Department of Physics - Ariel University
Computational Physics 4
Dr. Marcelo Schiffer

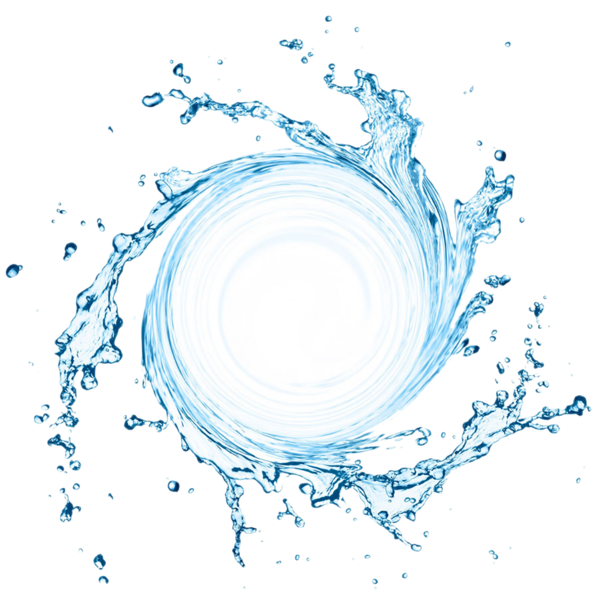
Navier-Stokes equation 2D project

Matlab & C++

Noach Detwiler
Tal Aharon
Yakov Hammer

August 24, 2021

Contents



1 Introduction

The Navier–Stokes equations mathematically express conservation of momentum and conservation of mass for Newtonian fluids. They are sometimes accompanied by an equation of state relating pressure, temperature and density. In this project we will investigate The partial differential equations which describe the motion of viscous fluid. The Naiver Stokes Equation in 2D. Implying CFD (Computational Fluid Dynamics), we calculate the numeric results using C++ & Matlab to create a streamline graph and represent the fluid dynamics of a closed box filled with the same fluid as the fluid coming from the top. The box is open in the top of the square and a constant stream with U=1 is applied. The C++ script will calculate the equations and save them in a file as matrix files and vectors.Using Matlab to create a 2D grid with the data/videos (see Matlab script at the end of the paper) . Our method found was very stable, able to calculate extremely high Reynolds Numbers (Up to 1 million!) and higher given better hardware. C++ was chosen above Python for efficiency purposes.

2 Problem expectations

As one can predict, the fluid will create vortex in the middle of the box.
The expected result after long period of time will be that the vectors of speed in the fluid will create a steady state which will converge until a long period of time. After defining the analytic equations we will use the analytic solution and create the numeric equation that co-respond to the real solution.Using the finite difference method.

3 Governing equations

Vorticity Function:

$$\frac{\partial \vec{v}}{\partial t} + \vec{v} \cdot \nabla \vec{v} = -\frac{\nabla p}{\rho} + v \cdot \nabla^2 \vec{v}, \qquad \vec{v} = \vec{v}(u, v)$$

Horizontal Velocity:

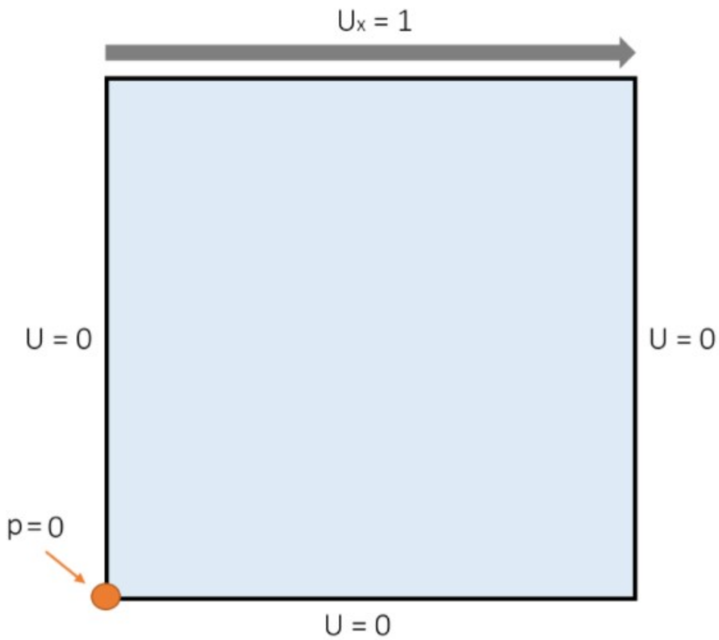
$$\frac{\partial u}{\partial t} + u \cdot \frac{\partial u}{\partial x} + v \cdot \frac{\partial u}{\partial y} = -\frac{1}{\rho} \cdot \frac{\partial p}{\partial x} + v \cdot (\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}) \cdot u, \qquad \nabla \cdot \vec{v} = 0$$

Vertical Velocity:

$$\frac{\partial v}{\partial t} + u \cdot \frac{\partial v}{\partial x} + v \cdot \frac{\partial v}{\partial y} = -\frac{1}{\rho} \cdot \frac{\partial p}{\partial x} + v \cdot (\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}) \cdot v, \qquad \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$

4 Problem orientation

For our assignment, we where given the lid driven cavity flow problem, as shown below:



5 Boundary conditions

Boundary Conditions are given by:
$v(0,y) = v(L,y) = v(x,0) = v(x,L) = 0$ and $u(0,y) = v(L,y) = v(x,0) = 0 \quad v(x,L) = U = 1$

In order to insure these conditions on the borders in C++, The following BC where applied to the code:
For u, horizontal velocity:

$$u(0,j) = u(l,j) = 0 \quad u(i,0) = -u(i,1) \quad u(i,L) = 2 - u(i,L-1)$$

For v, vertical velocity:

$$v(0,j) = -v(1,j) \quad v(L,j) = -v(L-1,j) \quad v(i,0) = v(i,L) = 0$$

For p, Vorticity:

$$p(0,j) = p(1,j) \quad p(L,j) = p(L-1,j) \quad p(i,0) = p(i,1) \quad p(i,L) = p(i,L-1)$$

The solution after reduction will come to two equation which we can write numerically:

$$\frac{\partial \omega'}{\partial t'} + u'\frac{\partial \omega'}{\partial x'} - v'\frac{\partial \omega'}{\partial y'} = \frac{1}{Re}(\frac{\partial^2}{\partial x'^2} + \frac{\partial^2}{\partial y'^2})\omega'$$

For the virtuosity (pressure does not appear) v' and u' are functions of the derivative of Ψ according to x' and y' :

$$v' = -\frac{\partial \Psi'}{\partial x'} \qquad u' = \frac{\partial \Psi'}{\partial y'}$$

$$\frac{\partial \omega'}{\partial t'} + \frac{\partial \Psi'}{\partial y'}\frac{\partial \omega'}{\partial x'} - \frac{\partial \Psi'}{\partial x'}\frac{\partial \omega'}{\partial y'} = \frac{1}{Re}(\frac{\partial^2}{\partial x'^2} + \frac{\partial^2}{\partial y'^2})\omega'$$

Laplace equation for the stream function with the virtuosity as the source term:

$$\frac{\partial^2 \Psi}{\partial x'^2} + \frac{\partial^2 \Psi}{\partial y'^2} = -\omega'$$

and we can find the velocity:

$$\omega' = \frac{\partial v'}{\partial x'} - \frac{\partial u'}{\partial y'}$$

$$v' = -\frac{\partial \Psi}{\partial x'} \qquad u' = \frac{\partial \Psi}{\partial y'}$$

Reynolds number will be defined by:

$$Re = \frac{L}{vU} = Const$$

6 Finite Difference methods

The Forward in Time, Center in Space (FTCS) finite difference method was chosen for increased accuracy and simplicity.

this scheme approximates the original equation with errors of first order in the time interval and second order in spatial coordinate grid

Streamform Function u,v

The FTCS euqation in C++ for u horizontal flow

$$u_{i,j}^{new} = u_{i,j} - dt\left(\frac{(u_{i+1,j})^2 - (u_{i-1,j})^2}{2dx} + \frac{1}{4}\frac{(u_{i,j}+u_{i,j+1})(v_{i,j}+v_{i+1,j}) - (u_{i,j}+u_{i,j-1})(v_{i+1,j-1}+v_{i,j-1})}{dy}\right) -$$

$$-\frac{dt}{dx}(p_{i+1,j} - p_{i,j}) + \frac{dt}{Re}\left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{dx^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{dy^2}\right) \quad [1]$$

for vertical flow v:

$$v_{i,j}^{new} = v_{i,j} - dt\cdot\left(\frac{1}{4}\left(\frac{(u_{i,j}+u_{i,j+1})(v_{i,j}+v_{i+1,j}) - (u_{i-1,j}+u_{i-1,j+1})(v_{i,j}+v_{i-1,j})}{dx}\right) + \frac{(v_{i,j+1})^2 - (v_{i,j-1})^2}{2dy}\right) -$$

$$-\frac{dt}{dy}(p_{i,j+1} - p_{i,j}) + \frac{dt}{Re}\left(\frac{v_{i+1,j} - 2v_{i,j} + v_{i-1,j}}{dx^2} + \frac{v_{i,j+1} - 2v_{i,j} + v_{i,j-1}}{dy^2}\right) \quad [2]$$

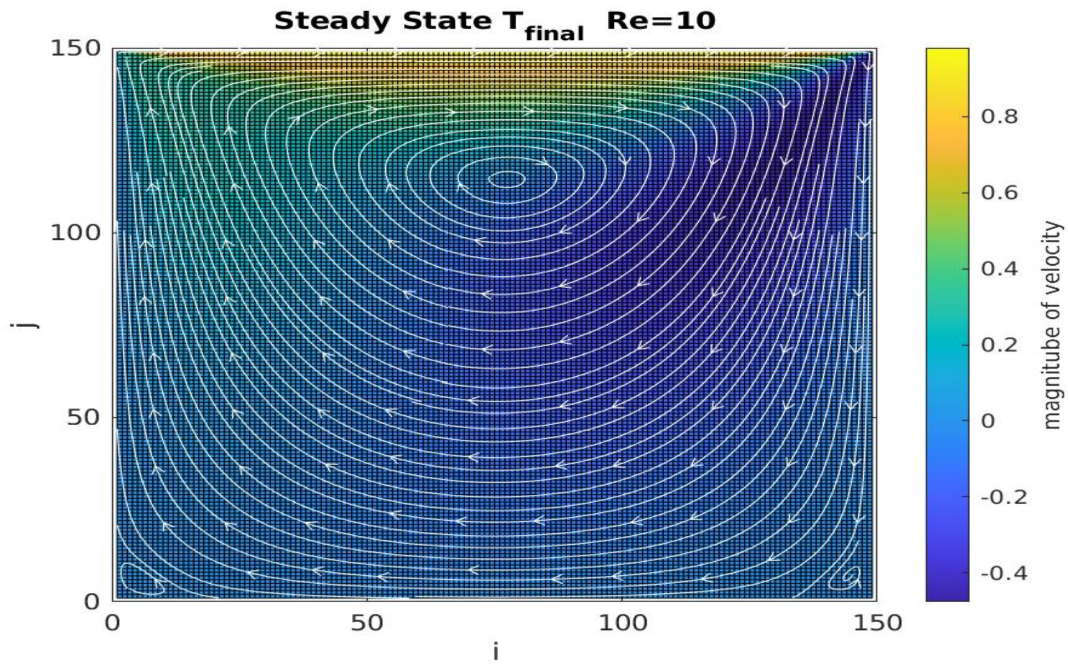
Vorticiy function p:

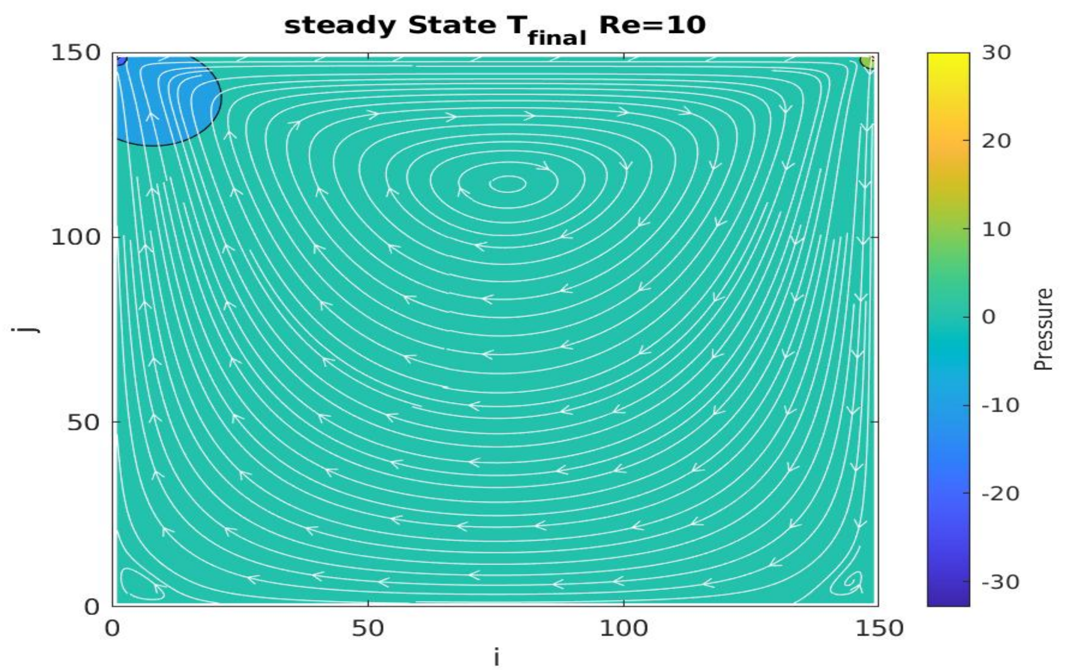
$$p_{i,j}^{new} = p_{i,j} - \Delta \cdot dt\left(\frac{u_{i,j}^{new} - u_{i-1,j}^{new}}{dx} + \frac{v_{i,j}^{new} - v_{i,j-1}^{new}}{dy}\right) \quad [3]$$

7 Examples with different Reynolds numbers.

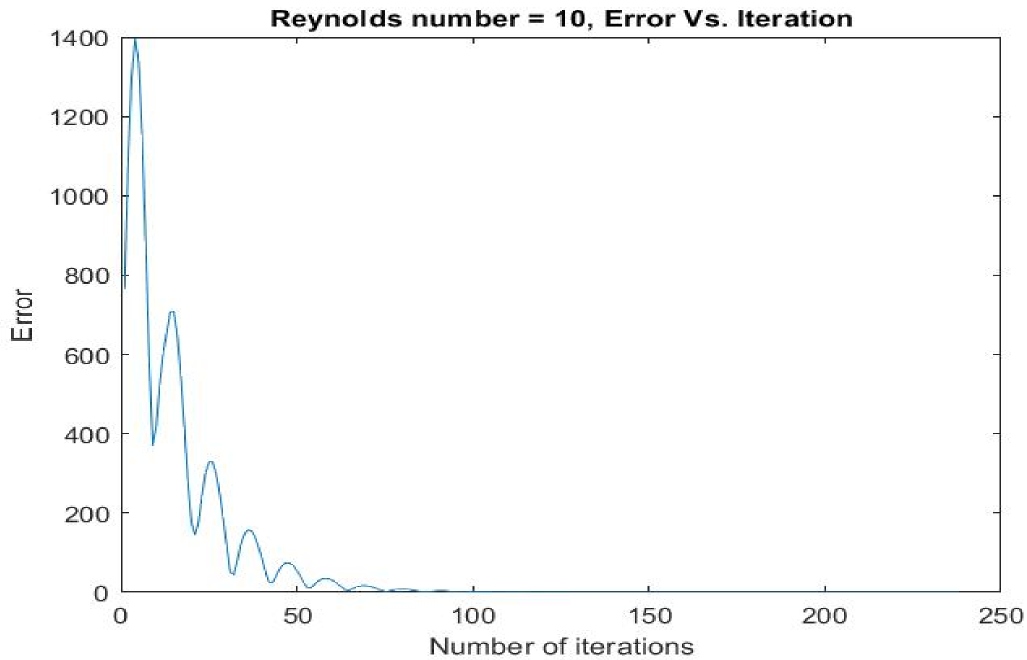
8 Reynolds number 10

YOUTUBE https://youtu.be/5_kY52rVcME

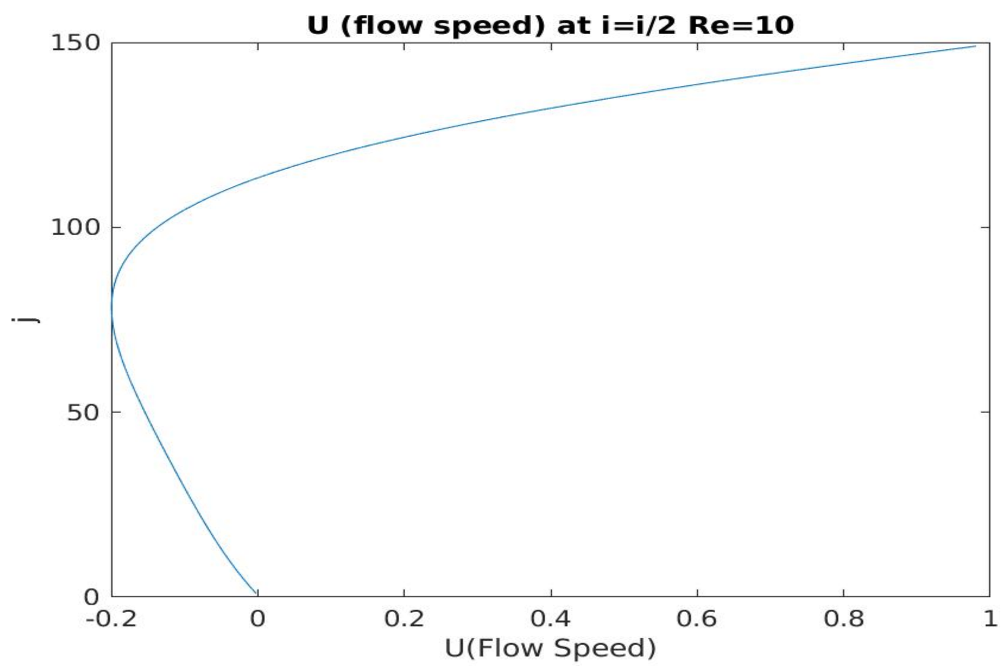




The error convergence over iterations while the code was running.

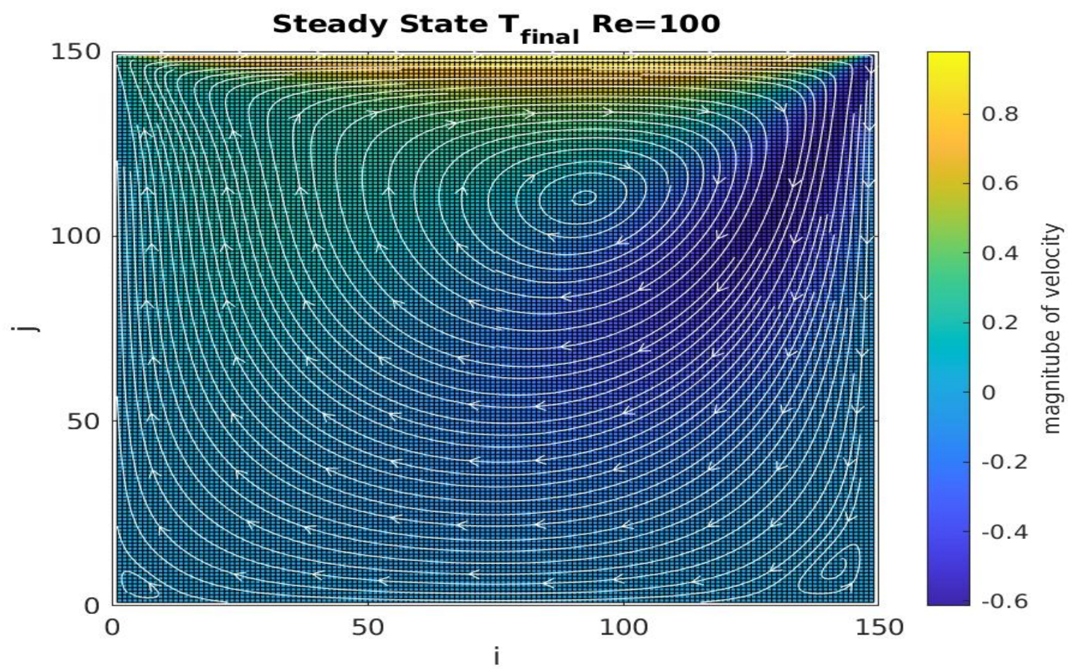


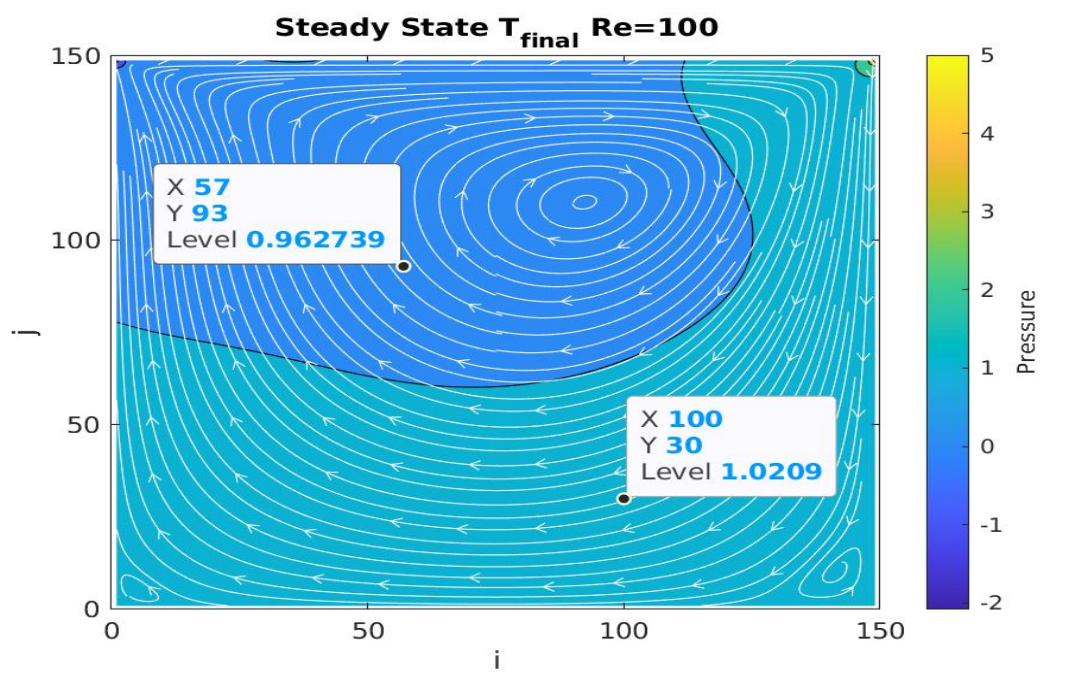
Speed U changes from negative to positive and ends up in 1 at the top end of the graph.



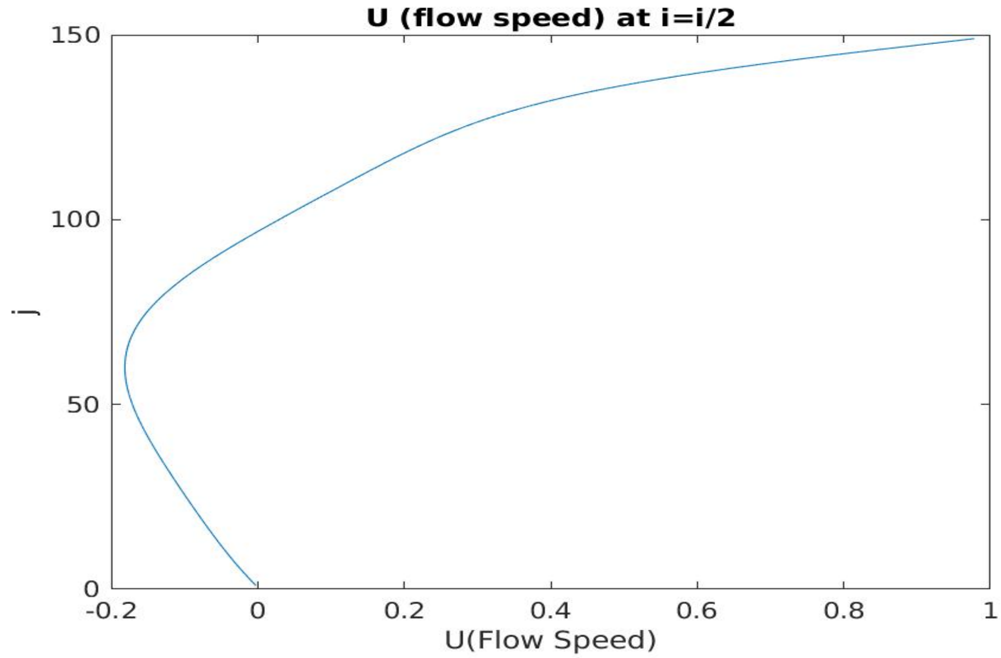
9 Reynolds number 100

YOU TUBE `https://youtu.be/gRE-5bY-LBg`

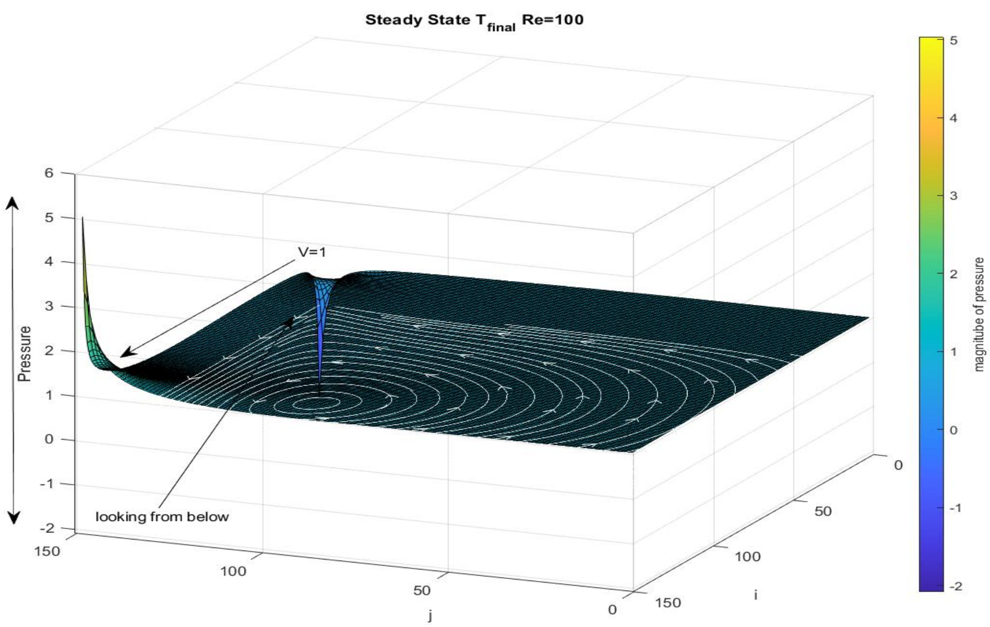




Speed U changes from negative to positive and ends up in 1 at the top end of the graph.

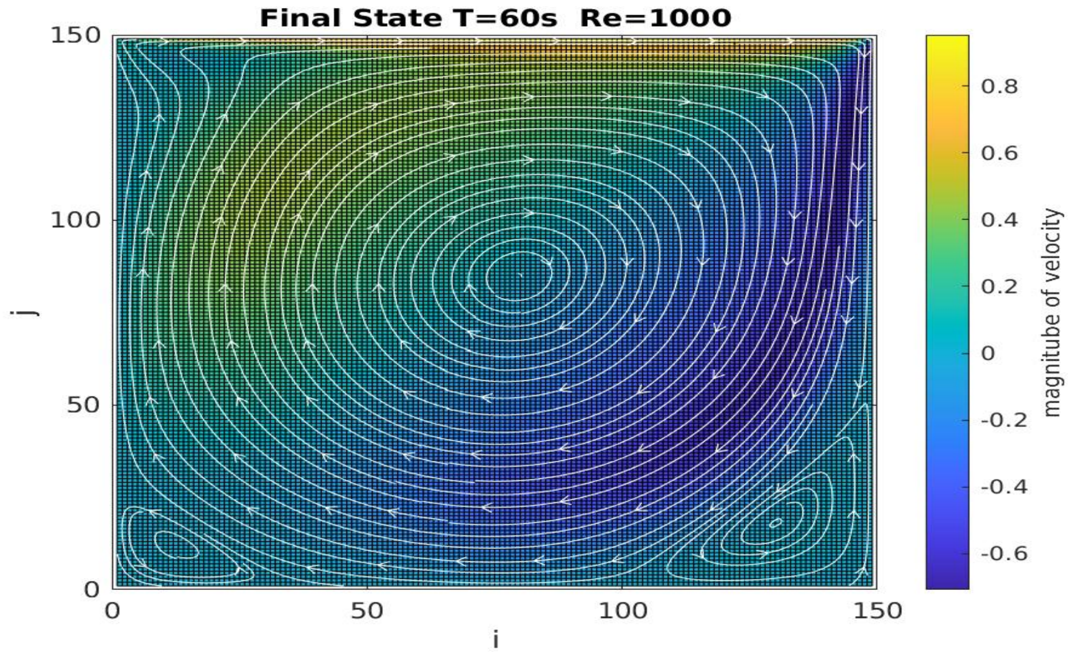


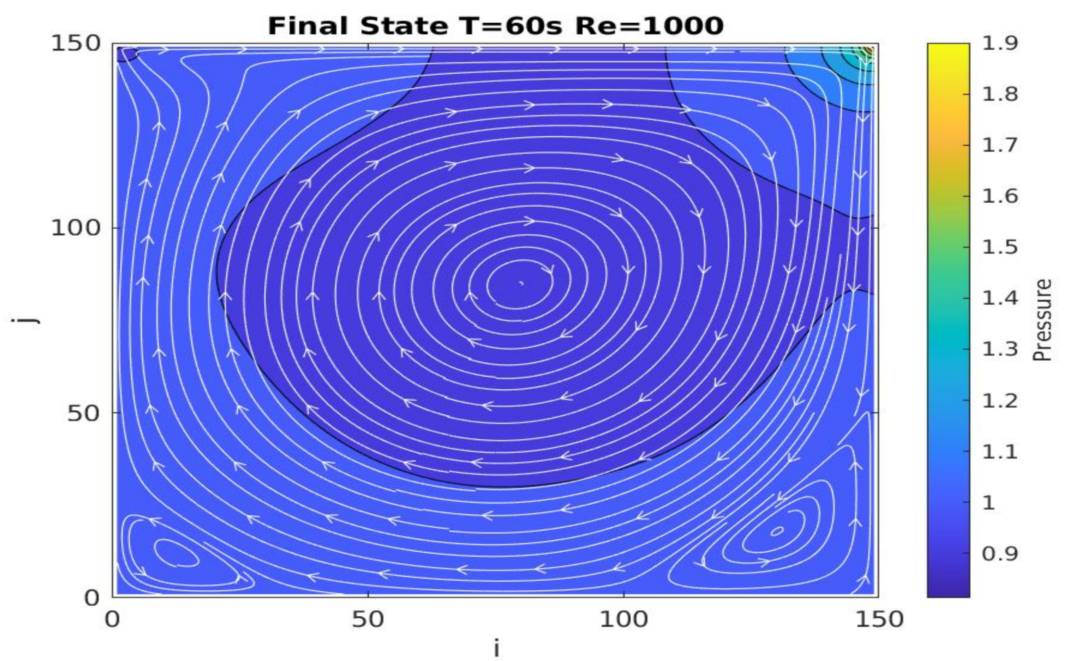
A graph to describe and visualise the vortex and the movement of the fluid.



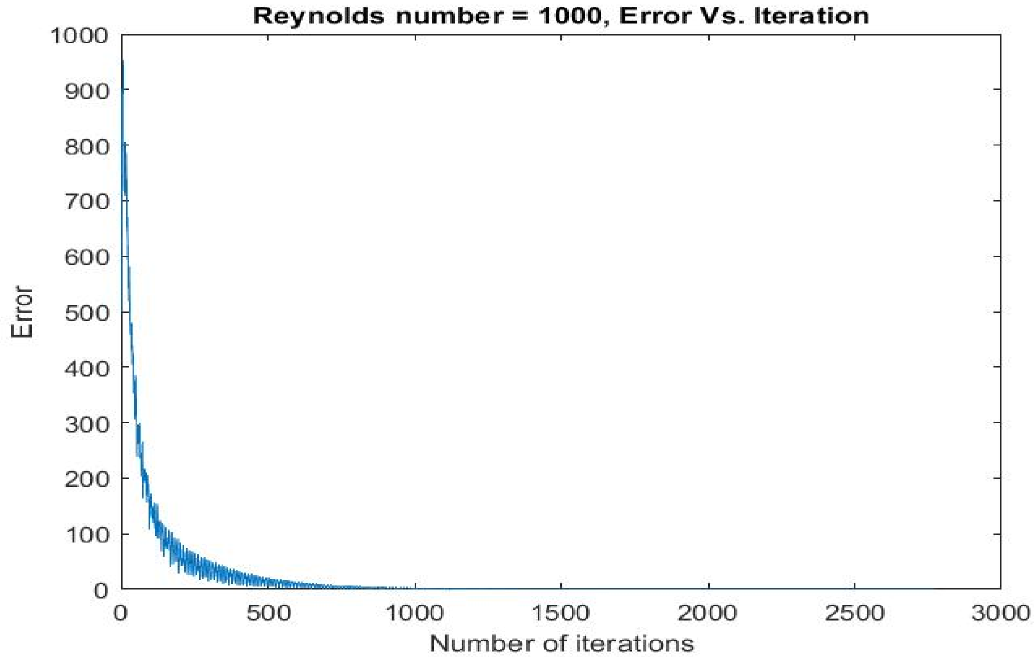
10 Reynolds number 1,000

$YOUTUBE : https : //youtu.be/Lz8YH_fYvgU$

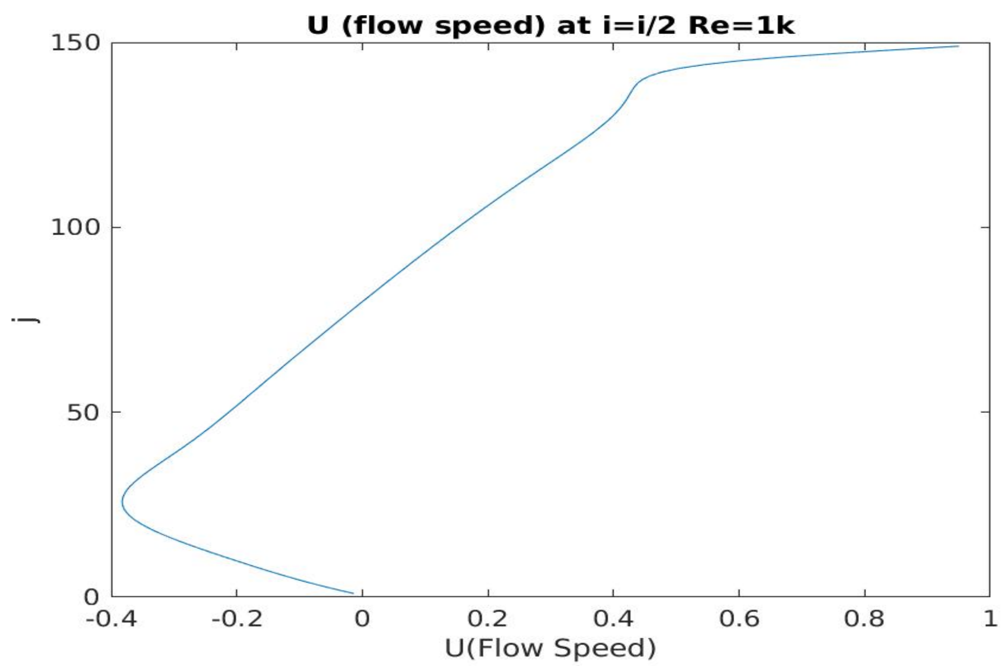




The error convergence over iterations while the code was running.

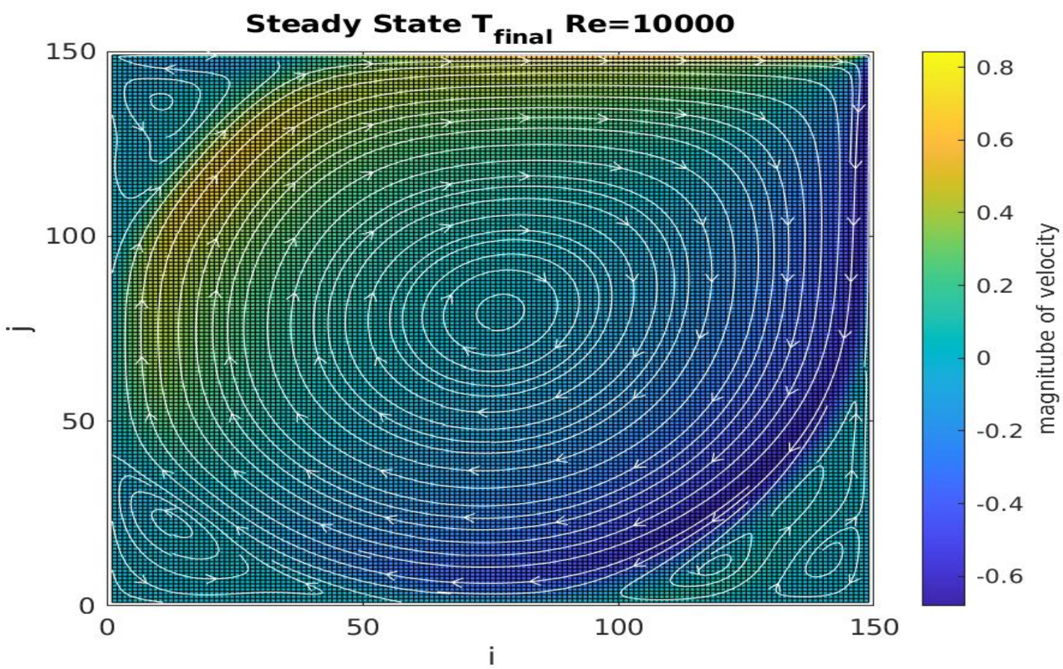


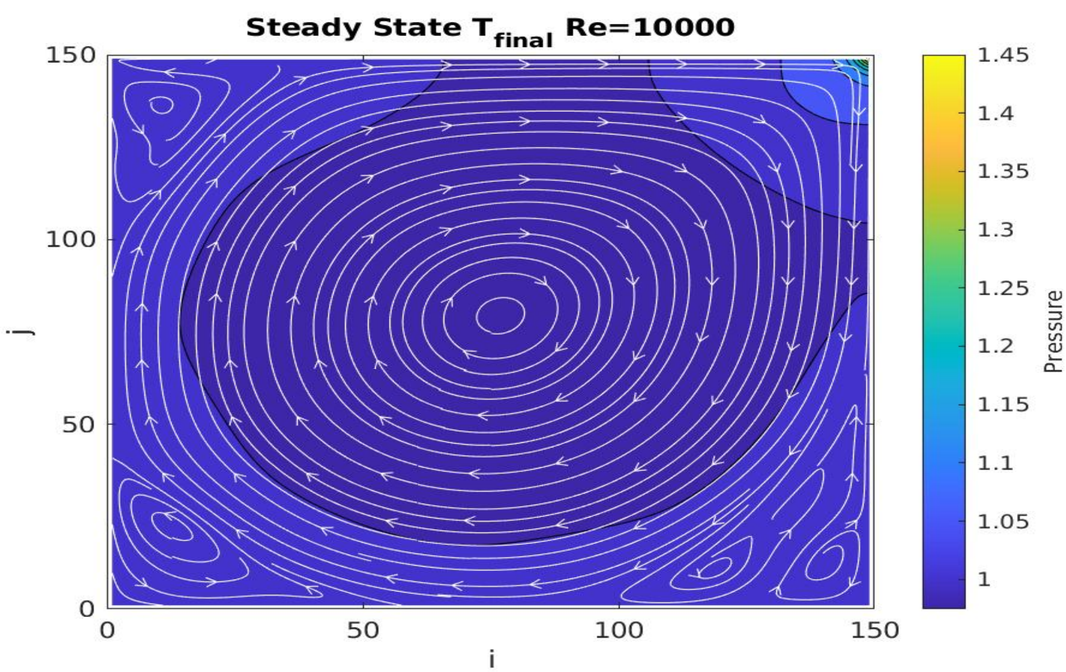
Speed U changes from negative to positive and ends up in 1 at the top end of the graph.



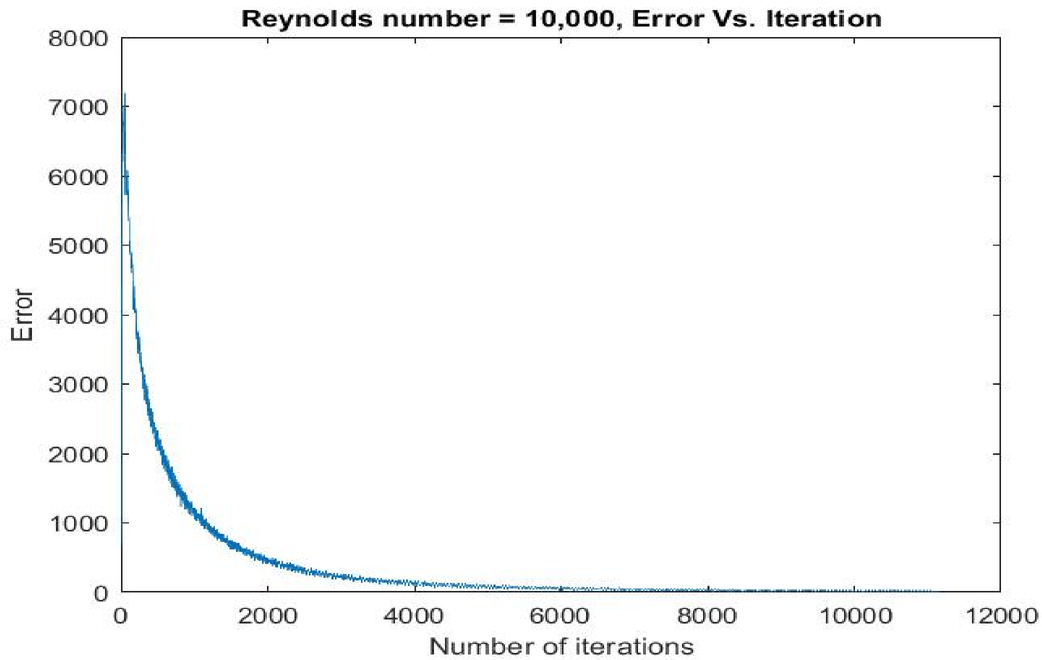
11 Reynolds number 10,000

YOU TUBE: `https://youtu.be/DasNqD4DGUM` YOUTUBE 32K PART 1: `https://youtu.be/kohhJDQAgJY`



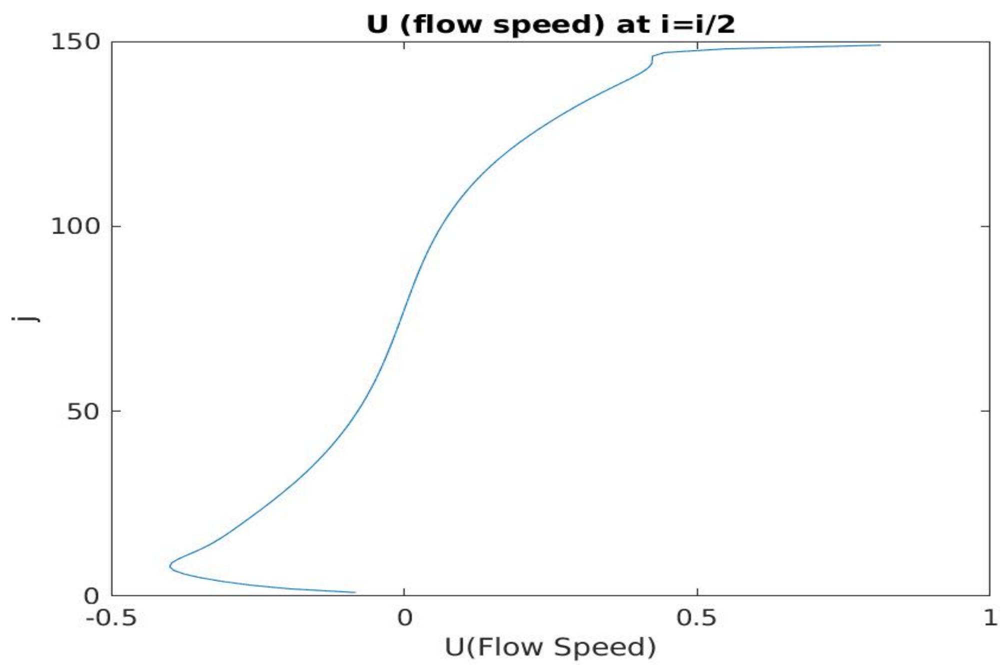


The error convergence over iterations while the code was running.



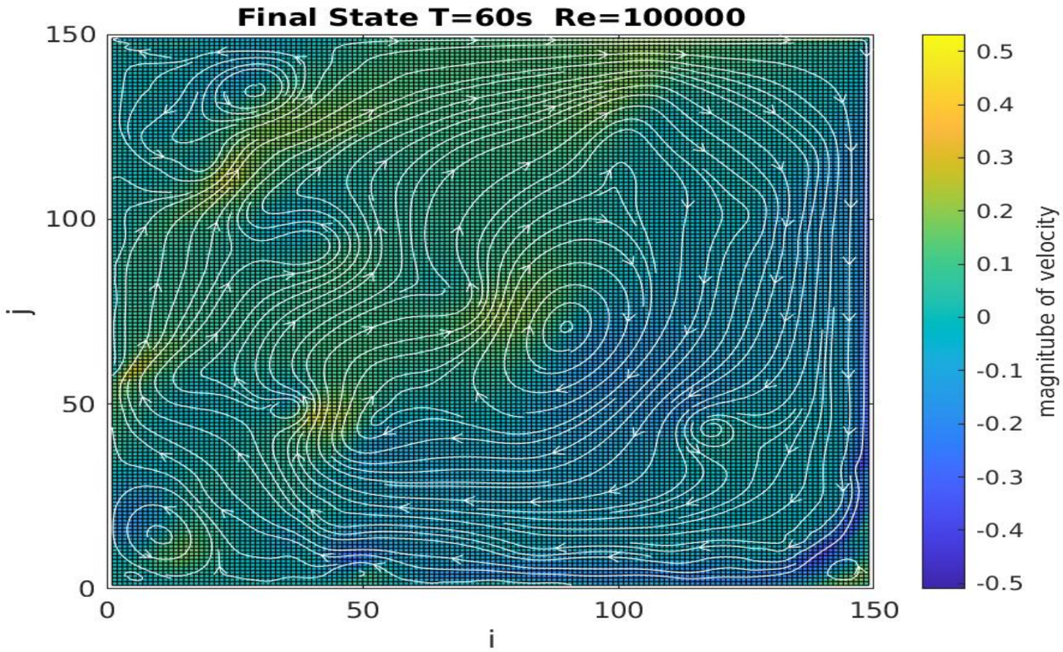
Speed U changes from negative to positive and ends up in 1 at the top end of the graph.

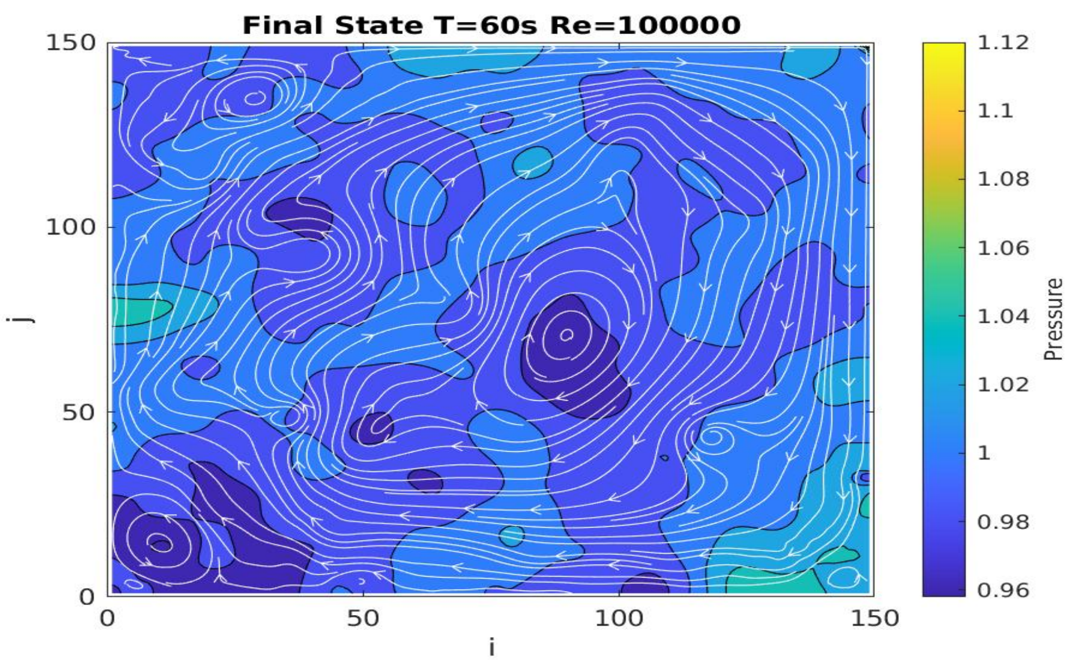
The error convergence over iterations while the code was running.



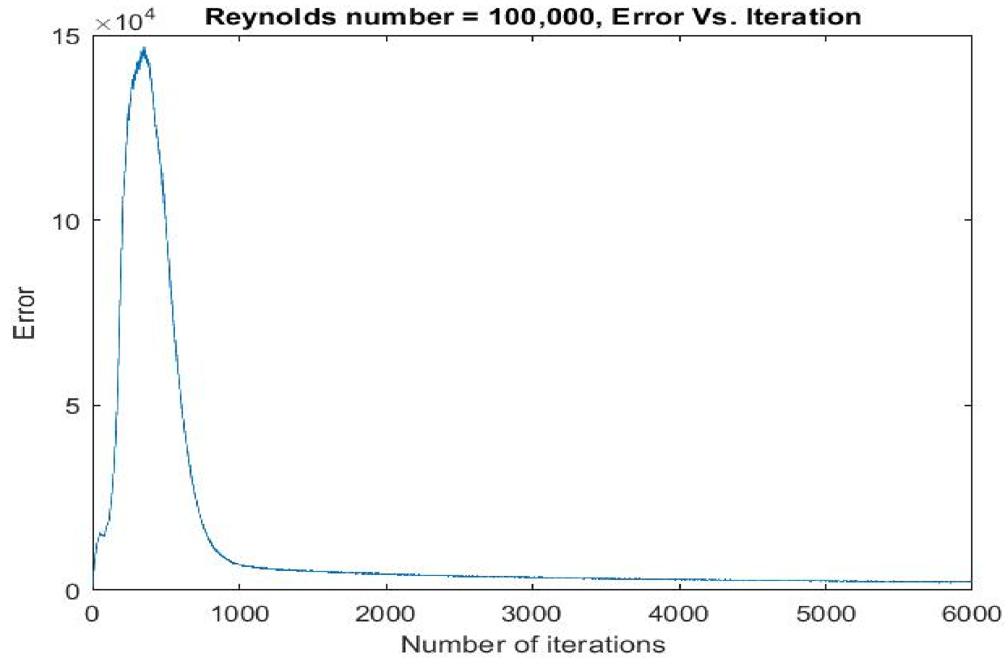
12 Reynolds number 100,000

YOUTUBE `https://youtu.be/Pgzn7NACCQo`

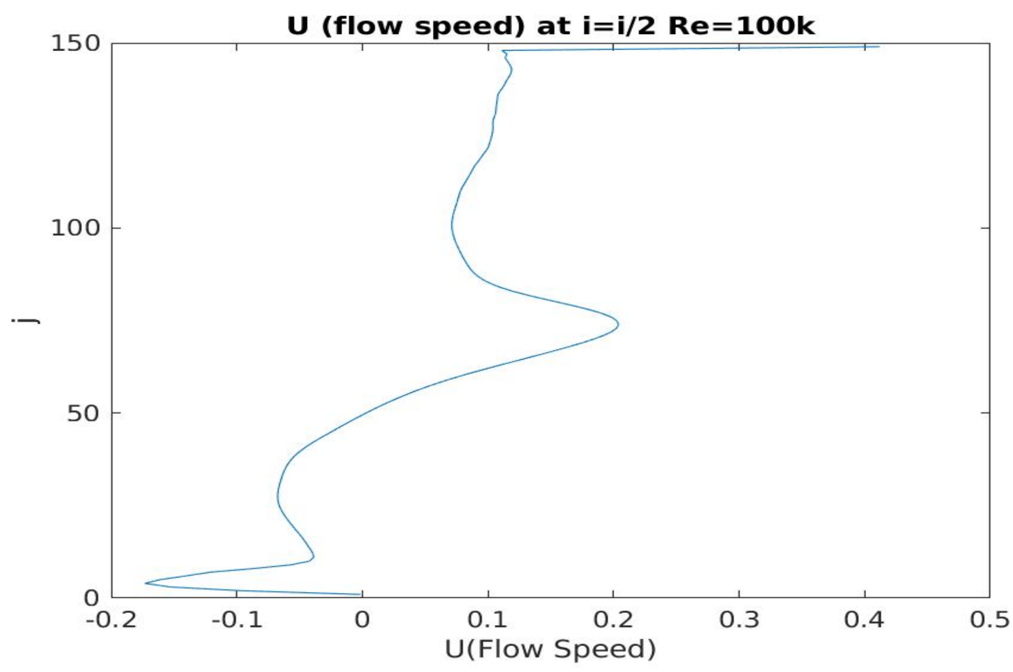

Final State T=60s Re=100000
j
i
magnitube of velocity
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5


Final State T=60s Re=100000
j
i
Pressure
1.12
1.1
1.08
1.06
1.04
1.02
1
0.98
0.96

The error convergence over iterations while the code was running.

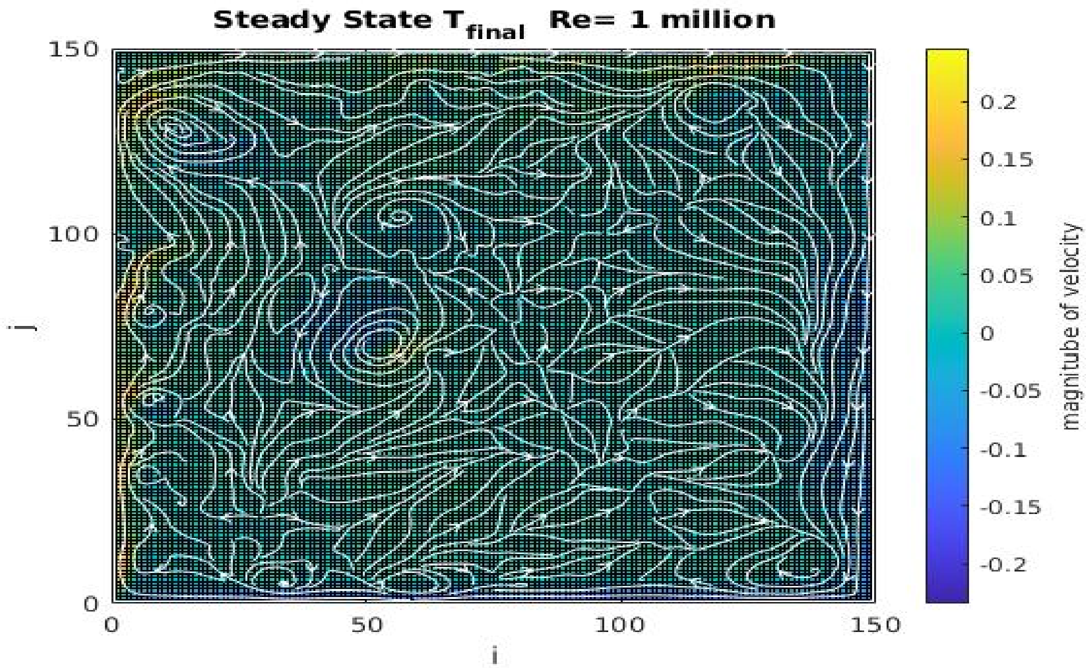


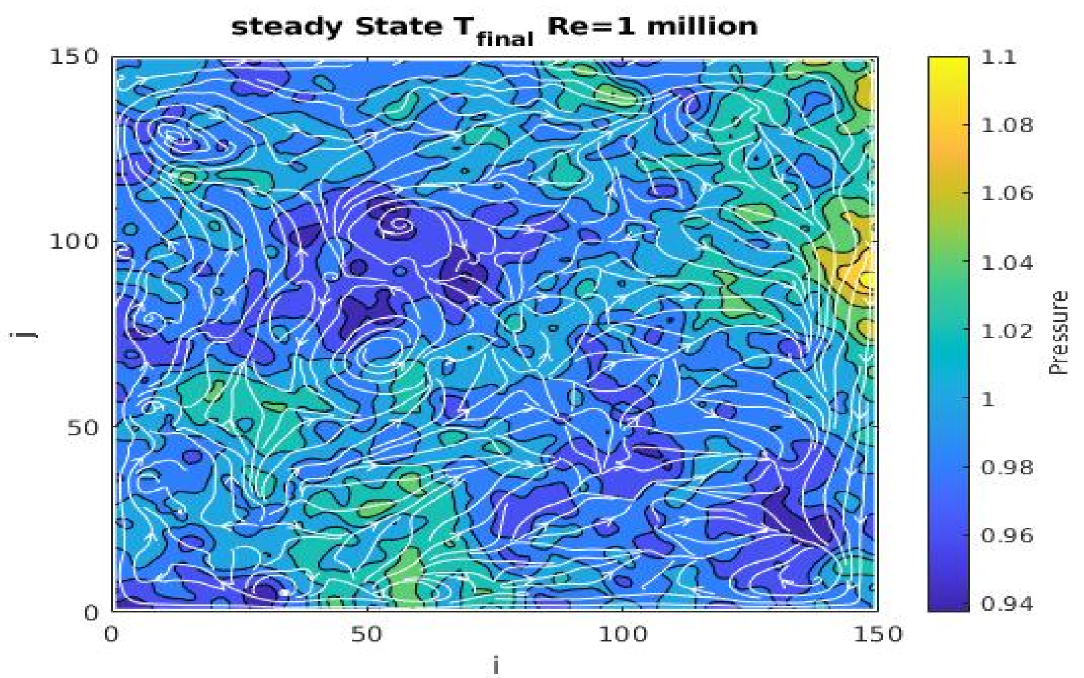
Speed U changes from negative to positive and ends up in 1 at the top end of the graph.



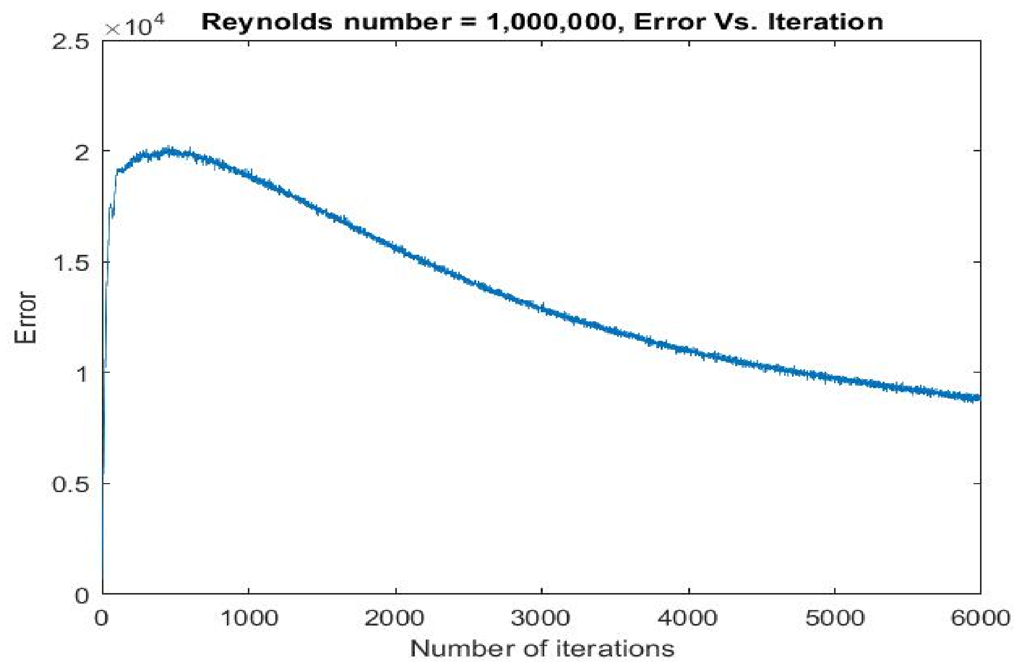
13 Reynolds number 1,000,000

Our Hardware's capabilities where shown here, we did not reach any solid conclusions, Given enough time and like 60gb of ram one could run the code.

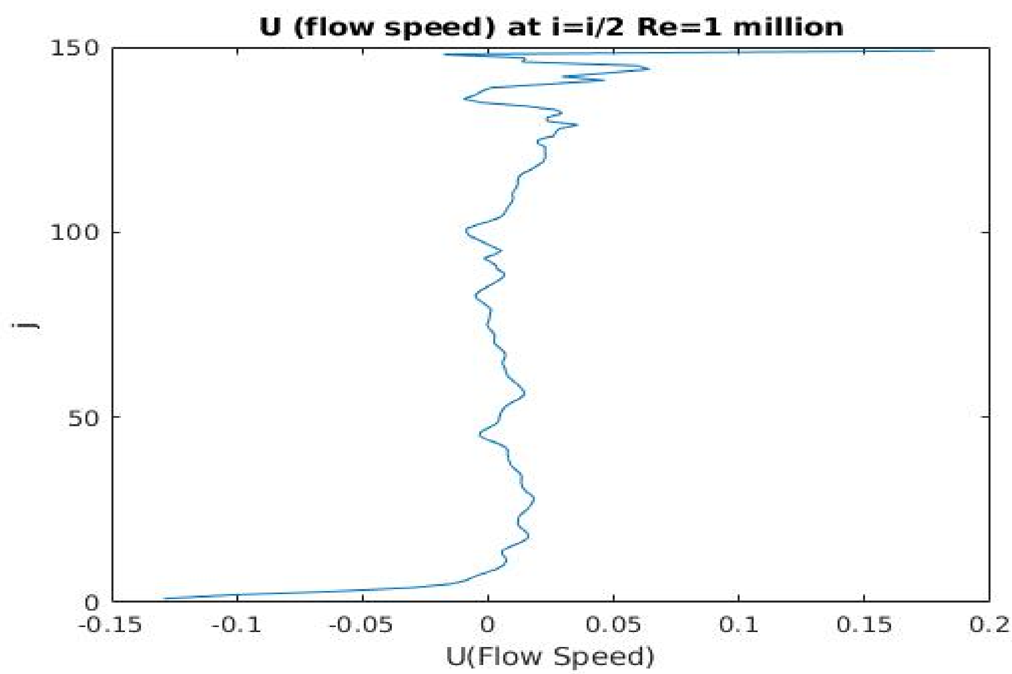




The error convergence over iterations while the code was running.



Speed U changes from negative to positive and ends up in 1 at the top end of the graph.



YouTube for all takes

(You can got to the video by double clicking the link) $Re = 10$:
https://youtu.be/5_kY52rVcME
$Re = 100$:
https://youtu.be/gRE-5bY-LBg
$Re = 1000$:
https://youtu.be/Lz8YH_fYvgU
$Re = 10k$:
https://youtu.be/DasNqD4DGUM
$Re = 32k$:
https://youtu.be/kohhJDQAgJY
$Re = 100k$:
https://youtu.be/Pgzn7NACCQo

14 Source code & Matlab

MATLAB SCRIPT

```matlab
Datap=load('Navier_stokes.dat');     %Loading In Data that was processed by Matlab
Datau=load('Navier_stokes2.dat');     % "
Datav=load('Navier_stokes3.dat');     % "
%%
n=199;                                % Gridsize from c++ - 1
G=length(Datav)/(n*n);                % Finding how many instances we have
%%
cp=num2cell(reshape(Datap, n*n, G ),1);  %rescaling/orgainzing vectors for plotting

cu=num2cell(reshape(Datau, n*n, G ),1); %rescaling/orgainzing vectors for plotting

cv=num2cell(reshape(Datav, n*n, G ),1); %rescaling/orgainzing vectors for plotting
%%
for(i=1:length(cp))
 C1p{i}=reshape(cp{i},n, n);% again reshaping from a 1xn^2 vector to nxn grid
end
for(i=1:length(cp))
 C1u{i}=reshape(cu{i},n, n);% again reshaping from a 1xn^2 vector to nxn grid
end
for(i=1:length(cp))
 C1v{i}=reshape(cv{i},n, n);% again reshaping from a 1xn^2 vector to nxn grid
end
%% VIDEO PRINTING
% the videos where quite large, so we must split the videos in four parts,
% j for 1:G, i for video purposes.

j=1;
for i=1:1500
   hold on

      %contourf(C1p{j},x,':');
      q=pcolor(C1u{j}+C1v{j});
      set(q, 'EdgeColor', 'none');
      %h=quiver(C1u{j},C1v{j},5);
      %set( h, 'Color', 'w' )
     axis([0 200 0 200])
      xlabel('i')
      ylabel('j')

      c = colorbar;
      c.Label.String = ' U_x Velocity';
      %caxis([0 0.7])
      %set(gcf, 'Position', get(0, 'Screensize'));
      drawnow
      frame(i)=getframe(gcf);
      hold off
      cla reset;
      j=j+1
end
```

```
%%
video = VideoWriter( 'part1.avi' , 'Motion JPEG AVI' ) ;
video.FrameRate=30;
open(video)
writeVideo(video, frame);
close(video)
%%
j=1500;
for i=1:1500
  hold on

     %contourf(C1p{j},x,':');
     q=pcolor(C1u{j}+C1v{j});
     set(q, 'EdgeColor', 'none');
     %h=quiver(C1u{j},C1v{j},4);
     %set( h, 'Color', 'w' )
    axis([0 200 0 200])
     xlabel('i')
     ylabel('j')

     c = colorbar;
     c.Label.String = 'U_x Velocity';
     %caxis([min(Datap) max(Datap)])
     %set(gcf, 'Position', get(0, 'Screensize'));
     drawnow
     frame(i)=getframe(gcf);
     hold off
     cla reset;
     j=j+1
end
%%
video = VideoWriter( 'part2.avi' , 'Motion JPEG AVI' ) ;
video.FrameRate=30;
open(video)
writeVideo(video, frame);
close(video)
pause(15)
%%
j=3000;
for i=1:1500
  hold on

     %contourf(C1p{j},x,':');
     q=pcolor(C1u{j}+C1v{j});
     set(q, 'EdgeColor', 'none');
     h=quiver(C1u{j},C1v{j},5);
     set( h, 'Color', 'w' )
    axis([0 200 0 200])
     xlabel('i')
     ylabel('j')

     c = colorbar;
     c.Label.String = 'U_x Velocity';
```

```
      %caxis([min(Datap) max(Datap)])
      %set(gcf, 'Position', get(0, 'Screensize'));
      drawnow
      frame(i)=getframe(gcf);
      hold off
      cla reset;
      j=j+1
end
%%
video = VideoWriter( 'part3.avi' , 'Motion JPEG AVI' ) ;
video.FrameRate=30;
open(video)
writeVideo(video, frame);
close(video)
pause(15)
%%
j=4500;
for i=1:1500
    hold on

      %contourf(C1p{j},x,':');
      q=pcolor(C1u{j}+C1v{j});
      set(q, 'EdgeColor', 'none');
      h=quiver(C1u{j},C1v{j},7);
      set( h, 'Color', 'w' )
     axis([0 200 0 200])
      xlabel('i')
      ylabel('j')

      c = colorbar;
      c.Label.String = ' U_x Velocity';
      %caxis([min(Datap) max(Datap)])
      %set(gcf, 'Position', get(0, 'Screensize'));
      drawnow
      frame(i)=getframe(gcf);
      hold off
      cla reset;
      j=j+1
end
video = VideoWriter( 'part4.avi' , 'Motion JPEG AVI' ) ;
video.FrameRate=30;
open(video)
writeVideo(video, frame);
close(video)
pause(15)
%% Plots
U=C1u{end};
V=C1v{end};
P=C1p{end};
figure(7)
contourf(P);
hold on
 o=streamslice(U,V,2);
```

```

  set( o, 'Color', 'w' )
        axis([0 150 0 150])
     xlabel('i')
     ylabel('j')


     c = colorbar;
     c.Label.String = 'Pressure';
title('Final State T_{final} Re=100k')
figure(2)
pcolor(U+V);
hold on
 o=streamslice(U,V,2);

  set( o, 'Color', 'w' )
        axis([0 150 0 150])
     xlabel('i')
     ylabel('j')


     c = colorbar;
     c.Label.String = 'magnitube of velocity';
title('Final State T_{final}  Re=100k')

figure(3)
centerline=U(:,75)+V(:,75)
 plot(centerline,[1:199])
  title('U (flow speed) at i=i/2 Re=100k')
  xlabel('U(Flow Speed)')
  ylabel('j')
```

———————— **C++ SOURCE CODE** ————————

```
#include <iostream>                                   // ------------------------------
#include <stdio.h>                                    // This Script was created by
#include <stdlib.h>                                   //
#include<time.h>                                      //      Noach Detwiler
#include<math.h>                                      //           &
#include <fstream>                                    //        Tal Aharon
                                                      // ------------------------------
using namespace std;
//------------------------------------------------------------------------------------
// This script will lay out the calculation of the Navier_stokes equation.
// 2D partial differential equations which describes Lid driven flow
// in compressible fluid with a two-sided lid-driven square cavity.
// Given by the finite difference method (FDM) we will calculate
// the numeric results using  C++.
//------------------------------------------------------------------------------------

    int main(){                                       // Initializing all variables.

        int n, i, j,d ;

        n=150;                                        // Defining the grid-size

        double u[n][n+1],un[n][n+1],uc[n][n+1];       // Initializing all needed arrays
        double v[n+1][n],vn[n+1][n],vc[n+1][n];       // u for horizontal velocity
        double p[n+1][n+1],pn[n+1][n+1],pc[n+1][n+1];// v for vertical velocity
        double m[n+1][n+1];                           // p for vorticity
        double dx,dy,dt,delta,err,Re,t;               // prepering the steps for the looop.
        int dmax=100000;                              // Size

        ofstream  myfile("Navier_stokes.dat");        // Opening data file for later use.
        ofstream  myfile2("Navier_stokes2.dat");      // Opening data file for later use.
        ofstream  myfile3("Navier_stokes3.dat");
        ofstream  myfile4("Navier_stokes4.dat");      // Opening data file for later use.
        myfile.precision(17);                         // Defining precision for each file.
        myfile2.precision(17);
        myfile3.precision(17);
        myfile4.precision(17);
        string buf;                                   // String stream buffer.


       d=1;             // intializing step counter, d
       dx = 1.0/(n-1);// dx = 1/gridsize
       dy = dx;         // we are using a square, therefore dy=dx
       dt = 0.0001;     // dt is chosen to fit the gridsize and a comfortable divergence num.
       delta = 4.0;     // WHAT IS THIS I DO NOT KNOW
       err = 16.0;      // Setting intial error above tolerance to start the loop
       Re = 100.0;      // Reynolds number for water: ~32e3
       double Divergence1, Divergence2;
    Divergence1 = dt/dx;
    Divergence2 = (1.0/Re)*dt/(dx*dx);
    printf("Divergence numbers are %lf and %lf\n", Divergence1, Divergence2);
```

```

// -----------------------------------------------------------------------
// u,v,p startup and setting Initial Conditions
//
          for(i=0;i<=(n-1);i++)
               for(j=0;j<=(n);j++){

                    u[i][j]=0.0;
                    u[i][n]=1.0;
                    u[i][n-1]=1.0;}

          for(i=0;i<=(n);i++)
               for(j=0;j<=(n-1);j++){
                    v[i][j]=0.0;}

          for(i=0;i<=(n);i++)
               for(j=0;j<=(n);j++){
                    p[i][j]=1.0;}
// -----------------------------------------------------------------------
//FD equation number [1] see text on page number 3.
t=0.0;
while(err > 0.001){
// Interior Points Calculation
for (i=1;i<=(n-2);i++)
for (j=1;j<=(n-1);j++){
     un[i][j] = u[i][j] - dt*((u[i+1][j]*u[i+1][j]-u[i-1][j]*u[i-1][j])/2.0/dx
     + 0.25*( (u[i][j]+u[i][j+1])*(v[i][j]+v[i+1][j])
     - (u[i][j]+u[i][j-1])*(v[i+1][j-1]+v[i][j-1]))/dy)
     - dt/dx*(p[i+1][j]-p[i][j]) + dt*1.0/Re*( (u[i+1][j]
     - 2.0*u[i][j]+u[i-1][j])/dx/dx
     + (u[i][j+1]-2.0*u[i][j]+u[i][j-1])/dy/dy );}

// B.C.
          for(j=1;j<=(n-1);j++){
               un[0][j]=0.0;
               un[n-1][j]=0.0;}
// B.C.
          for(i=0;i<=(n-1);i++){
               un[i][0]=-un[i][1];
               un[i][n]=2.0-un[i][n-1];}

//-----------------------------------------------------------------------
// FD equation number [2] see text on page number 3.
// Solves v-momentum

for (i=1; i<=(n-1); i++)
for (j=1; j<=(n-2); j++){
     vn[i][j] = v[i][j] - dt* ( 0.25*( (u[i][j]+u[i][j+1])*(v[i][j]+v[i+1][j])
                - (u[i-1][j]+u[i-1][j+1])*(v[i][j]+v[i-1][j]) )/dx
                + (v[i][j+1]*v[i][j+1]-v[i][j-1]*v[i][j-1])/2.0/dy )
                - dt/dy*(p[i][j+1]-p[i][j])
                + dt*1.0/Re*( (v[i+1][j]-2.0*v[i][j]+v[i-1][j])/dx/dx+(v[i][j+1]
                - 2.0*v[i][j]+v[i][j-1])/dy/dy );}
```

```
// B.C.
          for(j=1;j<=(n-2);j++){
               vn[0][j]=-vn[1][j];
               vn[n][j]=-vn[n-1][j];}

// B.C.
          for(i=0;i<=(n);i++){
               vn[i][0]=0.0;
               vn[i][n-1]=0.0;}
// ----------------------------------------------------------------------------------------
// FD equation number [3] see text on page number 3.
//continuity equation
for (i=1; i<=(n-1); i++)
for (j=1; j<=(n-1); j++){
pn[i][j]=p[i][j]-dt*delta*((un[i][j]-un[i-1][j])/dx+(vn[i][j]-vn[i][j-1])/dy);}

// B.C.
          for(i=0;i<=(n);i++){
               pn[i][0]=pn[i][1];
               pn[i][n]=pn[i][n-1];}

          for(j=0;j<=(n);j++){
               pn[0][j]=pn[1][j];
               un[n][j]=pn[n-1][j];}

// ----------------------------------------------------------------------------------------
err=0.0;

for (i=1; i<=(n-1); i++)
for (j=1; j<=(n-1); j++){
     m[i][j] = (  ( un[i][j]-un[i-1][j] )/dx + ( vn[i][j]-vn[i][j-1] )/dy  );
     err = err + fabs(m[i][j]);}

// residual[step] = log10(error);

if (d%1000==1){
          printf("Error is %5.5lf for the step %d\n", err, d);}

// Iterating u,v,p
for (i=0; i<=(n-1); i++)
for (j=0; j<=(n); j++){
     u[i][j] = un[i][j];}

          for(i=0;i<=(n);i++)
               for(j=0;j<=(n-1);j++){
                    v[i][j] = vn[i][j];}

          for(i=0;i<=(n);i++)
               for(j=0;j<=(n);j++){
                    p[i][j] = pn[i][j];}

d = d + 1;
t=t+dt;
```

```
//-----------------------------------------------------------------------
 Organizing the data to complete the tranfer.
//-----------------------------------------------------------------------
        if(d%100==1){
           for (i=1; i<=(n-1);i++){
               for (j=1; j<=(n-1);j++)
                myfile  << p[i][j] <<  '\n';}

        for (i=1; i<=(n-1);i++){
            for (j=1; j<=(n-1);j++)

        myfile2  << u[i][j] <<  '\n';}

        for (i=1; i<=(n-1);i++){
            for (j=1; j<=(n-1);j++)

        myfile3  << v[i][j]  <<  '\n';}

        myfile4 << err << '\n';}}

myfile.close();}
```

15 Conclusion

In this paper we investigated the Navier–Stokes equations considering the 2D scheme. We developed the numeric solution to the analytical differential equation and wrote it in a C++ code extracting the data of the problem and using Matlab graphical functions to display the rate of flow over time in our domain. Using spacial boundary conditions we managed to keep the rate of stream as if enclosed in a room.As the flow above the room is in constant speed, the walls of the cube remaining in the boarders of the domain. The matter will be defind by Reynolds number starting from 10 to high as 1,000,000. We have learned how to deal with complected numerical schemes and equations and how to simplify it to the software we use. To conclude the results received are satisfying to our desire and to the project demands with estimated error within our grid and time spatial while comparing them to the analytic solution(s).

our code is versatile, compact, fast and accurate. Looking are $Re = 100k$ you can see we are able to process and develop a large number of vortices. we are satisfied with our results and the imaging resulting.

Navier–Stokes theory is applied in many areas of specialization for example, in aerodynamic of moving objects to describe the physics of many phenomena of scientific and engineering interest. They may be used to model the weather, ocean currents, water flow in a pipe and air flow around a wing , neuro science (Ref Hodgkin-Huxley work on Action Potential Propagation), Calcium dynamics (Ref James Sneyd) As expected we got the results we thought we would.