Numerical Investigations in Computational Fluid Dynamics at IIT-M

Pratik Aghor and Ruchir Dwivedi

BITS Pilani

May-July 2014

Contents

1 The lid driven cavity problem

1.1 Assumptions

The fluid: Newtonian, incompressible and viscous.
Two dimensional flow.
The domain: Unit square domain. All walls except the top lid are rigid. The top lid is given a motion described by

$$u = Ucos(\omega t)$$

1.2 The Governing Equations and Boundary Conditions

1.2.1 Governing Equations

The set of equations governing the problem are 2d Navier-Stokes equations with incompressibility condition included. The continuity equation is

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

The 2 dimensional momentum equations are given by:
x-direction:

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

y-direction:

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

1.2.2 Boundary Conditions

$$u = U\cos(\omega t), v = 0\, at\, y = H$$

$$u = v = 0 at y = 0, x = 0 and x = L$$

1.3 Numerical Implementation

1.3.1 Weak Form

First of all, we derive the weak form of the above equations. The x and y direction components of the velocity are denoted by u1 and u2 respectively and those of test function v are denoted by v1 and v2 respectively.
We can write the x and y direction equations in the following vectorial form:(Where u now denotes a vector [u1,u2])

$$\frac{\partial u}{\partial t} + [u.\nabla]u = -\frac{1}{\rho}(\nabla p) + \nu(\triangle u)$$

Also the continuity equation can be written as :

$$\nabla.u = 0$$

To derive the weak form we define test functions v=[v1,v2] for velocity and q for pressure.
Multiply the first vector-equation by v and the continuity equation by q. Using Green's Formula we get the weak form as follows:

$$-\nu\int_\Omega(\nabla u.\nabla v)dx + \frac{1}{\rho}\int_\Omega p\nabla.vdx = \int_\Omega\frac{\partial u}{\partial t}vdx + \int_\Omega[u.\nabla]uvdx$$

$$\int_\Omega q\nabla.udx = 0$$

Introducing Bilinear forms a and b, the weak form can be written as

$$-(\nu a(u,v) + \frac{1}{\rho}b(v,p)) = \int_\Omega \frac{\partial u}{\partial t}vdx + \int_\Omega [u.\nabla]uvdx$$

$$b(u,q) = 0$$

where

$$a(u,v) = \int_\Omega (\nabla u.\nabla v)dx$$

and

$$b(v,p) = -\int_\Omega p\nabla.vdx$$

Therefore, the finite element problem statement becomes, find u^h *such that*
$-(\nu a(u_h, v_h) + \frac{1}{\rho}b(v_h, p_h)) = \int_\Omega \frac{\partial u}{\partial t}vdx + \int_\Omega [u.\nabla]uvdx$

$$b(u_h, q_h) = 0$$

1.3.2 Penalty Method

Penalty method replaces the above problem formulation by a more regular problem of finding $(u_h^\epsilon, p_h^\epsilon) - (\nu a(u_h^\epsilon, v_h^\epsilon) + \frac{1}{\rho}b(v_h^\epsilon, p_h^\epsilon)) = \int_\Omega \frac{\partial u}{\partial t}vdx + \int_\Omega [u.\nabla]uvdx$
$b(u_h^\epsilon, q_h^\epsilon) - \epsilon(p_h^\epsilon, q_h^\epsilon) = 0$
Formally, we have
$\nabla.u = \epsilon p_h^\epsilon$

1.3.3 Numerical Implementation

FreeFem++ source code for the lid driven cavity problem with periodically moving lid is as follows:

```
mesh Th=square(8,8);
plot(Th, wait=1, ps="oldmesh.jpg");
Th=splitmesh(Th,1+5*((x-0.5)^2+(y-0.5)^2));
plot(Th, wait=1, ps="newmesh.jpg");
fespace Xh(Th,P2);
fespace Mh(Th,P1);
Xh u2,v2;
Xh u1,v1;
Mh p,q;
Xh[int] xh(2);

xh[0] = x;
xh[1] = y;


cout << " nb of degree of freedom : " << Xh.ndof << endl;
cout << " nb of degree of freedom : " << Mh.ndof << endl;
real n=Xh.ndof;
Xh psi,phi;
//////////////To get streamlines, remove the comments from
//////////////the following block///////////////
solve streamlines(psi,phi) =
  int2d(Th)( dx(psi)*dx(phi) + dy(psi)*dy(phi))
+  int2d(Th)( -phi*(dy(u1)-dx(u2)))
```

```
+  on(1,2,3,4,psi=0);

////////////////////////////////////////////////////////////////////////////////

int i=0;
real  nu=1./100.;//nu is kinematic viscosity
real dt=0.1;     //time step
real alpha=1/dt;
real t=0;
real u1min,u1max,u2min,u2max;
real U=1;
real rho=1000;  //rho=density of fluid
real omega=2*pi/6;//omega= frequency oof oscillations of upper lid
for(t=0;t<6;t=t+dt)

{
Xh up1,up2;
///Weak Form for Navier-Stokes equations///
problem  NS ([u1,u2,p],[v1,v2,q],solver=GMRES,init=i) =
    int2d(Th)(
              alpha*( u1*v1 + u2*v2)
             + nu * ( dx(u1)*dx(v1) + dy(u1)*dy(v1)
             +  dx(u2)*dx(v2) + dy(u2)*dy(v2) )
             + p*q*(0.000001)
             -(1/rho)* p*dx(v1)-(1/rho)* p*dy(v2)
             + dx(u1)*q+ dy(u2)*q
            )
  + int2d(Th) ( -alpha*convect([up1,up2],-dt,up1)*v1
                -alpha*convect([up1,up2],-dt,up2)*v2 )
  + on(3,u1=U*cos(omega*t),u2=0)
                //include u=U*cos(omega*t) as a boundary condition
  + on(1,2,4,u1=0,u2=0)
;

for (i=0;i<=3;i++)
        {
        up1=u1;
        up2=u2;
        NS;
//solve N-S equations...update the values into convect operator
//using 'method of characteristics' and iterate to get good results
        } ;
/* Saves velocity, pressure and streamfunction to file */
ofstream f1("-velou"+t+".txt");
ofstream f2("-velov"+t+".txt");
ofstream f3("-press"+t+".txt");
ofstream f4("-sfunc"+t+".txt");
ofstream f5("u1_u2"+t+".txt");
real[int] Gx(n),Gy(n);

for (int j = 0; j < Xh.ndof; j++) {
    cout << j;

                Gx[j]=xh[0][][j];
                Gy[j]= xh[1][][j];
```

```
if(Gx[j]==0.5)
{
        f1 <<"y="<<Gy[j]<<"u1(t)="<< u1[][j]<<endl;
}
if(Gy[j]==0.5)
{
        f2 << "x="<<Gx[j]<<"u2(t)="<< u2[][j]<<endl;
}

//      f3 <<"p=" <<p[]<<endl;
        f4 <<"x="<<Gx[j] <<"y="<<Gy[j]<<"psi="<< psi[][j]<<endl;
//      f5 << "x="<<Gx[j]<<"y="<<Gy[j]<<"u1(t)="<< u1[][j]
//      <<"u2(t)="<< u2[][j]<<endl;


//      plot(coef=0.2,[u1,u2],fill=0,wait=1,ps="ufield(t)"+(t)+".jpg");
////Solve for streamlines if needed////
streamlines;
//plot(psi,wait=1,fill=0);

}
}
```

1.3.4 Splitmesh, adaptmesh

Adaptmesh:
If the function varies sharply in some regions, we must adapt the mesh in order to get better results. In FreeFem++, there is an inbuilt command 'adaptmesh' which adapts the mesh according to the requirements, all by itself.
(see example 5.4 in FreeFem++ manual by Hetch)

Splitmesh: If we want to split some part of the mesh triangles, we use splitmesh command in FreeFem++ The syntax used to split a mesh Th can be understood with the following example:

$$Th = splitmesh(Th, 1 + (x - 0.5)^2 + y^2)$$

The above statement asks Freefem++ to split the mesh as per the values given by the function:

$$1 = (x - 0.5)^2 + y^2$$

(See examples in FreeFem++ manual by Hetch)

1.3.5 Results

The results are as follows:

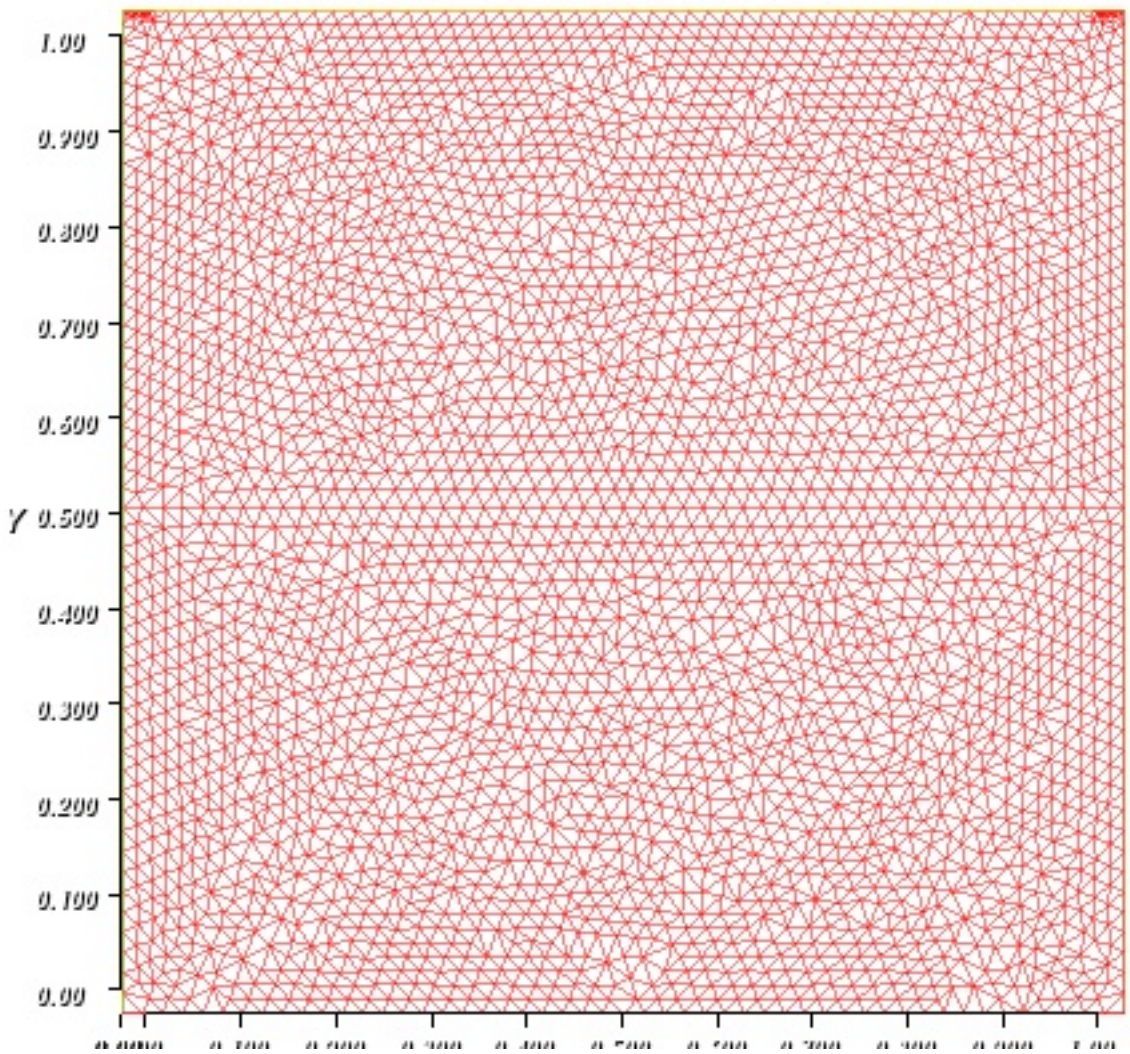

Figure 1.1: old mesh

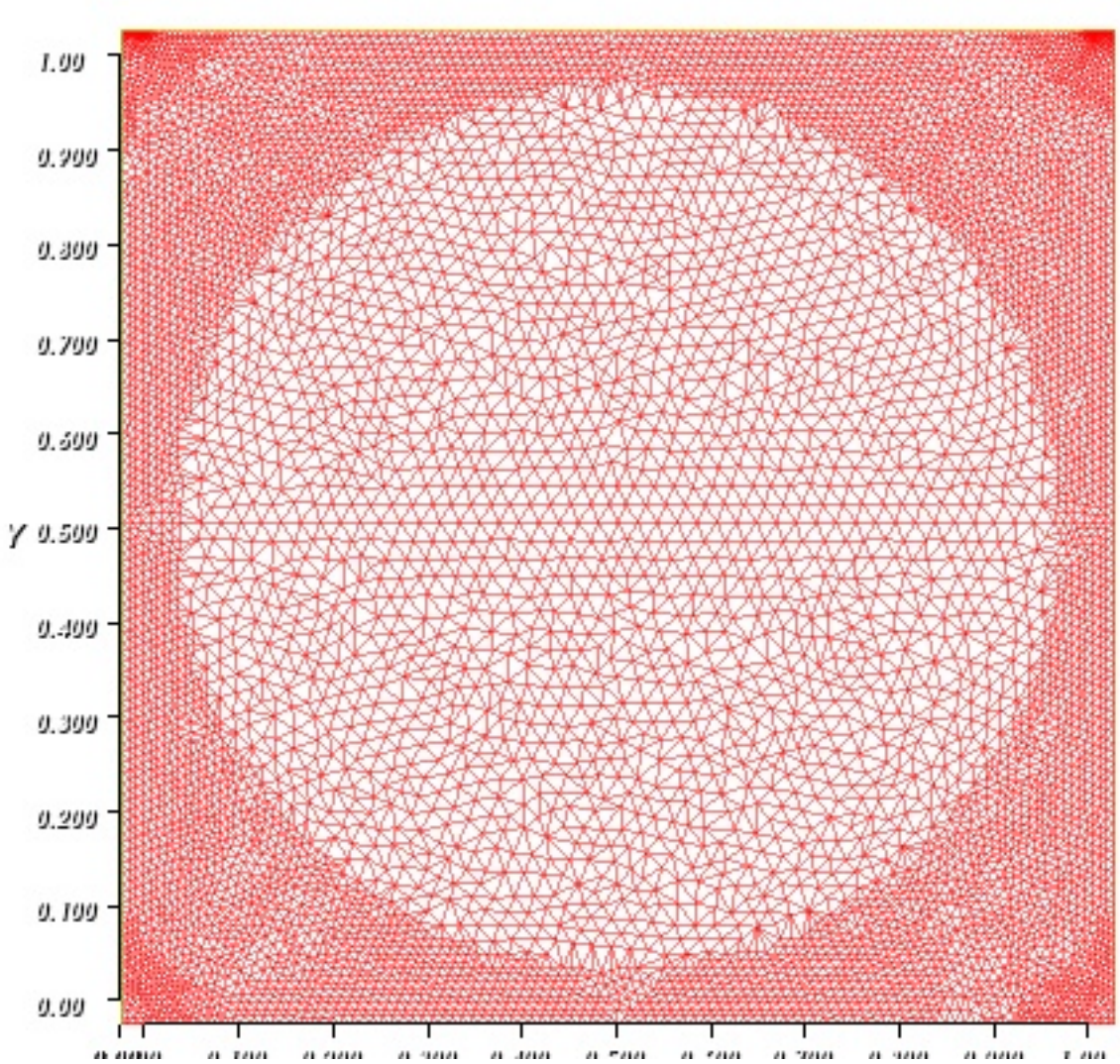

Figure 1.2: new mesh

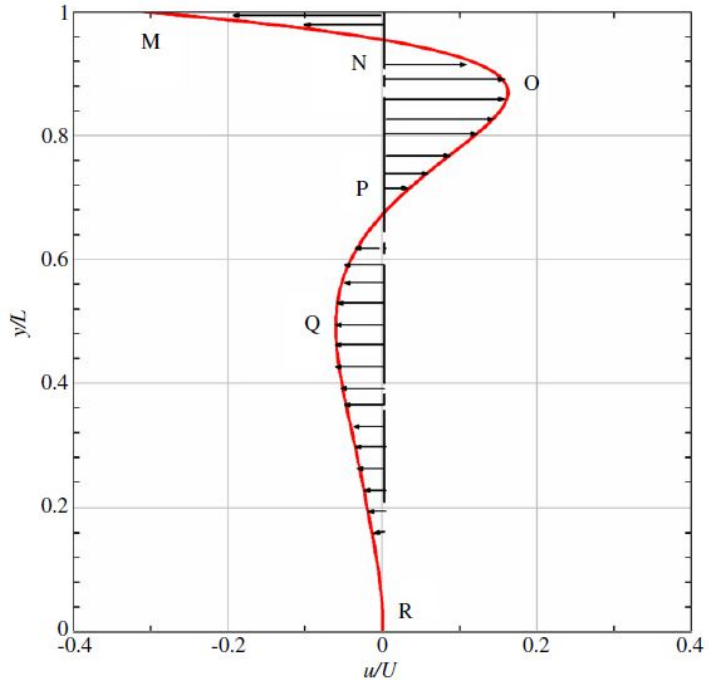

Fig. 8. The u-velocity profile along the vertical centerline of the cavity at $t = 0.3T$ for Re $= 100$ and $\omega = 2\pi/6$.

Figure 1.3: (1)y vs u at x=0.5 and t=0.3T=1.8 (Result from the paper)

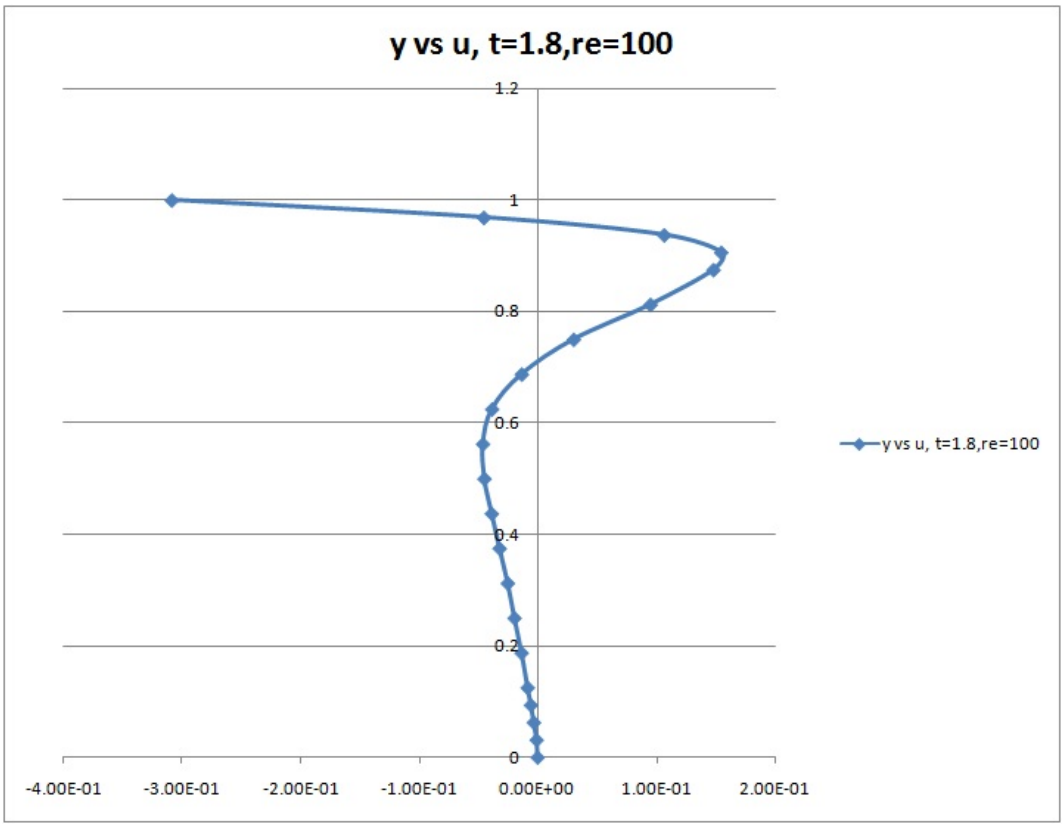

Figure 1.4: y vs u at x=0.5 and t=0.3T=1.8 (Our result)

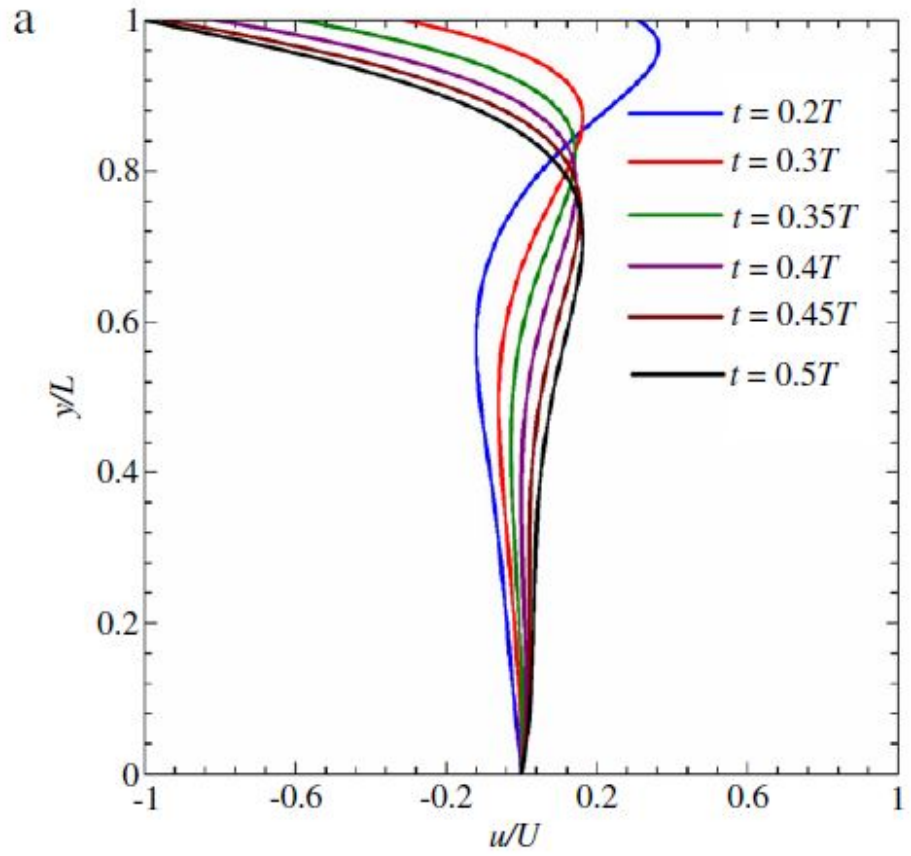

Figure 1.5: y vs u at x=0.5 (Result from the paper)

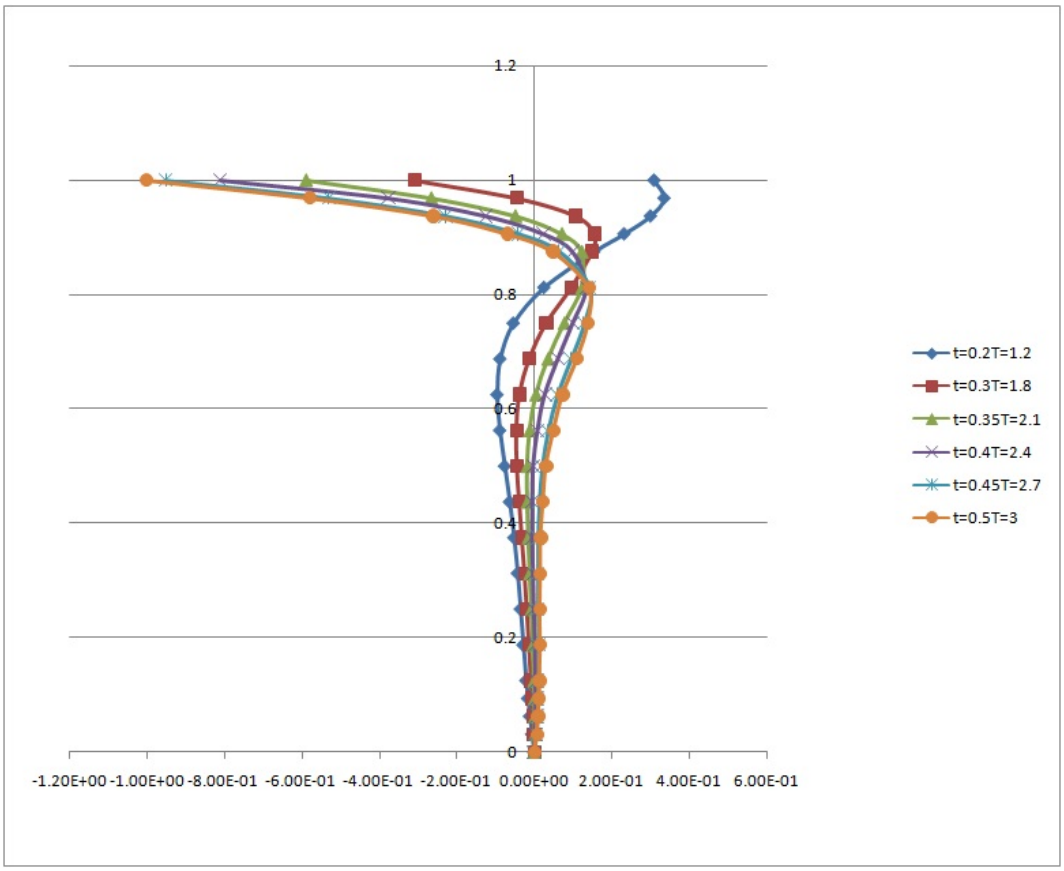

Figure 1.6: y vs u at x=0.5 (Our result)

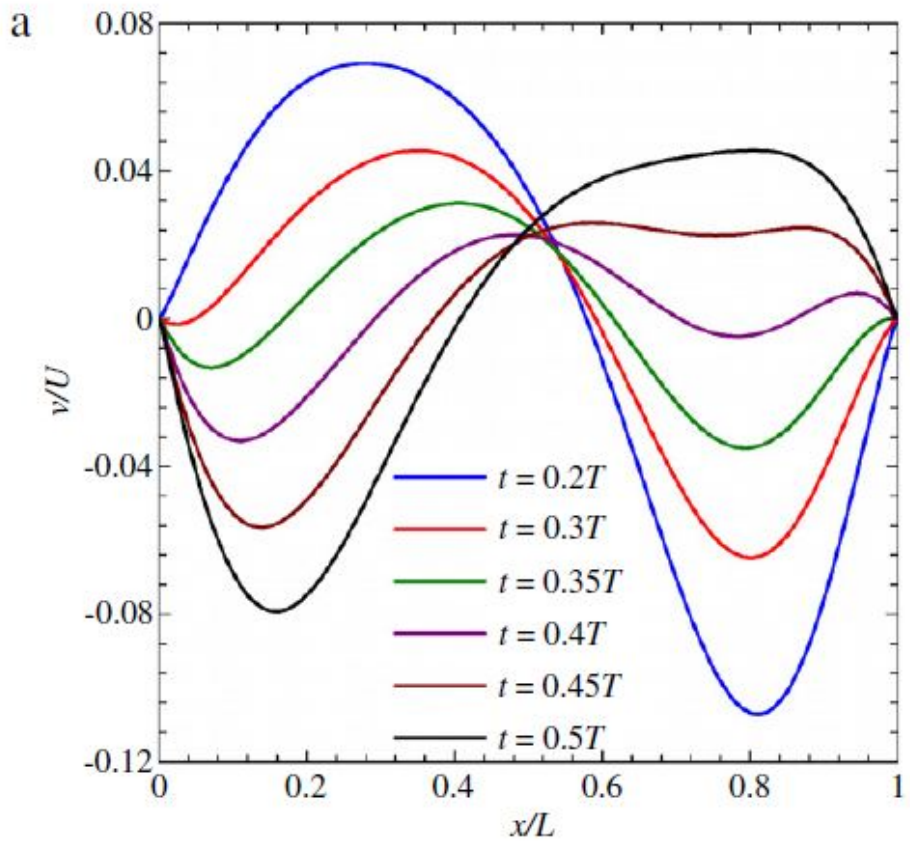

Figure 1.7: v vs x at y=0.5 (Result from the paper)

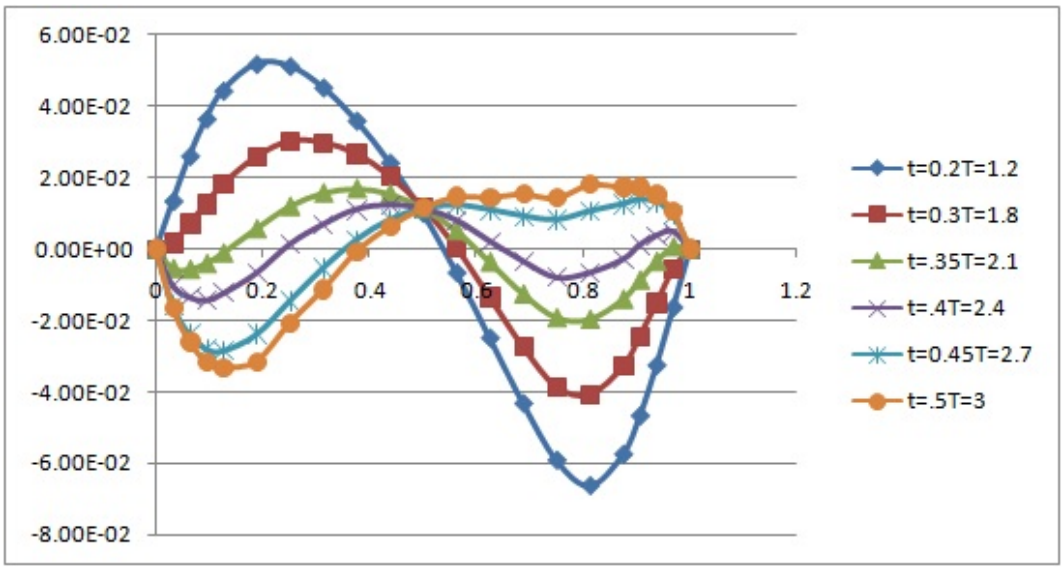

Figure 1.8: v vs x at y=0.5 (Our result)

2 Method of Characteristics

2.1 Mathematical Setting

The inhomogenous convection equation is given by

$$\frac{\partial q}{\partial t} + [u.\nabla]q = f$$

on Ω
$u(x,t)\,\epsilon R^d and\,\Omega\,\epsilon\,R^d$
This equation can be written as $\frac{Dq}{Dt} = f$
Let's introduce a variable such that,
$\frac{dq}{dt}(X(t),t) = f(X(t),t),\ \frac{dX(t)}{dt} = u(X(t),t)$
using time discretization,
$(q^{n+1}(x) - q^(n)(X^n(x)))/t^n = f^n(x)$
$where X^n(x)\,is\,an\,approximation\,of\,the\,solution\,at\,time\,t^n\,of\,the\,ordinary\,differential\,equation:$
$\frac{dX(t)}{dt} = u^{n(X(t))}, X(t^{n+1}) = x$
$where\,u^{n(x)} = u(x,t^n)$
$Using\,Taylor\,expansion,$
$q^n(X(t^n)) = q^n(X(t^{n+1})) - \triangle^n \sum_{i=1}^{d} \frac{\partial q}{\partial x_i}(X(t^n+1)\frac{\partial X_i}{\partial t}(t^n+1)$
$where X_i(t)\,are\,the\,ith\,components\,of\,X(t), q^n(x) = q(x,t^n)\,and\,X(t^{n+1})$
$q^n(X^(x)) = q^n(x) - \triangle t^n u^n(x)\nabla q^n(x) + o^(n)$
$applying\,Taylor\,expansion\,for \rightarrow q^n(x - u^n(x)t) + o^(n)$
$denoting\,the\,freefem++function$
$convect(u^n, -\triangle t^n, q^n) \approx q^n(x - u^n \triangle t^n)$
$we\,get\,the\,approximation,$
$q^n(X(x)) \approx convect(u^n, -\triangle t^n, q^n)$

3 Lid driven cavity problem for Bingham Fluids

3.1 Introduction

After successful completion of lid driven cavity problem for Newtonian fluids, we moved on to Bingham Fluids. Bingham fluids have a characteristic value of stress, commonly known as yield stress below which they behave as fluids while above which they are rigid.
We followed mixed finite element formulation.

3.1.1 Governing Equations and Boundary Conditions

$$Du = (\nabla u + \nabla u^T)/2$$

where Du is strain rate and u is velocity. Let

$$|Du| = (Du : Du)^{1/2}$$

be the Frobenius norm of Du.

That is to say,

$$|Du| = (4 * ((\frac{\partial u1}{\partial x})^2 + (\frac{\partial u1}{\partial y} + \frac{\partial u2}{\partial x})^2 + (\frac{\partial u2}{\partial y})^2))^{1/2}$$

Conservation of momentum in unsteady case gives

$$\nabla.\tau - \nabla p = f + \frac{D(u)}{Dt}$$

where

$$\frac{D(u)}{Dt} = [u.\nabla]u$$

and f comprises of external forces. u and p are unknowns of the problem.//

Du= $\begin{cases} 0 & \text{if } |\tau| \geq \tau_s \\ (1\text{-}(\tau_s/|\tau|))\tau/2\mu & \text{if } |\tau| < \tau_s \end{cases}$

where $\mu(> 0)$ is plastic viscosity and $\tau_s > 0$ is the yield stress.

These equations can be thought of as generalisation of unsteady problem having Ω_f a shear dependent viscosity

$$\hat{\mu} = 2\mu + \tau_s/|Du|)$$

To avoid singularities in —Du— we use regularised norm of Du.

$$|Du|_\epsilon = ((Du : Du) + (\epsilon)^2)^{1/2}$$

Hence the equations of motion take the following form:

$$\nabla.(2\mu + \tau_s/|Du|_\epsilon))Du - \nabla p = f + \frac{D(u)}{Dt}$$

along with

$$\nabla.u = 0$$

in Ω

A note should be made that fluid and rigid regions are unknwons a priorie and finding them is the part of the problem.

3.2 Weak Formulation

We used mixed formulation without any auxiliary tensor. We can also introduce an auxiliary tensor W and formulate the problem. We also have code for that formulation which will be included in the upcoming sections.

We define following bilinear forms:

$$a(u,v) = \int_\Omega 2\mu Du : Dv$$

on $H_0^1 X H_0^1$

$$b(p,v) = -\int_\Omega p(\nabla.v)$$

on $L_0^2 X H_0^1$
and

$$a_\epsilon(u,v) = a(u,v) + \int_\Omega (\tau_s/||Du||_\epsilon) Du : Dv$$

on $H_0^1 X H_0^1$

Therefore weak form for steady equation becomes

$$a_\epsilon(u,v) - b(p,v) + b(q,u) = (f,v)$$

3.3 Numerical Implementation

3.3.1 Approach 1: Mixed Finite Element Method without Auxiliary Tensor

The freefem++ source code that we wrote for mixed formulation is as follows:
This formulation did not work correctly as there was no provision in our code to identify the fluid and rigid regions. We also tried inserting auxiliary tensor 'W' and solve the same problem using mixed finite element method as described in (GIVE REFERENCE)

3.3.2 Approach 2: Mixed Finite Element Method with Auxiliary Tensor 'W'

The freefem++ source code for this formulation is as follows:

```
//border a(t=0,1){x=t;y=0;}
//border b(t=0,1){x=1;y=t;}
//border c1(t=1,0.96875){x=t;y=1;}
//border c2(t=0.96875,0.03125){x=t;y=1;}
//border c3(t=0.03125,0){x=t;y=1;}
//border d(t=1,0){x=0;y=t;}
//mesh Th=buildmesh(a(50)+b(50)+c1(50)+c2(50)+c3(50)+d(50));

mesh Th=square(8,8);
plot(Th,wait=1,ps="oldmesh.jpg");
Th=splitmesh(Th,1+5*((x-0.5)^2+(y-0.5)^2));
plot(Th,wait=1,ps="newmesh.jpg");
fespace Mh(Th,P1);
Mh p,q;
fespace Xh(Th,P2);
Xh up1,up2,u1,u2,v1,v2;
macro div(u) ( dx(u#1)+dy(u#2) ) //
macro StrainRate(u) [2.*dx(u#1), 2.*dy(u#2), dy(u#1)+dx(u#2)] //
macro W [Wxx,Wyy,Wxy]//
```

```
macro Z [Zxx,Zyy,Zxy]//

Xh Wxx,Wyy,Wxy,Zxx,Zyy,Zxy;
real n=Xh.ndof;

Xh[int] xh(2);
xh[0] = x;
xh[1] = y;

real eps=1e-5;
real  nu=1./1000.;
real dt=0.1;
real alpha=1/dt;
real t=0;
real U=1;
real rho=1000;
real omega=2*pi/6;
real a1=1/32;

func f1=sin(pi*x/2*a1);
func f2=1;
func f3=sin(pi*(1-x)/2*a1);
int i=0,j=0,iter=0;
for(int k=0;k<4;k++)
{
real taus=10^(k);
for(iter=0;iter<5;iter++)
{
real H1norm=(2* sqrt(( dx(u1)^2 +.5* (dy(u1) + dx(u2))^2 + dy(u2)^2 )+eps^2 ));//
Xh psi,phi;
//////////////To get streamlines, remove the comments from
//////////////the following block///////////////
problem streamlines(psi,phi) =
  int2d(Th)( dx(psi)*dx(phi) + dy(psi)*dy(phi))
+  int2d(Th)( -phi*(dy(u1)-dx(u2)))
//+  on(a,b,d,c1,c2,c3,psi=0);
+on(1,2,3,4,psi=0);

////////////////////////////////////////////////////////////////////////////
problem NSf ([u1,u2,p,Wxx,Wyy,Wxy],[v1,v2,q,Zxx,Zyy,Zxy]) =
    int2d(Th)(
          alpha*( u1*v1 + u2*v2)
        + 2*nu*StrainRate(u)'*StrainRate(v)
        +taus*(1/rho)*StrainRate(v)'*W
        -taus*(1/rho)*StrainRate(u)'*Z
        -(1/rho)*p*div(v)
        - q*div(u)
        -(1e-5)*p*q
        +(H1norm)*W'*Z
           )
  + int2d(Th) ( -alpha*convect([up1,up2],-dt,up1)*v1
                -alpha*convect([up1,up2],-dt,up2)*v2 )
 // + on(c1,u1=f1  ,u2=0) + on(c2,u1=f2  ,u2=0)+ on(c3,u1=f3,u2=0)
                //include u=U*cos(omega*t) as a boundary condition
  //+ on(a,b,d,u1=0,u2=0)
```

```
        +on(3,u1=1,u2=0)+on(1,2,4,u1=0,u2=0)
;
////////////////////////////////////////////////////////////////////////////////
problem   NSr ([u1,u2,p,Wxx,Wyy,Wxy],[v1,v2,q,Zxx,Zyy,Zxy]) =
    int2d(Th)(
          alpha*( u1*v1 + u2*v2)
//       + 2*nu*StrainRate(u)'*StrainRate(v)
//       +taus*(1/rho)*(1/H1norm)*StrainRate(v)'*W
//       -taus*(1/rho)*(1/H1norm)*StrainRate(u)'*Z
         -(1/rho)*p*div(v)
         - q*div(u)
         -(1e-5)*p*q
         +(H1norm)*W'*Z
            )
//  + int2d(Th) ( -alpha*convect([up1,up2],-dt,up1)*v1
//                -alpha*convect([up1,up2],-dt,up2)*v2 )
//  + on(c1,u1=f1  ,u2=0) + on(c2,u1=f2  ,u2=0)+ on(c3,u1=f3,u2=0)
                //include u=U*cos(omega*t) as a boundary condition
 // + on(a,b,d,u1=0,u2=0)
        +on(3,u1=1,u2=0)+on(1,2,4,u1=0,u2=0)

;


for (i=0;i<=3;i++)
        {
        up1=u1;
        up2=u2;

        NSf;
        streamlines;
        }

//solve N-S equations...update the values into convect operator
//using 'method of characteristics' and iterate to get good results
////Solve for streamlines if needed////
plot(psi,fill=0,wait=1,ps="psi(taus)"+((taus))+".jpg");
ofstream ff1("velou.txt");
ofstream ff2("velov.txt");
ofstream ff3("u1x.txt");
ofstream ff4("u1y.txt");
ofstream ff5("u2x.txt");
ofstream ff6("u2y.txt");
real[int] Gx(n),Gy(n),u1x(n),u1y(n),u2x(n),u2y(n);

for (int j = 0; j < Xh.ndof; j++) {
    cout << j;

                Gx[j]=xh[0][][j];
                Gy[j]= xh[1][][j];
                ff1 <<"x="<<Gx[j]<<"y="<<Gy[j]<<"u1="<< u1[][j]<<endl;
                ff2<<"x="<<Gx[j]<<"y="<<Gy[j]<<"u2="<<u2[][j]<<endl;
                ff3<<"x="<<Gx[j]<<"y="<<Gy[j]<<"u1x="<<endl;
                ff4<<"x="<<Gx[j]<<"y="<<Gy[j]<<"u1y="<<dy(u1)<<endl;
                                }
```

```
}

}
```

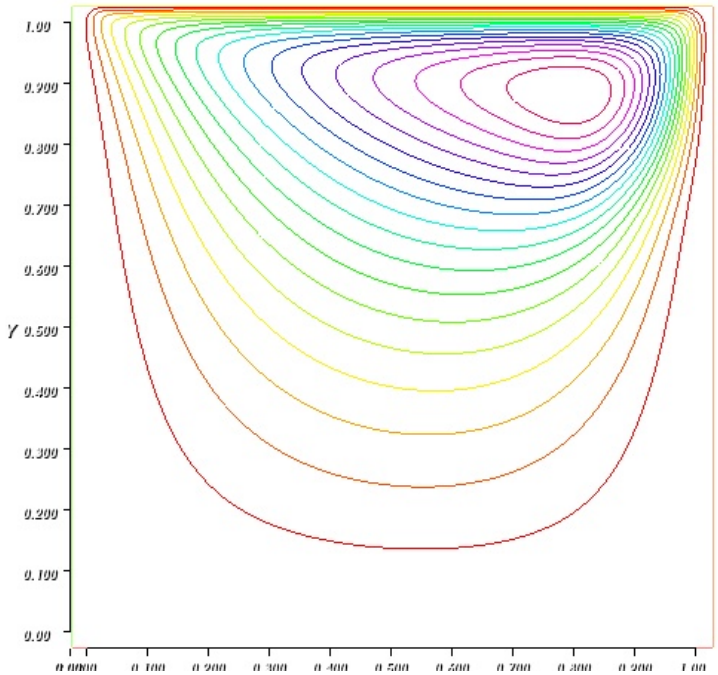

Figure 3.1: Auxiliary Tensor Approach (taus=0)

The streamline plots for different values of yield stress are as follows:Which do not match with the published data.

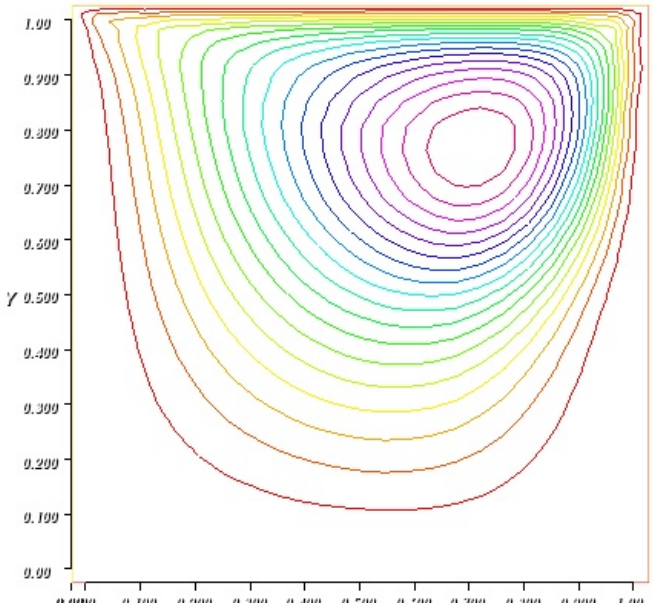

Figure 3.2: Auxiliary Tensor Approach (taus=1)

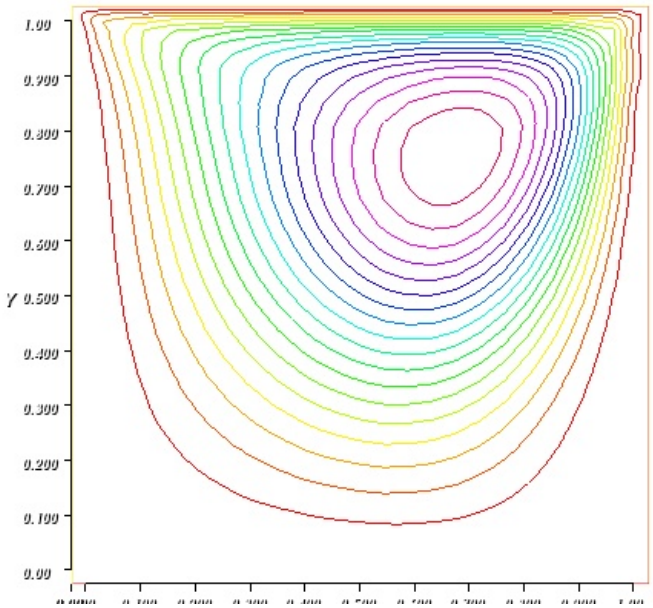

Figure 3.3: Auxiliary Tensor Approach (taus=10)

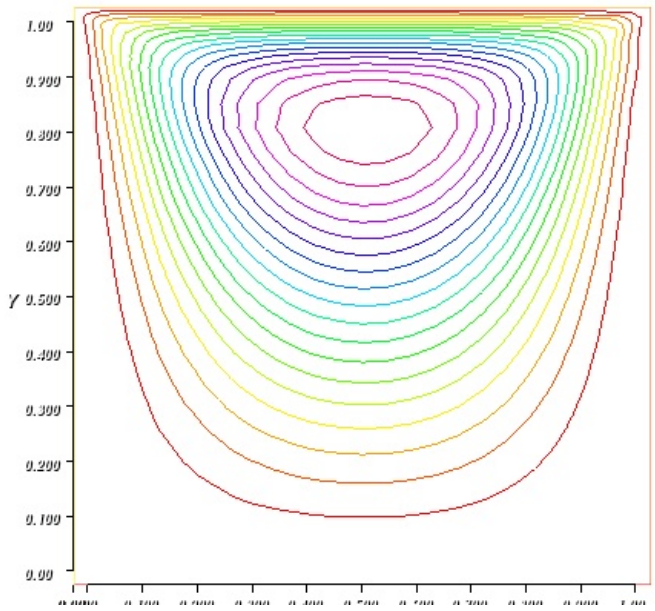

Figure 3.4: Auxiliary Tensor Approach (taus=100)

3.3.3 Approach 3: Lions-Glowinski Augmented Formulation

So, finally, we went for another approach. We tried Uzawa scheme for the Lions-Glowinski augmented formulation which uses Lagrange multipliers. (GIVE TWO REFERENCES Paper and bingham programme). We learned a lot from the following link: http://www.um.es/freefem/ff++/uploads/Main/Bingham.pdf

We modified the code available for Stokes problem for Bingham fluids into our formulation.
The freefem++ source code is as follows:

```
real taus = 10;
real dPdL = -9.6;
real r = 10.;

macro div(u) ( dx(ux)+dy(uy) ) //
macro StrainRate(u) [2.*dx(ux), 2.*dy(uy), dy(ux)+dx(uy)] //

mesh Th = square(10,10);
fespace Qh(Th,P1);
fespace Xh(Th,P2);
fespace Lh(Th,P1dc);
Lh TmGxx=0., TmGyy=0., TmGxy=0.;
Lh Txx=0., Tyy=0., Txy=0.;
Lh Gdotxx=0., Gdotyy=0., Gdotxy=0.;
Xh ux,uy,vx,vy,psi,phi,up1,up2;
Qh p,q;
real dt=0.1;
real alpha=1/dt;
real rho=1;

problem NS([ux,uy,p],[vx,vy,q],init=true) =

 int2d(Th)(
         alpha*( ux*vx + uy*vy))
+
int2d(Th)(   (1/rho)*r*(StrainRate(u)'*StrainRate(v))-(1/rho)*p*div(v) -q*div(u) +1e-
 // +int2d(Th)( dPdL*vx )
```

```
        +int2d(Th)(  [TmGxx,TmGyy,TmGxy]'*[dx(vx),dy(vy),dy(vx)+dx(vy)]  )

 + int2d(Th) ( -alpha*convect([up1,up2],-dt,up1)*vx
               -alpha*convect([up1,up2],-dt,up2)*vy  )

        +on(3,ux=1,uy=0)+on(1,2,4,ux=0,uy=0);


problem streamlines(psi,phi) =
  int2d(Th)( dx(psi)*dx(phi) + dy(psi)*dy(phi))
+  int2d(Th)( -phi*(dy(ux)-dx(uy)))
+  on(1,2,3,4,psi=0);

for(real t=0;t<5;t=t+dt)
{
for(int iter=0; iter<100; ++iter)
{
        for(int i=0;i<3;i++)
        {
        up1=ux;
        up2=uy;
        NS;
        }
        Gdotxx = 2.*dx(ux);
        Gdotyy = 2.*dy(uy);
        Gdotxy = dx(uy)+dy(ux);

        for(int k=0; k<Lh.ndof; ++k){
                real TaGxx=Txx[][k]+r*Gdotxx[][k],
                     TaGyy=Tyy[][k]+r*Gdotyy[][k],
                     TaGxy=Txy[][k]+r*Gdotxy[][k];
                real TaGnorm = sqrt( .5*(TaGxx^2+TaGyy^2+2.*TaGxy^2) );
                real res = TaGnorm-taus;
                if( 0<res ){
                        real c = r/(1.+r)*res/TaGnorm;
                        real q = 1.-c;
                        Txx[][k] = TaGxx*q;
                        Tyy[][k] = TaGyy*q;
                        Txy[][k] = TaGxy*q;

                        q = 1.-2.*c;
                        TmGxx[][k] = TaGxx*q;
                        TmGyy[][k] = TaGyy*q;
                        TmGxy[][k] = TaGxy*q;


                }
                else {
                        Txx[][k] = TaGxx;
                        Tyy[][k] = TaGyy;
                        Txy[][k] = TaGxy;

                        TmGxx[][k] = TaGxx;
                        TmGyy[][k] = TaGyy;
                        TmGxy[][k] = TaGxy;
```

```
                                }
        }
}
//plot(ux,dim=2,fill=true,value=true);
streamlines;
plot(psi,fill=0,wait=1,ps="psi(taus)"+(taus)+"(t)"+t+".jpg");

}
```

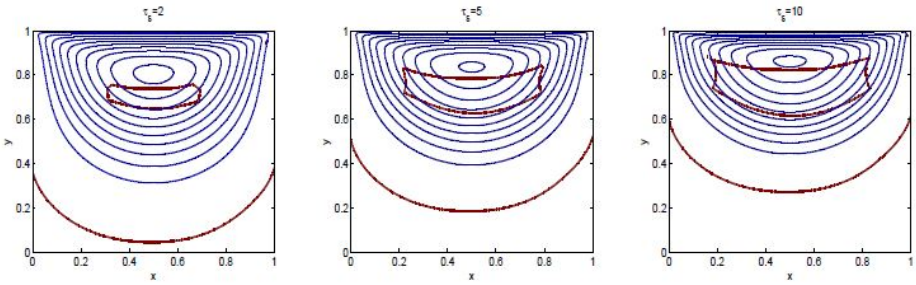

Figure 3.5: published results

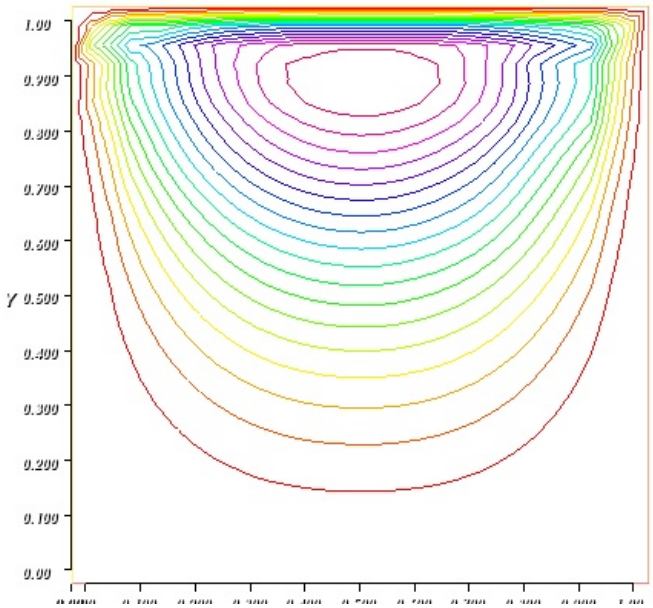

Figure 3.6: Lions-Glowinski Augmented Formulation (taus=2)

The comparison of streamline plots was done and we got fairly good matches.

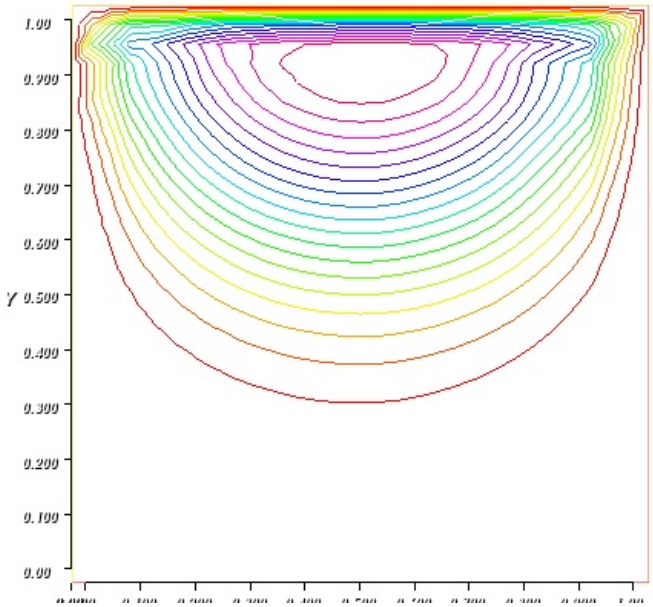

Figure 3.7: Lions-Glowinski Augmented Formulation (taus=5)

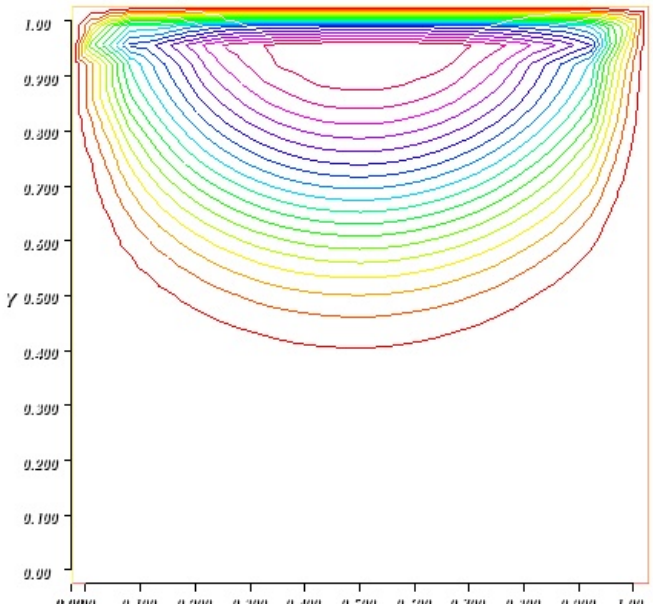

Figure 3.8: Lions-Glowinski Augmented Formulation (taus=10)

4 Lid Driven Cavity Problem for Carreau Fluid

4.1 Introduction

In case of Carreau fluids, the constitutive equation is different. Otherwise, the main formulation is similar. Mixed Finite Element formulation was used (GIVE REFERENCE) and freefem++ source code is given below:

```
//border a(t=0,1){x=t;y=0;}
//border b(t=0,1){x=1;y=t;}
//border c1(t=1,0.96875){x=t;y=1;}
//border c2(t=0.96875,0.03125){x=t;y=1;}
//border c3(t=0.03125,0){x=t;y=1;}
//border d(t=1,0){x=0;y=t;}
//mesh Th=buildmesh(a(45)+b(40)+c1(27)+c2(27)+c3(27)+d(40));

mesh Th=square(20,20);
plot(Th,wait=1,ps="oldmesh.jpg");
Th=splitmesh(Th,1+5*((x-0.5)^2+(y-0.5)^2));
plot(Th,wait=1,ps="newmesh.jpg");
fespace Xh(Th,P2);
fespace Mh(Th,P1);
Xh u2,v2,up1,up2;
Xh u1,v1;
Mh p,q;
Xh[int] xh(2);
macro div(u) ( dx(u#1)+dy(u#2) ) //
macro StrainRate(u) [2.*dx(u#1), 2.*dy(u#2), dy(u#1)+dx(u#2)] //
xh[0] = x;
xh[1] = y;
real n=Xh.ndof;
real eps=1e-5;
Xh psi,phi;
real H1norm=( sqrt(4*( dx(u1)^2 +.5* (dy(u1) + dx(u2))^2 + dy(u2)^2 )+ eps^2));
real eta,eta0,eta1;
real lambda=1,n1=0.5,a2=1;
eta0=0.004*(2.17^4);
eta1=0.004;
eta=eta1+((eta0-eta1)/(1+(H1norm*lambda)^a2)^((1-n1)/a2));

//////////////To get streamlines, remove the comments from
//////////////the following block///////////////
solve streamlines(psi,phi,solver=GMRES) =
  int2d(Th)( dx(psi)*dx(phi) + dy(psi)*dy(phi))
+  int2d(Th)( -phi*(dy(u1)-dx(u2)))
+  on(1,2,3,4,psi=0)
//+on(a,b,c1,c2,c3,d,psi=0);
;
////////////////////////////////////////////////////
int i=0;
//real  nu=1./1000.;//nu is kinematic viscosity
```

```
real dt=0.1;     //time step
real alpha=1/dt;
real t=0;
real U=0.0016;
real rho=1000;  //rho=density of fluid
//real omega=2*pi/6;//omega= frequency of oscillations of upper lid
real a1=1/32;
func f1=U*sin(pi*x/2*a1);
func f2=U;
func f3=U*sin(pi*(1-x)/2*a1);
problem   NSf ([u1,u2,p],[v1,v2,q]) =
    int2d(Th)(
              alpha*( u1*v1 + u2*v2)
//       + 2*(eta/rho)*StrainRate(u)'*StrainRate(v)
         +(1/rho)*eta*(2*dx(u1)*dx(v1)+
         dy(u1)*dx(v2)+dx(u2)*dy(v1)+dx(u2)*dx(v2)+dy(u1)*dy(v1)+
         2*dy(u2)*dy(v2))
         -(1/rho)*p*div(v)
         - q*div(u)
         -(1e-5)*p*q
            )
//       +int1d(Th,c1,c2,c3)(v1*eta*(dy(u1)+dx(u2))+v2*(eta*2.*dy(u2)-p))
         +int1d(Th,3)(v1*eta*(dy(u1)+dx(u2))+v2*(eta*2.*dy(u2)-p))

           +int2d(Th)
                 ( -alpha*convect([up1,up2],-dt,up1)*v1
                 -alpha*convect([up1,up2],-dt,up2)*v2 )
  //+ on(c1,u1=f1  ,u2=0) + on(c2,u1=f2  ,u2=0)+ on(c3,u1=f3,u2=0)
                 //include u=U*cos(omega*t) as a boundary condition
//  + on(a,b,d,u1=0,u2=0)
        +on(3,u1=1,u2=0)
         + on(1,2,4,u1=0,u2=0)
;
////////////////////////////////////////////////////////////////////////////

for(real t=0;t<6;t=t+dt)
{
for (i=0;i<=5;i++)
         {
         up1=u1;
         up2=u2;
         NSf;

         }
         streamlines;
         ofstream ff1("u1(m)(t)"+(n1)+(t)+".txt");
         ofstream ff2("u2(m)(t)"+(n1)+(t)+".txt");
//ofstream ff3("-press"+t+".txt");
//ofstream ff4("-sfunc"+t+".txt");
real[int] Gx(n),Gy(n);

for (int j = 0; j < Xh.ndof; j++) {
    cout << j;

                 Gx[j]=xh[0][][j];
```

```
                Gy[j]= xh[1][][j];

        if(Gx[j]==0.5)
        {
        ff1 <<"y="<<Gy[j]<<" "<<"u1(t)="<< (u1[][j])/U<<endl;
        }
        if(Gy[j]==0.5)
        {
        ff2 << "x="<<Gx[j]<<" "<<"u2(t)="<< (u2[][j])/U<<endl;
        }
//      f3 <<"x="<<Gx[j] <<"y="<<Gy[j]<<"p=" <<p[][j]<<endl;
//      ff4 <<"x="<<Gx[j] <<"y="<<Gy[j]<<"psi="<< psi[][j]<<endl;
                                        }
        plot(psi,fill=0,wait=1,ps="psi(t)(m)"+(t)+(n1)+".jpg");
}
        /* Saves velocity, pressure and streamfunction to file */

////////////////////////////////////////////////
```

5 Lid Driven Cavity Problem for Oldroyd-b Model

5.1 Governing Equations and Boundary Conditions

The governing equations and boundary conditions for Oldroyd-b model are as follows:

$$-\nabla p + \mu \triangle u + (\frac{\eta}{\tau})\nabla .c = \frac{Du}{Dt}$$

$in \Omega X(0,T)$

$$\nabla .u = 0$$

$$\frac{\partial c}{\partial t} + (u.\nabla)c - [(\nabla u)c + c(\nabla u)^T] = \frac{1}{\tau}(I - c)$$

$in \Omega X(0,T)$

$$c(0) = c_0$$

$in \Omega$

$$u = g(t) on \Gamma X(0,T)$$

with $\int (g(t).nd\Gamma) = 0$ on $(0,T)$

Here, u and p are unknown velocity and pressure fields.
c is conformation tensor.
μ and η are solvent and polymer viscosities.
n is outward normal vector at boundary Γ.
g(t)$ischosenasgivenbyFattalandKupferman$[2005] :

$$g(t) = \begin{cases} (\text{g(x,t),0})^T & \text{if } on\text{x—x=(x,1)}^T, 0 < x < 1 \\ 0 & \text{otherwise } on\Gamma \end{cases}$$

$where$
$g(x,t) = 8(1 + \tanh 8(t - 0.5))x^2(1 - x)^2$

Wiessenberg no. is given by $Wi = \frac{\tau U}{L}$