and that is given in detail in this particular function. We have seen all this in detail convective term again initialized; alpha is a common factor that is also defined. Then we calculate for convection term, u at different location and that what we shown here; u east, u west, u north, u south similarly v north and v south, finally you calculate convective term then you compute fractional step velocity including convection term as well as diffusion term as shown here.

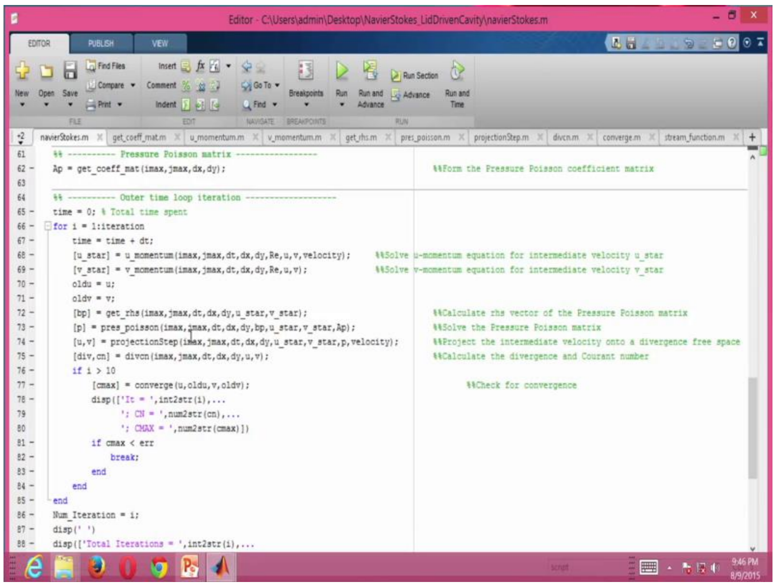
(Refer Slide Time: 26:01)



```
        uSouth = .5*(u(i,j-1)+u(i,j));
        vNorth = .5*(v(i+1,j)+v(i,j));
        vSouth = .5*(v(i,j-1)+v(i+1,j-1));
        convective_term(i,j) = -((uEast^2-uWest^2)/dx + (uNorth*vNorth-uSouth*vSouth)/dy);
    end
end

%%Compute fractional step velocity
for i=2:imax-1
    for j=2:jmax

        [illegible]
        u_star(i,j)=u(i,j)+dt*convective_term(i,j)-alpha*(dy^2*(u(i+1,j)-2*u(i,j)+u(i-1,j))+dx^2*(u(i,j+1)-2*u(i,j)+u(i,j-1)));

    end
end

%%Apply BCs
u_star(1,2:jmax)=0.0; %Left wall
u_star(imax,2:jmax)=0.0; %Right wall
u_star(1:imax, 1) = -u_star(1:imax,2); %Bottom wall
u_star(1:imax, jmax+1) =2.0-u_star(1:imax,jmax); %Top wall



return
end
```

And that will be u star because we have neglected pressure in the first step. Once you calculate u star, you need to apply boundary condition and corresponding lines are shown here.

(Refer Slide Time: 26:19)

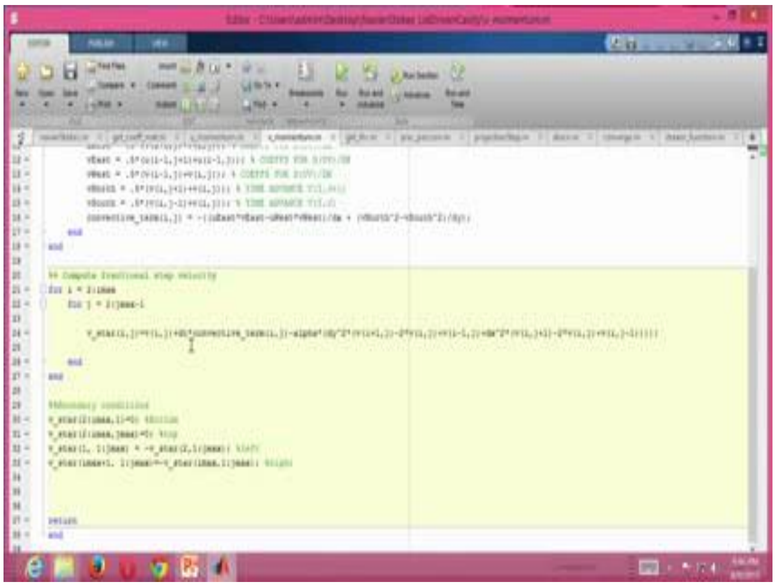


Editor - C:\Users\admin\Desktop\NavierStokes_LidDrivenCavity\navierStokes.m

```
%% ---------- Pressure Poisson matrix -----------------
Ap = get_coeff_mat(imax,jmax,dx,dy);                                    %%Form the Pressure Poisson coefficient matrix

%% ---------- Outer time loop iteration -------------------
time = 0; % Total time spent
for i = 1:iteration
    time = time + dt;
    [u_star] = u_momentum(imax,jmax,dt,dx,dy,Re,u,v,velocity);     %%Solve u-momentum equation for intermediate velocity u_star
    [v_star] = v_momentum(imax,jmax,dt,dx,dy,Re,u,v);              %%Solve v-momentum equation for intermediate velocity v_star
    oldu = u;
    oldv = v;
    [bp] = get_rhs(imax,jmax,dt,dx,dy,u_star,v_star);                    %%Calculate rhs vector of the Pressure Poisson matrix
    [p] = pres_poisson(imax,jmax,dt,dx,dy,bp,u_star,v_star,Ap);          %%Solve the Pressure Poisson matrix
    [u,v] = projectionStep(imax,jmax,dt,dx,dy,u_star,v_star,p,velocity); %%Project the intermediate velocity onto a divergence free space
    [div,cn] = divcn(imax,jmax,dt,dx,dy,u,v);                            %%Calculate the divergence and Courant number
    if i > 10
        [cmax] = converge(u,oldu,v,oldv);                                      %%Check for convergence
        disp(['It = ',int2str(i),...
              '; CN = ',num2str(cn),...
              '; CMAX = ',num2str(cmax)])
        if cmax < err
            break;
        end
    end
end
Num_Iteration = i;
disp(' ')
disp(['Total Iterations = ',int2str(i),...
```

And come back to main code, so we have explained u momentum, similar steps are there in v momentum function also.

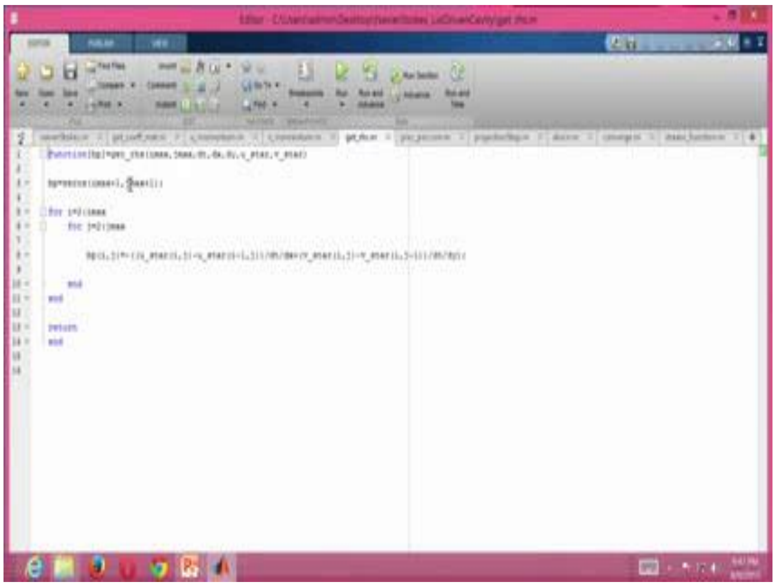
(Refer Slide Time: 26:31)



```
        uEast = .5*(u(i-1,j+1)+u(i-1,j)); % COEFFS FOR D(UV)/DX
        uWest = .5*(v(i-1,j)+v(i,j)); % COEFFS FOR D(UV)/DX
        vNorth = .5*(v(i,j+1)+v(i,j)); % TIME ADVANCE V(I,J+1)
        vSouth = .5*(v(i,j-1)+v(i,j)); % TIME ADVANCE V(I,J)
        convection_term(i,j) = -((uEast*vEast-uWest*vWest)/dx + (vNorth^2-vSouth^2)/dy);
    end
end

%% Compute fractional step velocity
for i = 2:imax
    for j = 2:jmax-1

        v_star(i,j)=v(i,j)+dt*(convection_term(i,j)-alpha*(dy^2*(v(i+1,j)-2*v(i,j)+v(i-1,j))+dx^2*(v(i,j+1)-2*v(i,j)+v(i,j-1))));

    end
end

%Boundary condition
v_star(2:imax,1)=0; %bottom
v_star(2:imax,jmax)=0; %top
v_star(1, 1:jmax) = -v_star(2,1:jmax); %left
v_star(imax+1, 1:jmax)=-v_star(imax,1:jmax); %right



return
end
```

That is again displayed here separately v star equal to convection term and diffusion term. Once you find v star apply the boundary condition as shown here. Come back to main, so we have u momentum, v momentum explained then pressure Poisson equation set the right side of the pressure Poisson equation first that is in the module get R.H.S; again go to the particular module.

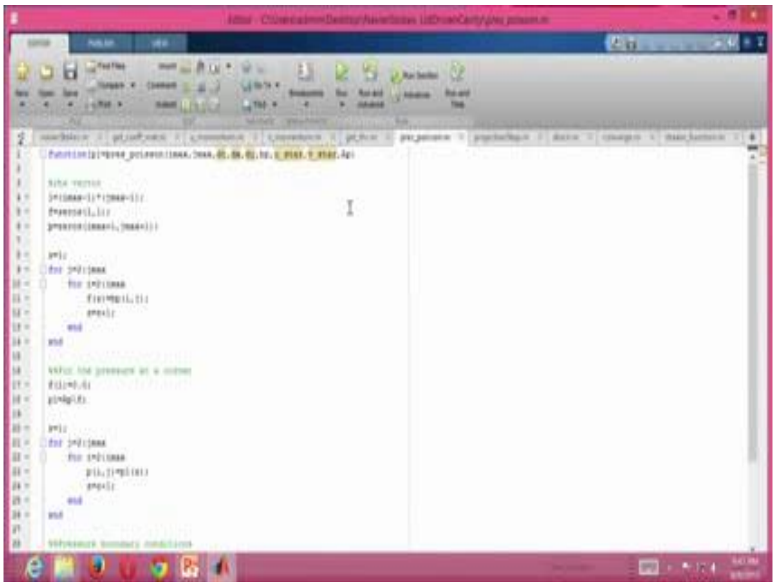
(Refer Slide Time: 26:58)



```
function[bp]=get_rhs(imax,jmax,dt,dx,dy,u_star,v_star)

bp=zeros(imax+1,jmax+1);

for i=2:imax
    for j=2:jmax

        bp(i,j)=-((u_star(i,j)-u_star(i-1,j))/dt/dx+(v_star(i,j)-v_star(i,j-1))/dt/dy);

    end
end

return
end
```

And it is show here, initialized and calculate all the coefficient values of the right side matrix. Go to the main then we solved the pressure Poisson equation press; Poisson is the corresponding module go to the press Poisson dot m.

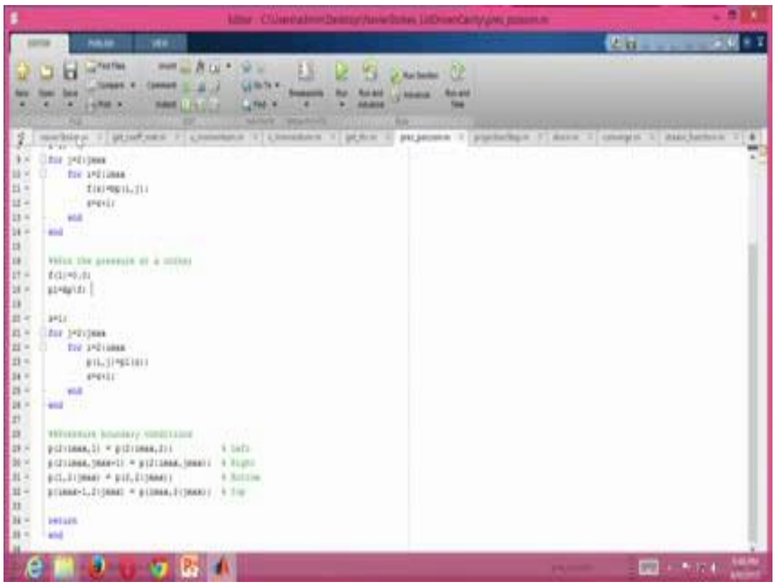
(Refer Slide Time: 27:15)



```
function[p]=press_poisson(imax,jmax,dt,dx,dy,tp,u_star,v_star,Ap)

%rhs vector
l=(imax-1)*(jmax-1);
f=zeros(l,1);
p=zeros(imax+1,jmax+1);

s=1;
for j=2:jmax
    for i=2:imax
        f(s)=tp(i,j);
        s=s+1;
    end
end

%%fix the pressure at a corner
f(1)=0.0;
pl=Ap\f;

s=1;
for j=2:jmax
    for i=2:imax
        p(i,j)=pl(s);
        s=s+1;
    end
end

%%Pressure boundary conditions
```

So, we have initialized pressure then f R.H.S vector of the pressure poisson equation.

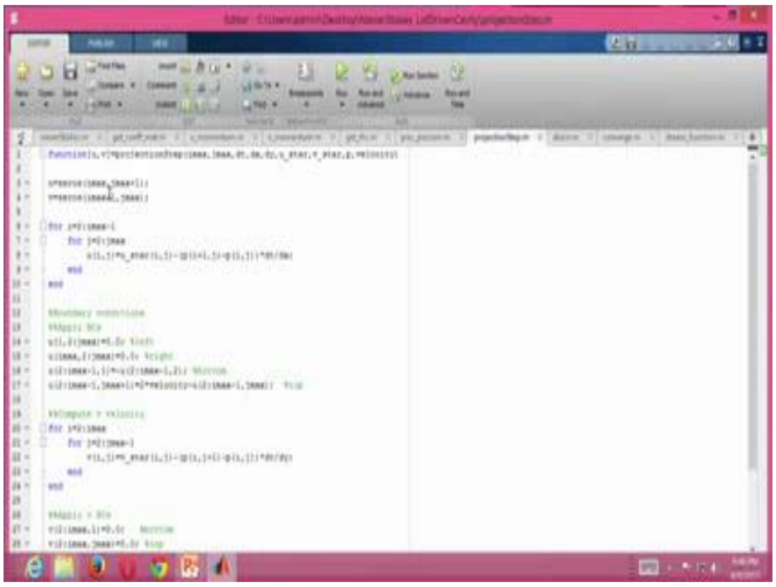
(Refer Slide Time: 27:28)



```
for j=2:jmax
    for i=2:imax
        f(s)=tp(i,j);
        s=s+1;
    end
end

%%fix the pressure at a corner
f(1)=0.0;
pl=Ap\f;

s=1;
for j=2:jmax
    for i=2:imax
        p(i,j)=pl(s);
        s=s+1;
    end
end

%%Pressure boundary conditions
p(2:imax,1) = p(2:imax,2);          % Left
p(2:imax,jmax+1) = p(2:imax,jmax);  % Right
p(1,2:jmax) = p(2,2:jmax);          % Bottom
p(imax+1,2:jmax) = p(imax,2:jmax);  % Top

return
end
```

Then we define nu of the pressure because, we are interested only the pressure difference and one point is pin to 0 and all over values are referred with respect to a pinned point and that is what this particular line. Then we solve the matrix and this is inbuilt matrix inversion procedure, so pressure is obtained. Once you obtained pressure then the applied boundary condition for left side, right side, bottom wall and top wall. Again go to the main. So, we are done now pressure Poisson equation, we got pressure, then this pressure is projected to get the corrected velocity, and that is what done in the module projection step. So, we go to the projection step module.

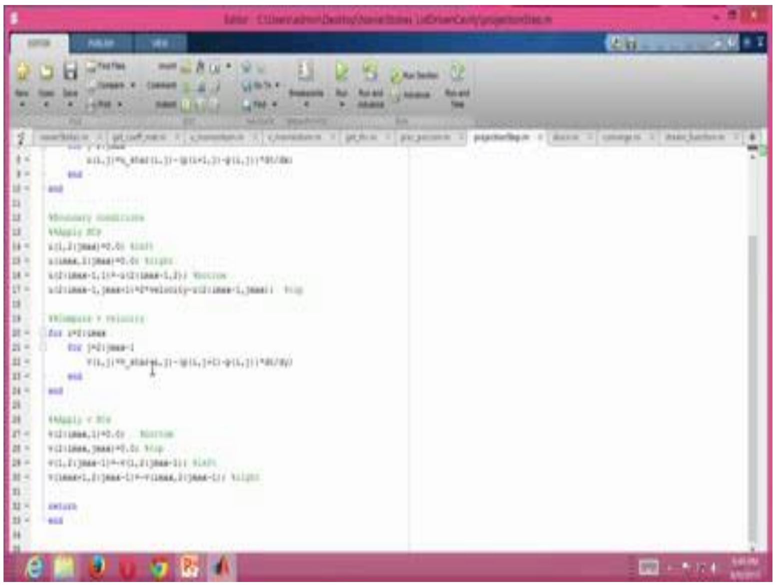
(Refer Slide Time: 28:16)



```
function[u,v]=projectionStep(imax,jmax,dt,dx,dy,u_star,v_star,p,velocity)

u=zeros(imax,jmax+1);
v=zeros(imax+1,jmax);

for i=2:imax-1
    for j=2:jmax
        u(i,j)=u_star(i,j)-(p(i+1,j)-p(i,j))*dt/dx;
    end
end

%boundary conditions
%Apply BCs
u(1,2:jmax)=0.0; %left
u(imax,2:jmax)=0.0; %right
u(2:imax-1,1)=-u(2:imax-1,2); %bottom
u(2:imax-1,jmax+1)=2*velocity-u(2:imax-1,jmax); %top

%Compute v velocity
for i=2:imax
    for j=2:jmax-1
        v(i,j)=v_star(i,j)-(p(i,j+1)-p(i,j))*dt/dy;
    end
end

%Apply v BCs
v(2:imax,1)=0.0; %bottom
v(2:imax,jmax)=0.0; %top
```

So, again values are initialized zeros for u and v then you correct the velocity. So, u star was initially obtained. So, pressure is obtained from pressure Poisson equation using this two, you get the correct velocity expression as shown. And once you get u then we need to ensure boundary condition that is also done using these lines for left boundary, right boundary, bottom wall and top wall.

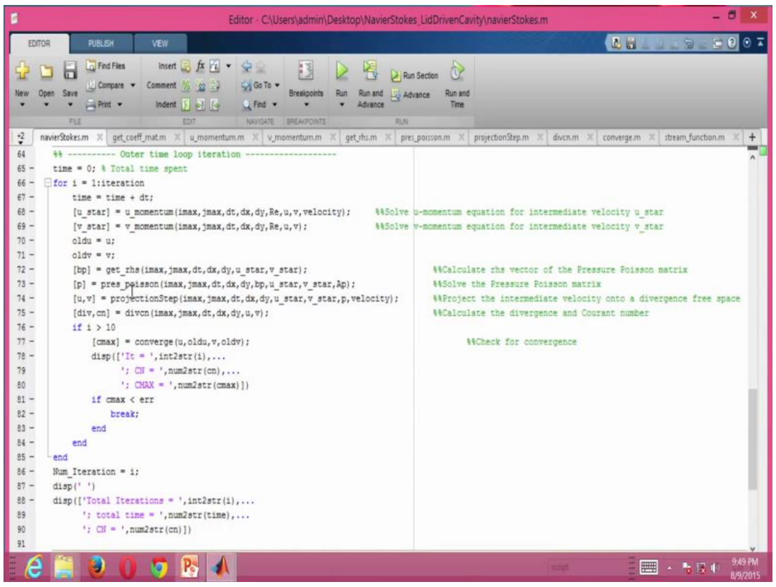
(Refer Slide Time: 28:49)



```
            u(i,j)=u_star(i,j)-(p(i+1,j)-p(i,j))*dt/dx;
        end
end

%Boundary conditions
%Apply BCs
u(1,2:jmax)=0.0; %left
u(imax,2:jmax)=0.0; %right
u(2:imax-1,1)=-u(2:imax-1,2); %bottom
u(2:imax-1,jmax+1)=2*velocity-u(2:imax-1,jmax); %top

%Compute v velocity
for i=2:imax
    for j=2:jmax-1
        v(i,j)=v_star(i,j)-(p(i,j+1)-p(i,j))*dt/dy;
    end
end

%Apply v BCs
v(2:imax,1)=0.0; %bottom
v(2:imax,jmax)=0.0; %top
v(1,2:jmax-1)=-v(2,2:jmax-1); %left
v(imax+1,2:jmax-1)=-v(imax,2:jmax-1); %right

return
end
```

Similarly for v equation, so v star is used and pressure - corrected pressure is used. So, you get correct velocity. Once you get correct second component of velocity v apply boundary condition for v and that is done by these lines go back to the main.

(Refer Slide Time: 29:07)

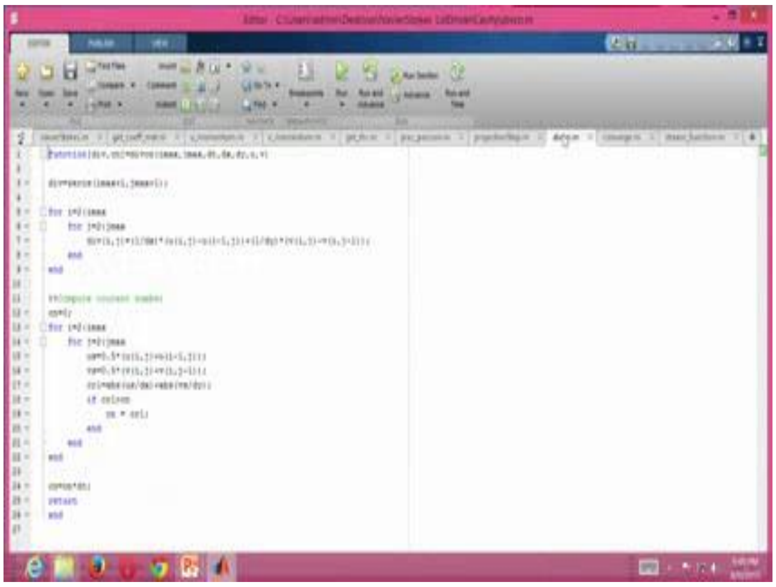


Editor - C:\Users\admin\Desktop\NavierStokes_LidDrivenCavity\navierStokes.m

```
%% ---------- Outer time loop iteration -------------------
time = 0; % Total time spent
for i = 1:iteration
    time = time + dt;
    [u_star] = u_momentum(imax,jmax,dt,dx,dy,Re,u,v,velocity);     %%Solve u-momentum equation for intermediate velocity u_star
    [v_star] = v_momentum(imax,jmax,dt,dx,dy,Re,u,v);              %%Solve v-momentum equation for intermediate velocity v_star
    oldu = u;
    oldv = v;
    [bp] = get_rhs(imax,jmax,dt,dx,dy,u_star,v_star);                 %%Calculate rhs vector of the Pressure Poisson matrix
    [p] = pres_poisson(imax,jmax,dt,dx,dy,bp,u_star,v_star,Ap);       %%Solve the Pressure Poisson matrix
    [u,v] = projectionStep(imax,jmax,dt,dx,dy,u_star,v_star,p,velocity);   %%Project the intermediate velocity onto a divergence free space
    [div,cn] = divcn(imax,jmax,dt,dx,dy,u,v);                         %%Calculate the divergence and Courant number
    if i > 10
        [cmax] = converge(u,oldu,v,oldv);                                    %%Check for convergence
        disp(['It = ',int2str(i),...
              '; CN = ',num2str(cn),...
              '; CMAX = ',num2str(cmax)])
        if cmax < err
            break;
        end
    end
end
Num_Iteration = i;
disp(' ')
disp(['Total Iterations = ',int2str(i),...
      '; total time = ',num2str(time),...
      '; CN = ',num2str(cn)])
```

So, we have done pressure Poisson projection step to get correct u and v velocity. Now we need to check whether divergence is satisfied. So, we have separate module So, divcn.

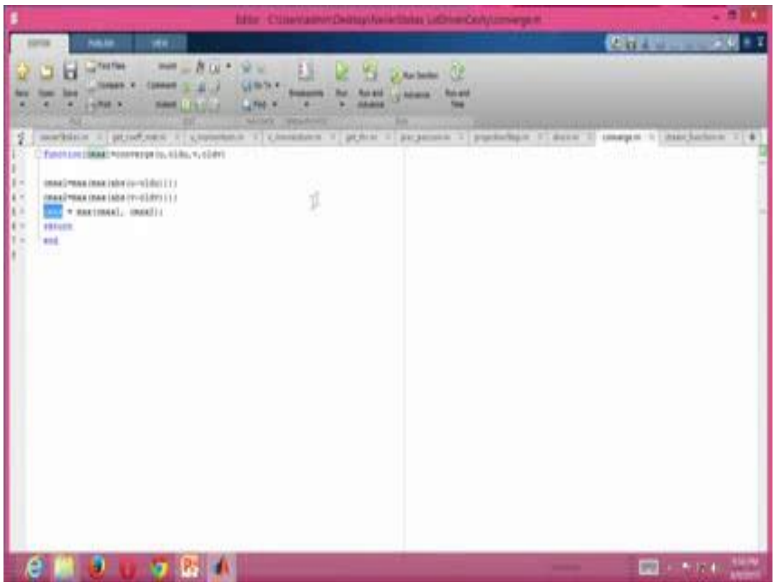
(Refer Slide Time: 29:21)



```
function[div,cn]=divcn(imax,jmax,dt,dx,dy,u,v)

div=zeros(imax+1,jmax+1);

for i=2:imax
    for j=2:jmax
        div(i,j)=(1/dx)*(u(i,j)-u(i-1,j))+(1/dy)*(v(i,j)-v(i,j-1));
    end
end

%%compute courant number
cn=0;
for i=2:imax
    for j=2:jmax
        ue=0.5*(u(i,j)+u(i-1,j));
        ve=0.5*(v(i,j)+v(i,j-1));
        cnl=abs(ue/dx)+abs(ve/dy);
        if cnl>cn
            cn = cnl;
        end
    end
end

cn=cn*dt;
return
end
```

Let us go to that module, so in divcn calculate the continuity error. So, this is dou u by dou x and dou v by dou y, then you calculate also the Courant number it is helpful. So, you calculate the Courant number based on this expression here. Come back to the main. So, we calculate divergence then we calculate Courant number then we calculate converge.

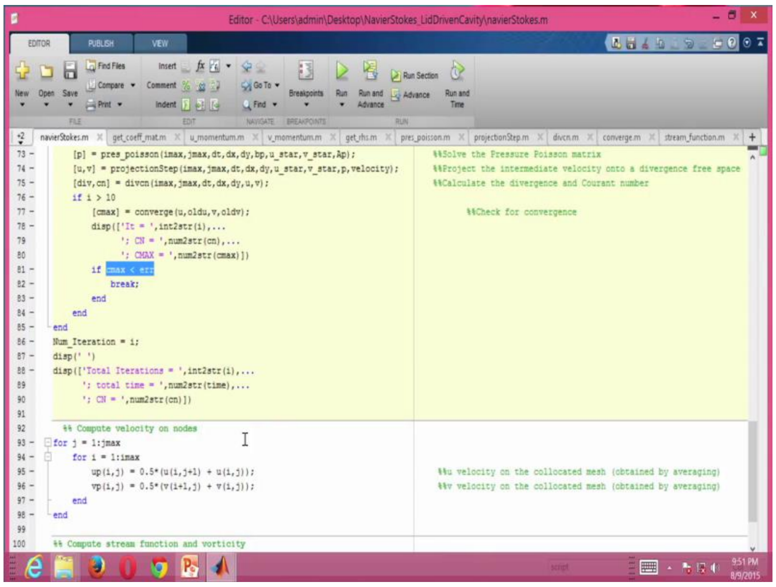
(Refer Slide Time: 29:53)



```
function[rmax]=converge(u,oldu,v,oldv)

rmax1=max(max(abs(u-oldu)));
rmax2=max(max(abs(v-oldv)));
rmax = max(rmax1, rmax2);
return
end
```

Next we go to converge. This finds out the absolute error and finds the maximum of absolute error separately for u velocity and v velocity. And these are lines here and you find the maximum of these two and that is what is attributed here as a c max. So, come back to the main. So, c max is compared against convergence limit that is set; and if it is satisfied then we break; otherwise, we repeat the steps. So, once the solution are obtained then we do a post processing.

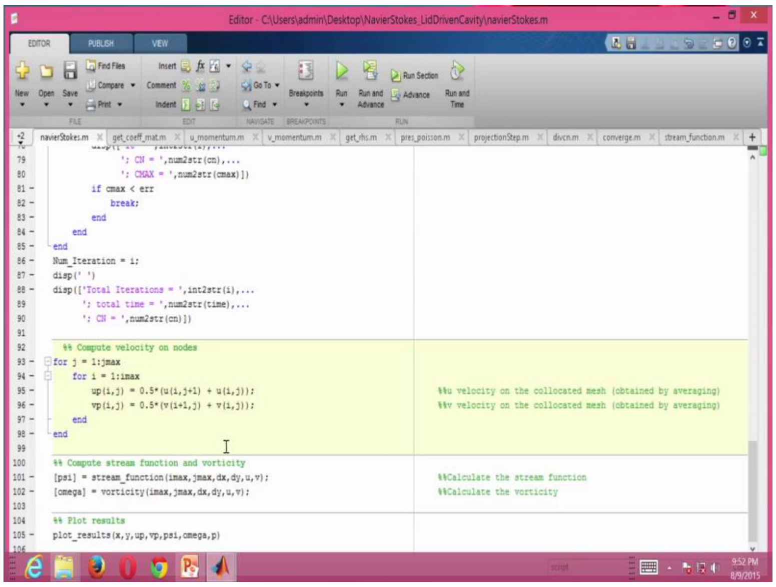
(Refer Slide Time: 30:28)



```
    [p] = pres_poisson(imax,jmax,dt,dx,dy,bp,u_star,v_star,Ap);                 %%Solve the Pressure Poisson matrix
    [u,v] = projectionStep(imax,jmax,dt,dx,dy,u_star,v_star,p,velocity);        %%Project the intermediate velocity onto a divergence free space
    [div,cn] = divcn(imax,jmax,dt,dx,dy,u,v);                                   %%Calculate the divergence and Courant number
    if i > 10
        [cmax] = converge(u,oldu,v,oldv);                                           %%Check for convergence
        disp(['It = ',int2str(i),...
              '; CN = ',num2str(cn),...
              '; CMAX = ',num2str(cmax)])
        if cmax < err
            break;
        end
    end
end
Num_Iteration = i;
disp(' ')
disp(['Total Iterations = ',int2str(i),...
      '; total time = ',num2str(time),...
      '; CN = ',num2str(cn)])

  %% Compute velocity on nodes
for j = 1:jmax
    for i = 1:imax
        up(i,j) = 0.5*(u(i,j+1) + u(i,j));                                      %%u velocity on the collocated mesh (obtained by averaging)
        vp(i,j) = 0.5*(v(i+1,j) + v(i,j));                                      %%v velocity on the collocated mesh (obtained by averaging)
    end
end

%% Compute stream function and vorticity
```

Before going to the post processing, you see the number of displayed commands here. So, display on the screen iteration then CN for Courant number, then c max displays maximum convergence obtained. Similarly at the end of the loop, again we have display of total iteration number of time it has taken and then corresponding Courant number.

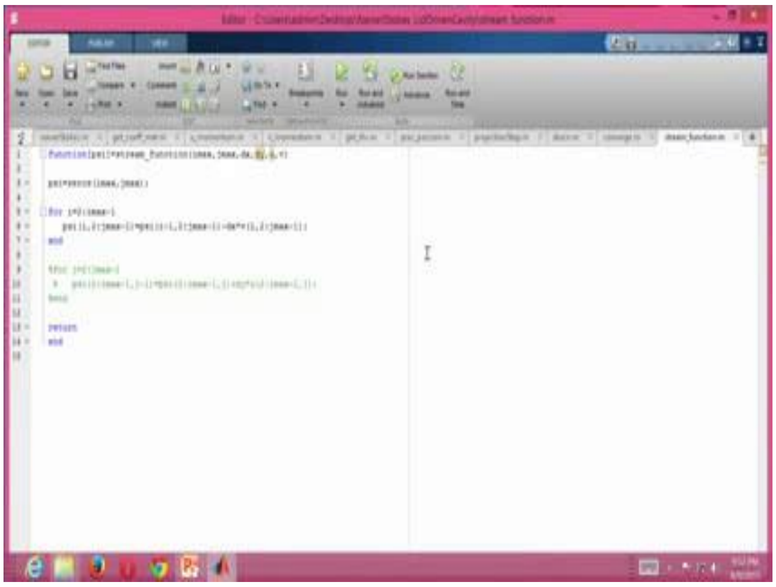
(Refer Slide Time: 31:01)



```
            disp([' it = ',int2str(i),...
                '; CN = ',num2str(cn),...
                '; CMAX = ',num2str(cmax)])
            if cmax < err
                break;
            end
        end
    end
    Num_Iteration = i;
    disp(' ')
    disp(['Total Iterations = ',int2str(i),...
        '; total time = ',num2str(time),...
        '; CN = ',num2str(cn)])

    %% Compute velocity on nodes
    for j = 1:jmax
        for i = 1:imax
            up(i,j) = 0.5*(u(i,j+1) + u(i,j));      %%u velocity on the collocated mesh (obtained by averaging)
            vp(i,j) = 0.5*(v(i+1,j) + v(i,j));      %%v velocity on the collocated mesh (obtained by averaging)
        end
    end

    %% Compute stream function and vorticity
    [psi] = stream_function(imax,jmax,dx,dy,u,v);   %%Calculate the stream function
    [omega] = vorticity(imax,jmax,dx,dy,u,v);       %%Calculate the vorticity

    %% Plot results
    plot_results(x,y,up,vp,psi,omega,p)
```

Then we do a post processing, before doing the post processing we have to convert the staggered arrangement of variables to the collocated arrangement of variables; in other words all the variables are stored at one location. So, it is easy to calculate derivatives and put it to get stream function and vorticity. So, this particular line does the conversion, because we have defined simple uniform grid, it is a basic arithmetic we get summation, average and then we get u and v at collocated location. Once you get u and v we calculate stream function. So, we have a separate module stream function.

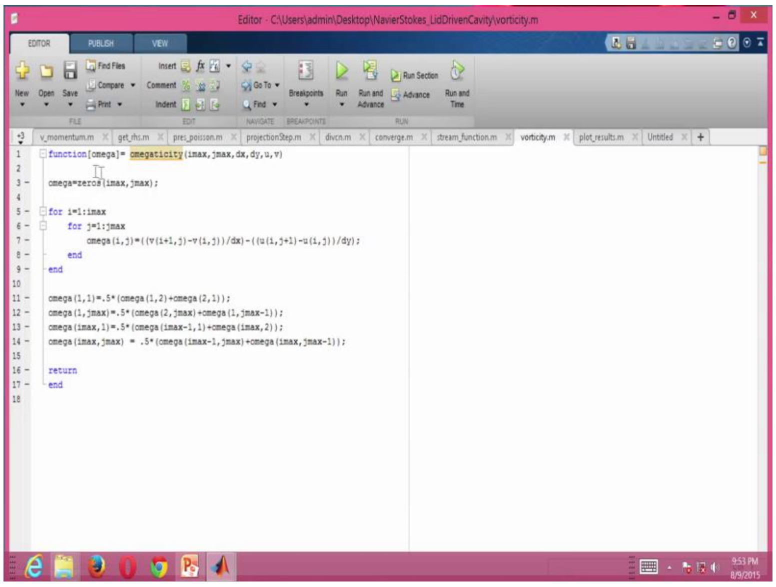
(Refer Slide Time: 31:45)



```
function[psi]=stream_function(imax,jmax,dx,dy,u,v)

psi=zeros(imax,jmax);

for i=2:imax-1
    psi(i,2:jmax-1)=psi(i-1,2:jmax-1)-dx*v(i,2:jmax-1);
end

%for j=2:jmax-1
%    psi(2:imax-1,j)=psi(2:imax-1,j-1)+dy*u(2:imax-1,j);
%end

return
end
```

So, we go to the stream function module and that is calculated as shown here. So, we can calculate stream function either from u velocity or from v velocity. So, what is used here is from u velocity, and then you can also calculate from v velocity. So, both are here. So, one in the green line, green color is based on u velocity, and one that is actually used here is based on v-velocity. Again come back to the main. So, we calculate stream function then we calculate vorticity. So, vorticity again based on velocity.

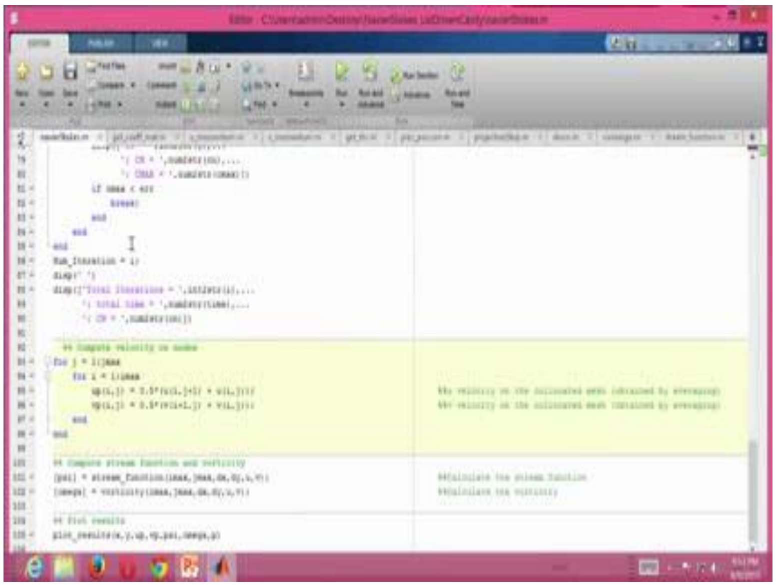
(Refer Slide Time: 32:28)



```
function[omega]= omegaticity(imax,jmax,dx,dy,u,v)

omega=zeros(imax,jmax);

for i=1:imax
    for j=1:jmax
        omega(i,j)=((v(i+1,j)-v(i,j))/dx)-((u(i,j+1)-u(i,j))/dy);
    end
end

omega(1,1)=.5*(omega(1,2)+omega(2,1));
omega(1,jmax)=.5*(omega(2,jmax)+omega(1,jmax-1));
omega(imax,1)=.5*(omega(imax-1,1)+omega(imax,2));
omega(imax,jmax) = .5*(omega(imax-1,jmax)+omega(imax,jmax-1));

return
end
```

So vorticity is shown here, so based on the expression and using the corresponding velocity component, vorticity is defined; and these are lines - specific lines for calculating vorticity at specific locations.

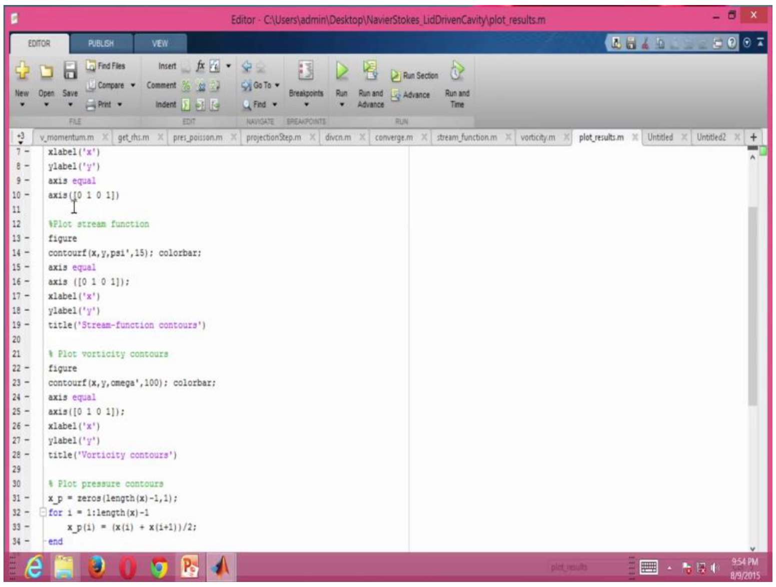
(Refer Slide Time: 32:42)



```
             '; CN = ',num2str(cn),...
             '; CMAX = ',num2str(cmax)])
        if cmax < err
            break;
        end
    end
end
Num_Iteration = i;
disp(' ')
disp(['Total Iterations = ',int2str(i),...
      '; total time = ',num2str(time),...
      '; CN = ',num2str(cn)])

 %% Compute velocity on nodes
for j = 1:jmax
    for i = 1:imax
        up(i,j) = 0.5*(u(i,j+1) + u(i,j));          %u velocity on the collocated mesh (obtained by averaging)
        vp(i,j) = 0.5*(v(i+1,j) + v(i,j));          %v velocity on the collocated mesh (obtained by averaging)
    end
end

%% Compute stream function and vorticity
[psi] = stream_function(imax,jmax,dx,dy,u,v);       %Calculate the stream function
[omega] = vorticity(imax,jmax,dx,dy,u,v);           %Calculate the vorticity

%% Plot results
plot_results(x,y,up,vp,psi,omega,p)
```

So, come back to the main then again we have plotting. So, we calculate stream function vorticity; and from solution, we have velocity and pressure we can plot them and look at the results. So, we go to the separate modules plot results.

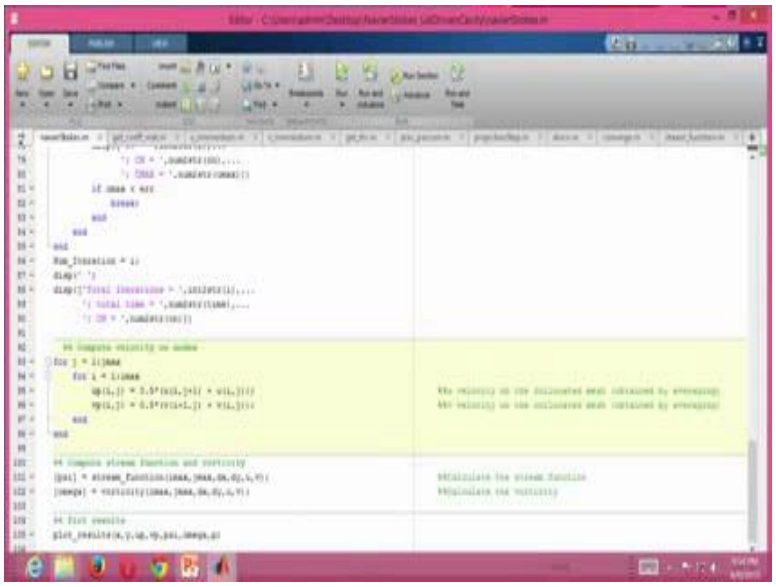
(Refer Slide Time: 33:05)



```
xlabel('x')
ylabel('y')
axis equal
axis([0 1 0 1])

%Plot stream function
figure
contourf(x,y,psi',15); colorbar;
axis equal
axis ([0 1 0 1]);
xlabel('x')
ylabel('y')
title('Stream-function contours')

% Plot vorticity contours
figure
contourf(x,y,omega',100); colorbar;
axis equal
axis([0 1 0 1]);
xlabel('x')
ylabel('y')
title('Vorticity contours')

% Plot pressure contours
x_p = zeros(length(x)-1,1);
for i = 1:length(x)-1
    x_p(i) = (x(i) + x(i+1))/2;
end
```

So, in plot results we have again x axis defined y axis defined, then stream function label is given stream function contour. So, contour function then plot vorticity contour, plot pressure contour then corresponding figure with the color bar then plot centerline velocity. So, let us go back to the main part.

(Refer Slide Time: 33:44)



```
            '; CN = ',num2str(cn),...
            '; CMAX = ',num2str(cmax)])
        if cmax < eps
            break;
        end
    end
end
Num_Iteration = i;
disp(' ')
disp(['Total Iterations = ',int2str(i),...
    '; total time = ',num2str(time),...
    '; CN = ',num2str(cn)])

 %% Compute velocity on nodes
for j = 1:jmax
    for i = 1:imax
        up(i,j) = 0.5*(u(i,j+1) + u(i,j));       %u velocity at the collocated mesh (obtained by averaging)
        vp(i,j) = 0.5*(v(i+1,j) + v(i,j));       %v velocity at the collocated mesh (obtained by averaging)
    end
end

%% Compute stream function and vorticity
[psi] = stream_function(imax,jmax,dx,dy,u,v);    %Calculate the stream function
[omega] = vorticity(imax,jmax,dx,dy,u,v);        %Calculate the vorticity

%% Plot results
plot_results(x,y,up,vp,psi,omega,p)
```

So, we have everything explained. Now we will actually run the code. So, we have already got all the results just look at one-by-one. So, this is the running of the code with the iteration number, Courant number and convergence criteria specified, every line is with that number. So, program has gone around 313 iteration total time is 0.0662, and CN is 0.005862. Now just look at the results this is the velocity vectors, and you can see the velocity with the corresponding arrow mark. Then we have next figure for stream function contour we have explained this separately, now it is run and got along with the code, this is for the vorticity, and this is for pressure, and this is for centerline velocity.

In this module, we have particularly seen projection step, correction step, getting the u and v velocity, and then looking at the results by different post processing. We can have u and v velocity plotted as it is; either in the form of the line or in the form of the contour. We can also look at the results in the form of the pressure contour. In addition, we also seen have to get stream function and vorticity values, and plotting corresponding contours. We have also listed we have also displayed complete listing of the code and explained line-by-line from the beginning of the code till the end. And this completes end of this week eight lecture.

Thank you.