Finite Elements for Fluids
Prof. Sheu

Yannick Deleuze*†

http://homepage.ntu.edu.tw/~ydeleuze/ff2016/

Taipei, April 28, 2016

1 Homework from April 7, 2016

Use the code from the exercices. Write a small report with comments and discussion on the results and provide the code used.

- Convection-diffusion equation

$$\begin{aligned} \nabla \cdot (\mathbf{u}\phi) - \nabla \cdot (\nu \nabla \phi) = 0 \quad & \text{in } [0,1] \times [0,1], \\ \phi = 1 \quad & \text{on } \Gamma_1 \cup \Gamma_4 \\ \phi = 0 \quad & \text{on } \Gamma_2 \cup \Gamma_3 \end{aligned} \tag{1}$$

  with $\mathbf{u} = (-u_{max}, -u_{max})$.

  1. Write the variational formulation of the convection-diffusion equation.
  2. Plot the solution for different value of $h = \frac{1}{N_K} \sum_K h_k$ of the convection-diffusion equation using the non-stabilized method, the artificial viscosity method and the SUPG methods in the case $Pe = \max_K(Pe_K) < 1$ and $Pe > 1$.
  3. In the case, $Pe < 1$, compare the solutions from each stabilized method with the non-stabilized method using the H^1 norm.

- Stokes equation

  1. Write the variational formulation of the Stokes equations

$$\begin{aligned} -\frac{1}{\mathrm{Re}} \nabla^2 \mathbf{u} + \nabla p = \mathbf{f} \quad & \text{in } [0, L] \times [0,1], \\ \nabla \cdot \mathbf{u} + \varepsilon p = 0, & \\ \mathbf{u} = (y(1-y), 0) \quad & \text{on } \Gamma_1, \\ \mathbf{u} = (0,0) \quad & \text{on } \Gamma_{2-4}, \\ -\frac{1}{\mathrm{Re}} \boldsymbol{\nabla \mathbf{u}} \cdot \mathbf{n} + p \cdot \mathbf{n} = (0,0) \quad & \text{on } \Gamma_3. \end{aligned} \tag{2}$$

  2. Analytical validation in the square $[0,1] \times [0,1]$ with the $||\cdot||_V$ norm according to the analytical solution

$$u_1 = y(1-y),\ u_2 = 0,\ p = \frac{2}{\mathrm{Re}}(1-x),\ f = 0.$$

     Plot the error for different values of $h = \max_K h_K$ for the P2-P1, P1b-P1, and P1-P1 elements.
  3. Solve the Stokes equation in the domain of your choice with the boundary condition of your choice. Give the variational formulation and a plot of the solution.

*Department of Engineering Science and Ocean Engineering, National Taiwan University
†ydeleuze@ntu.edu.tw

2 Correction

2.1 Convection-diffusion equation

Let $V_\varphi = \{w \in H^1(\Omega) | w = \varphi \text{ on } \Gamma_1 \cup \Gamma_4, w = 0 \text{ on } \Gamma_2 \cup \Gamma_3\}$. One multiplies (1) by a test function $v \in V_0$ and integrate over the entire domain. The variational formulation of the convection-diffusion problem becomes

find $\phi \in V_1$ such that $\forall v \in V_0$:

$$a(\phi, v) = 0 \tag{3}$$

where the bilinear form

$$a(\phi, v) = \int_\Omega -\phi\, \mathbf{u} \cdot \nabla v + \nu \nabla \phi \cdot \nabla v.$$

2.1.1 Solutions for $Pe < 1$ and $Pe > 1$

Let $\nu = 1$ and $U_{inf} = 150$.

Solutions for $Pe = 5.80024$ The usual Galerkin method (non-stabilized) solution features a oscillations. From figure 1, one can observe that the artificial viscosity and SUPG method can give stabilized solutions without oscillation. From the cuts along the x and y directions in figure 1 (e)-(f), one can see that the SUPG (2) method give the better approximation.

Solutions for $Pe = 0.820483$ The usual Galerkin method (non-stabilized) solution give the best approximation. From the cuts along the x and y directions in figure 2 (e)-(f), one can see that the SUPG (2) method give the approximation with less artificial viscosity.

2.1.2 Error for SUPG and artificial methods when $Pe < 1$

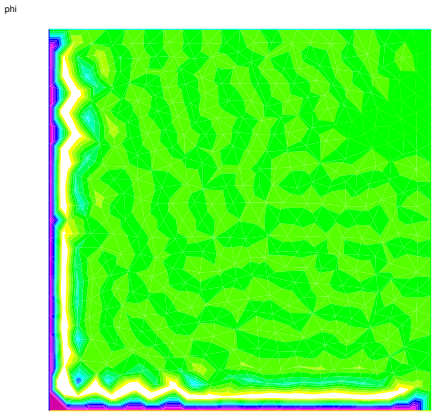
Please see figure 2 for the comparison of the solution obtained by the different methods. The error between the artificial and SUPG methods and the the Galerkin (non-stabilized) is given in figure 3. The difference methods are of order 1 when using P1 finite elements. One can see that the SUPG (2) method give the approximation with less artificial viscosity.

2.1.3 FreeFem++ code

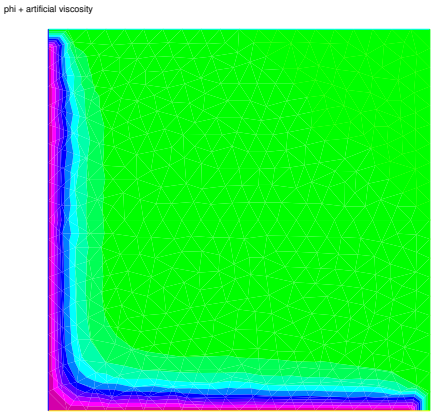
```
/*       Generating Mesh */
real L=1;
border b1(t=1,0)          {x=0;          y=t;          label=4;}
border b2(t=0,L)          {x=t;          y=0;          label=1;}
border b3(t=0,1)          {x=L;          y=t;          label=2;}
border b4(t=L,0)          {x=t;          y=1;          label=3;}
real Dx=0.001, Dy=0.001;
mesh Th= buildmesh(b1(ceil(1./Dy))+b2(ceil(L/Dx))+b3(ceil(1./Dy))+b4(ceil(L/Dx)));
plot(Th, wait=1);

/*       Finite Element Spaces */
fespace Vh(Th,P1);
fespace Xh(Th,P0);

/*       Problem parameters       */
real nu=1.;
Vh u=-10, v=-10;
real Uinf = max(u[].linfty,v[].linfty);
real U2   = sqrt(u^2+v^2);
plot([u,v], wait=1, cmm=" flow field [u,v], |U|inf="+Uinf,coef=0.1);
real [int] viso(-0.3:0.1:1.); //define fixed isolines

/*       Macros  */
macro a(phih,vh) int2d(Th)(nu * ( dx(phih)*dx(vh) + dy(phih)*dy(vh) ) + (u * dx(phih) + v
    * dy(phih) ) * vh) //eom

/*       Defining the variational problem         */
varf CD (phi,w) =
          a(phi,w)
        +on(1,4,phi=1)+on(2,3,phi=0);

```

```
varf CDstab(phi,w) =
    a(phi,w)
  + int2d(Th)( Uinf*hTriangle*(dx(phi)*dx(w) + dy(phi)*dy(w))) //artifical viscosity
  + on(1,4,phi=1)+on(2,3,phi=0);

real delta = 1.0;
Xh tau = hTriangle/U2;
varf CDsupg(phi,w) =
    a(phi,w)
  + int2d(Th)( delta*(-nu*(dxx(phi)+dyy(phi))+u*dx(phi)+v*dy(phi)) //SUPG
    *tau*(u*dx(w) + v*dy(w)) )
  +on(1,4,phi=1)+on(2,3,phi=0);

/*       Peclet Number    */
Xh hK = hTriangle;
Xh PeK = Uinf * hK / 2. / nu;
real Pe = PeK[].max;
cout << "Pe = " + Pe << endl;
plot(PeK,wait=1,fill=1,value=1,cmm="Peclet number : Pe="+Pe, ps="PeK"+Pe+".eps");

/*       Solving the problems     */
Vh phi;
matrix A = CD(Vh,Vh,solver = UMFPACK, factorize=0); // assemble the FE matrix
real[int] b = CD(0,Vh); // Dirichlet boundary condition
phi[] = A^-1*b; //solve the problem
plot(phi, wait=1, fill=1, cmm=" phi ",viso=viso, ps="phi"+Pe+".eps");

Vh phistab;
A = CDstab(Vh,Vh,solver = UMFPACK, factorize=0); // assemble the FE matrix
b = CDstab(0,Vh); // Dirichlet boundary condition
phistab[] = A^-1*b; //solve the problem
plot(phistab, wait=1,fill=1, cmm=" phi + artificial viscosity",viso=viso, ps="artificial
    viscosity"+Pe+".eps");

Vh phisupg;
A = CDsupg(Vh,Vh,solver = UMFPACK, factorize=0); // assemble the FE matrix
b = CDsupg(0,Vh); // Dirichlet boundary condition
phisupg[] = A^-1*b; //solve the problem
plot(phisupg, wait=1,fill=1, cmm=" phi + SUPG",viso=viso, ps="SUPG"+Pe+".eps");

tau = hTriangle*nu/2./U2;
Vh phisupg2;
A = CDsupg(Vh,Vh,solver = UMFPACK, factorize=0); // assemble the FE matrix
b = CDsupg(0,Vh); // Dirichlet boundary condition
phisupg2[] = A^-1*b; //solve the problem
plot(phisupg2, wait=1,fill=1, cmm=" phi + SUPG (2)",viso=viso, ps="SUPG2"+Pe+".eps");


/*       Compare the different solutions */
//h1 errors
macro H1Norm(u) (sqrt( int2d(Th)(u^2 + (dx(u))^2 + (dy(u))^2 ) ) ) //eom
{
Vh errstab = phi - phistab;
real errstabH1 = H1Norm(errstab);
Vh errSUPG = phi - phisupg;
real errSUPGH1 = H1Norm(errSUPG);
Vh errSUPG2 = phi - phisupg2;
real errSUPG2H1 = H1Norm(errSUPG2);
ofstream ff("err.sol",append);
ff << hK[].max << " " << Pe << " " << errstabH1 << " " << errSUPGH1 << " " << errSUPG2H1
    << endl;
}

// Plot the solution along two cuts parallel to the OX and OY axis
int Npts=10;
real[int] ox1(Npts),ox2(Npts);
real[int] oy1(Npts),oy2(Npts);
```

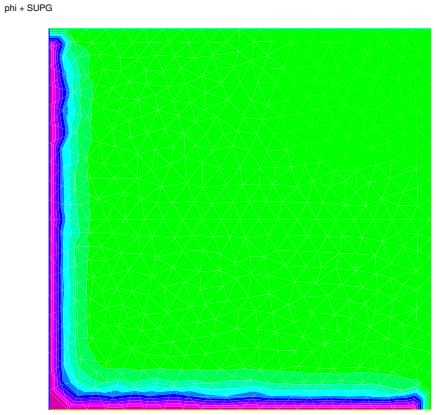
```
for (int j=0;j<Npts;j++) {
// to obtain a OY-axis
    oy1[j]=0.;
    oy2[j]=-.1+0.15*j;
};
for (int j=0;j<10;j++) {
// to obtain a OX-axis
    ox1[j]=-0.1+j*0.15;
    ox2[j]=0.;
};

//solution along x=0.5
{
real xx = 0.5;
Npts = ceil(1./Dy);
ofstream ff("xcut"+Pe+".sol");
real[int] phiV(Npts),phiY(Npts);
real[int] phistabV(Npts),phistabY(Npts);
real[int] phisupgV(Npts),phisupgY(Npts);
real[int] phisupg2V(Npts),phisupg2Y(Npts);
for (int j=0;j<Npts;j++) {
    phiY[j]=Dy*j;
    phiV[j]= phi(xx,phiY[j]);
    phistabV[j]= phistab(xx,phiY[j]);
    phisupgV[j]= phisupg(xx,phiY[j]);
    phisupg2V[j]= phisupg2(xx,phiY[j]);
    ff << phiY[j] << " " << phiV[j] << " " << phistabV[j] << " " << phisupgV[j] << " " <<
        phisupg2V[j] << endl;
};
plot([phisupg2Y,phisupg2V],[phisupgY,phisupgV],[phistabY,phistabV], [phiY,phiV],[ox1,ox2
    ],[oy1,oy2],wait=1, cmm = " solutions along x=0.5");
}


//solution along y=0.5
{
real yy = 0.5;
Npts = ceil(1./Dx);
ofstream ff("ycut"+Pe+".sol");
real[int] phiV(Npts),phiX(Npts);
real[int] phistabV(Npts),phistabY(Npts);
real[int] phisupgV(Npts),phisupgY(Npts);
real[int] phisupg2V(Npts),phisupg2Y(Npts);
for (int j=0;j<Npts;j++) {
    phiX[j]=Dx*j;
    phiV[j]= phi(phiX[j],yy);
    phistabV[j]= phistab(phiX[j],yy);
    phisupgV[j]= phisupg(phiX[j],yy);
    phisupg2V[j]= phisupg2(phiX[j],yy);
    ff << phiX[j] << " " << phiV[j] << " " << phistabV[j] << " " << phisupgV[j] << " " <<
        phisupg2V[j] << endl;
};
plot([phisupg2Y,phisupg2V],[phisupgY,phisupgV],[phistabY,phistabV], [phiX,phiV],[ox1,ox2
    ],[oy1,oy2],wait=1, cmm = " solutions along y=0.5");
}
```



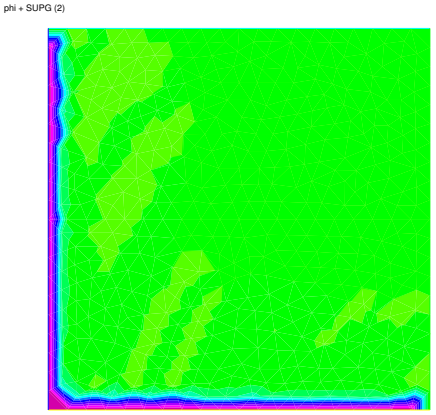
(a) Non-stabilized



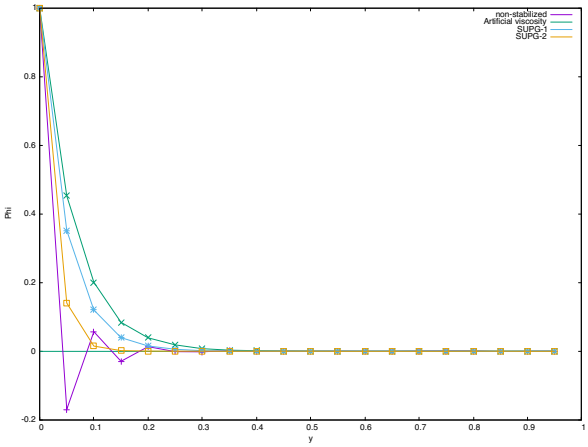
(b) Artificial viscosity



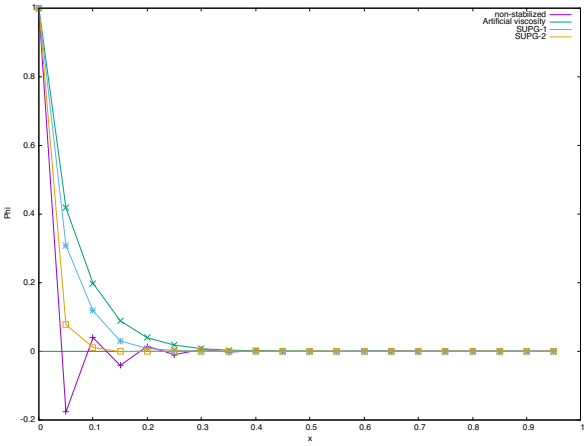
(c) $SUPG(1)$



(d) $SUPG(2)$

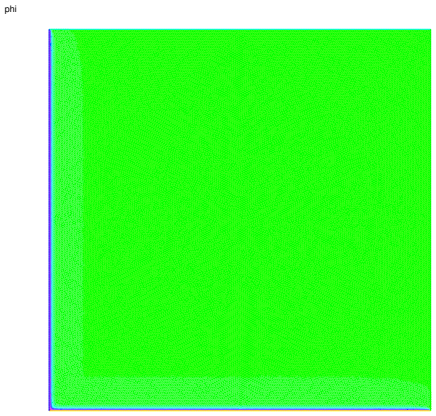


(e) Solutions along $x = 0.5$

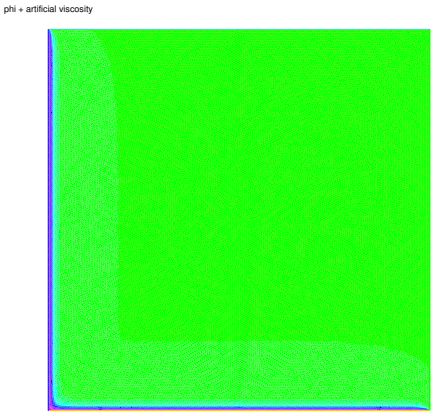


(f) Solutions along $y = 0.5$

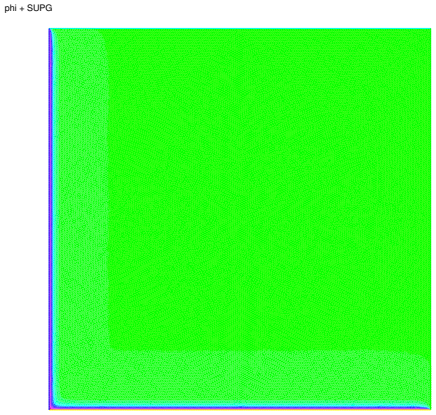
Figure 1: $Pe = 5.80024$



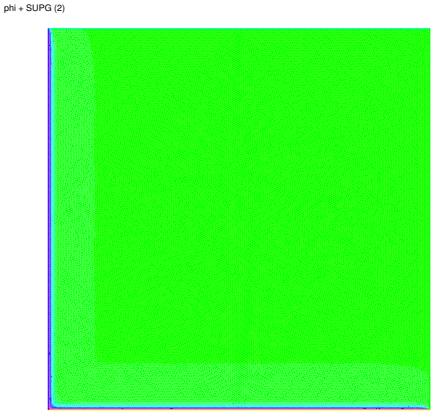
(a) Non-stabilized



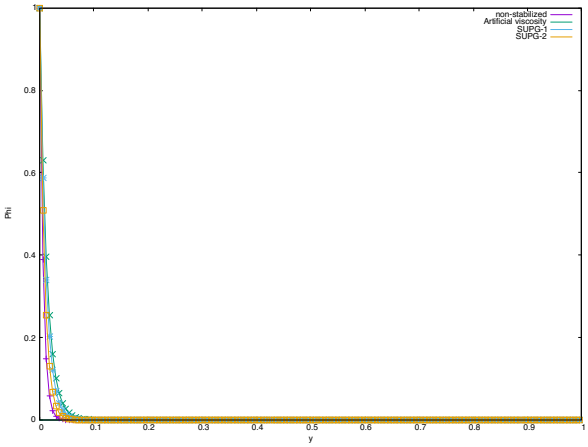
(b) Artificial viscosity



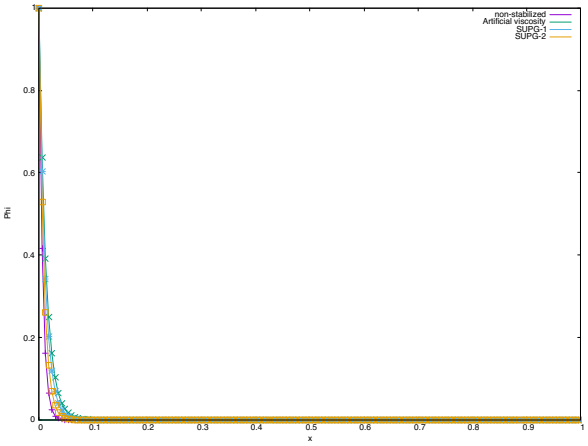
(c) $SUPG(1)$



(d) $SUPG(2)$



(e) Solutions along $x = 0.5$



(f) Solutions along $y = 0.5$

Figure 2: $Pe = 0.820483$

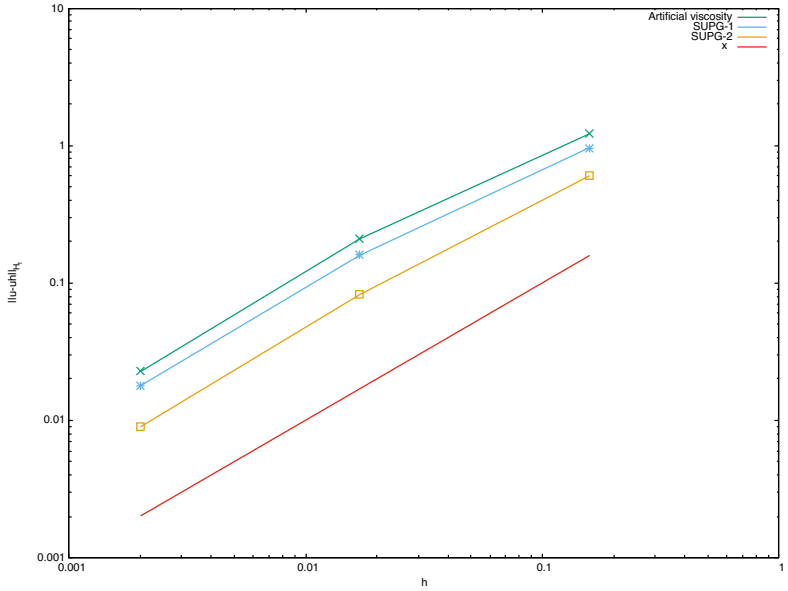


Figure 3: Error $||u_{\mathrm{non-stabilized}} - u_i||$ with i =artificial viscosity, SUPG(1), SUPG(2)

2.2 Stokes equations

2.2.1 Variational formulation

Let the space $V_\phi = \left\{(\mathbf{w}, r) \in [H_0^1(\Omega)]^2 \times L^2(\Omega) | \mathbf{w}_{|\Gamma_1} = \phi, \mathbf{w}_{|\Gamma_{2-4}} = 0\right\}$. Then the variational formulation of problem (2) is

find $(\mathbf{u}, p) \in V_\mathbf{g}$ such that for all $(\mathbf{v}, q) \in V_0$

$$\begin{aligned} \frac{1}{\mathsf{Re}} \int_\Omega (\nabla u_1 \nabla v_1 + \nabla u_2 \nabla v_2) - \int_\Omega p(\partial_x v_1 + \partial_y v_2) \\ + \int_\Omega (\partial_x u_1 + \partial_y u_2)\, q + \int_\Omega \varepsilon p q = \int_\Omega (f_1 v_1 + f_2 v_2), \end{aligned} \tag{4}$$

with $\mathbf{g}(x, y) = (y(1-y), 0)$.

2.2.2 Analytic validation

From the figures 4 and 5, one can see that the solution improves with $h \to 0$ and $\varepsilon \to 0$. In figure 4, the approximation of the solution given by the P_2-P_1 method is limited by the choice of ε. n figure 5, the approximations of the solution given by the P_1b-P_1 and P_1-P_1 methods are limited by the choice of h.

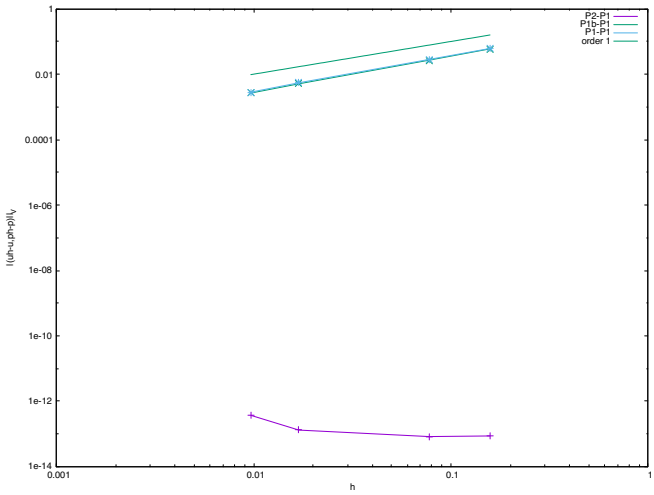


Figure 4: Error $||(\mathbf{u_h} - \mathbf{u}, p_h - p)||_V$ with respect to the mesh size h with $\varepsilon = 10^{-13}$

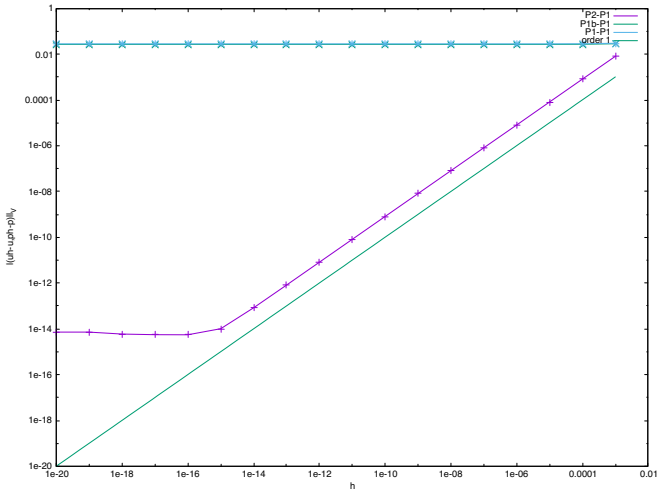


Figure 5: Error $||(\mathbf{u_h} - \mathbf{u}, p_h - p)||_V$ with respect to the parameter ε with $h = 0.0773366$.

2.2.3 FreeFem++ code

```
/*       Generating Mesh */
real L=1;
border b1(t=1,0)          {x=0;          y=t;          label=1;} //Gamma1
border b2(t=0,L)          {x=t;          y=0;          label=2;} //Gamma2
border b3(t=0,1)          {x=L;          y=t;          label=3;} //Gamma3
border b4(t=L,0)          {x=t;          y=1;          label=4;} //Gamma4
real Dx=0.01, Dy=0.01;
mesh Th= buildmesh(b1(ceil(1./Dy))+b2(ceil(L/Dx))+b3(ceil(1./Dy))+b4(ceil(L/Dx)));
plot(Th,wait=1);

/*       Finite Element Spaces */
fespace Vh(Th,[P2,P2,P1]);
fespace Vhb(Th,[P1b,P1b,P1]);
fespace VhStab(Th,[P1,P1,P1]);
fespace P2h(Th,P2); // for plots
fespace Ph(Th,P1); // for plots
fespace Pbh(Th,P1b); // for plots
fespace Xh(Th,P0);

/*       macros  */
macro grad(u) [dx(u),dy(u)] //eom ¡- IMPORTANT : macro ends with a comments

/*       Defining the variational problem        */
func g=y*(1-y);
real f1=0.,f2=0.;
real Re = 1;
real nu = 1./Re;
real epsilon = 1e-3;
varf bilinear ([u1,u2,p],[v1,v2,q]) = //P2-P1 or P1b-P1 fomulation
    int2d(Th)( nu* ( dx(u1)*dx(v1) + dy(u1)*dy(v1)
             +  dx(u2)*dx(v2) + dy(u2)*dy(v2) )
             - p*q*epsilon
             - p*dx(v1)- p*dy(v2)
             - dx(u1)*q- dy(u2)*q
            )
  + on(1,u1=g,u2=0) // boundary conditions
  + on(2,4,u1=0,u2=0)
  ;

  real delta =0.01;
  Xh hK2 = hTriangle * hTriangle;
  varf bilinearStab ([u1,u2,p],[v1,v2,q]) = //Stabilized P1-P1 formulation
    int2d(Th)( nu* ( dx(u1)*dx(v1) + dy(u1)*dy(v1)
             +  dx(u2)*dx(v2) + dy(u2)*dy(v2) )
             + p*q*epsilon
             + delta * hK2* (dx(p)*dx(q)+dy(p)*dy(q) )
             - p*dx(v1)- p*dy(v2)
             + dx(u1)*q+ dy(u2)*q
            )
  + on(1,u1=g,u2=0) // boundary conditions
  + on(2,4,u1=0,u2=0)
  ;

varf linear([u1,u2,p],[v1,v2,q]) =  int2d(Th)( f1*v1+f2*v2 ) ;

/*       Solving the problem      */
real time=clock();
Vh [u,v,p];
matrix A = bilinear(Vh,Vh,solver=UMFPACK); // assemble the FE matrix
real[int] bc = bilinear(0,Vh); // Dirichlet boundary condition
real[int] b = linear(0,Vh); // assemble the right hand side vector
b += bc; // modifying the entires for the Dirichlet BC.
u[] = A^-1*b; //solve the problem
time = clock()-time;
cout << "Mixed formulation P2-P1 : Solver time = " + time << endl;
```

```
66
67  real timeb=clock();
68  Vhb [ub,vb,pb];
69  matrix Ab = bilinear(Vhb,Vhb,solver=UMFPACK); // assemble the FE matrix
70  real[int] bcb = bilinear(0,Vhb); // Dirichlet boundary condition
71  real[int] bb = linear(0,Vhb); // assemble the right hand side vector
72  bb += bcb; // modifying the entires for the Dirichlet BC.
73  ub[] = Ab^-1*bb; //solve the problem
74  timeb = clock()-timeb;
75  cout << "Mixed formulation P1b-P1 : Solver time = " + timeb << endl;
76
77  real times=clock();
78  VhStab [uss,vss,pss];
79  matrix As = bilinearStab(VhStab,VhStab,solver = UMFPACK, factorize=0); // assemble the FE
        matrix
80  real[int] bcs = bilinearStab(0,VhStab); // Dirichlet boundary condition
81  real[int] bs = linear(0,VhStab); // assemble the right hand side vector
82  bs += bcs; // modifying the entires for the Dirichlet BC.
83  uss[] = As^-1*bs; //solve the problem
84  times = clock()-times;
85  cout << "Stabilized P1-P1 formulation : Solver time = " + times << endl;
86
87  /* Visualize the solution        */
88  plot(coef=0.03,cmm="Mixed formulation P2-P1 : Flow field [u,v]",[u,v], value=1, wait=1);
89  plot(cmm="Mixed formulation P2-P1 : Pressure p",p, value=1, fill=1, wait=1);
90  P2h V = sqrt(u*u+v*v);
91  plot(cmm="Mixed formulation P2-P1 : Fluid velocity magnitude V",V, value=1, fill=1, wait
        =1);
92
93  plot(coef=0.03,cmm="Mixed formulation P1b-P1 : Flow field [u,v]",[ub,ub], value=1, wait
        =1);
94  plot(cmm="Mixed formulation P1b-P1 : Pressure p",pb, value=1, fill=1, wait=1);
95  Pbh Vb = sqrt(ub*ub+vb*vb);
96  plot(cmm="Mixed formulation P1b-P1 : Fluid velocity magnitude V",Vb, value=1, fill=1,
        wait=1);
97
98  plot(coef=0.03,cmm="Stabilized P1-P1 formulation : Flow field [u,v]",[uss,vss], value=1,
        wait=1);
99  plot(cmm="Stabilized P1-P1 formulation : Pressure p",pss, value=1, fill=1, wait=1);
100 Ph Vs = sqrt(uss*uss+vss*vss);
101 plot(cmm="Stabilized P1-P1 formulation : Fluid velocity magnitude V",Vs, value=1, fill=1,
         wait=1);
102
103 /* Compatibility condition*/
104 real uIN  = int1d(Th,1)(g*N.x);
105 real uOUT = int1d(Th,3)(u*N.x+v*N.y);
106 real errU = sqrt((uIN+uOUT)*(uIN+uOUT));
107 cout << endl << "||uIN-(-uOUT)||_L2 = " + errU << endl << endl;
108
109
110 /* Analytical validation         */
111 func ue = y*(1-y);
112 func dyue = 1-2*y;
113 func ve = 0.;
114 func pe = 2./Re*(1-x);
115
116 macro NuH1(u) (sqrt( int2d(Th)((u-ue)*(u-ue) + dx(u)*dx(u)+ (dy(u)-dyue)*(dy(u)-dyue)))
        //eom
117 macro NvH1(v) (sqrt( int2d(Th)(v*v + dx(v)*dx(v)+ dy(v)*dy(v) ))) //eom
118 macro NpL2(p) (sqrt( int2d(Th)((p-pe)*(p-pe)))) //eom
119 macro NV(u,v,p) (sqrt(u*u + v*v + p*p)) //eom
120
121 //Mixed formulation
122 real errUnormH1 = NuH1(u);
123 real errVnormH1 = NvH1(v);
124 real errPnormL2  = NpL2(p);
125 real errnormV = NV(errUnormH1,errVnormH1,errPnormL2);
126 cout << endl << "Mixed formulation P2-P1"<< endl;
```

```
127 cout << "||u1h-u1||_H1 =" + errUnormH1 << endl;
128 cout << "||u2h-u2||_H1 =" + errVnormH1 << endl;
129 cout << "||ph-p||_L2 =" + errPnormL2 << endl;
130 cout << "||(uh-u,ph-p)||_V =" + errnormV << endl << endl;
131
132 real errUbnormH1 = NuH1(ub);
133 real errVbnormH1 = NvH1(vb);
134 real errPbnormL2  = NpL2(pb);
135 real errnormVb = NV(errUbnormH1,errVbnormH1,errPbnormL2);
136 cout << endl << "Mixed formulation P1b-P1"<< endl;
137 cout << "||u1h-u1||_H1 =" + errUbnormH1 << endl;
138 cout << "||u2h-u2||_H1 =" + errVbnormH1 << endl;
139 cout << "||ph-p||_L2 =" + errPbnormL2 << endl;
140 cout << "||(uh-u,ph-p)||_V =" + errnormVb << endl << endl;
141
142 real errUsnormH1 = NuH1(uss);
143 real errVsnormH1 = NvH1(vss);
144 real errPsnormL2  = NpL2(pss);
145 real errnormVs = NV(errUsnormH1,errVsnormH1,errPsnormL2);
146 cout << endl << "Stabilized P1-P1 formulation"<< endl;
147 cout << "||u1h-u1||_H1 =" + errUsnormH1 << endl;
148 cout << "||u2h-u2||_H1 =" + errVsnormH1 << endl;
149 cout << "||ph-p||_L2 =" + errPsnormL2 << endl;
150 cout << "||(uh-u,ph-p)||_V =" + errnormVs << endl << endl;
```