Assignment 2, Mek4250

Sebastian Gjertsen

6. mai 2016

1

7.1

First we have to show:

$$a(u,v) \leq C||u||_{H_0^1}||v||_{H_0^1}$$

We start with:

$$\int \nabla u : \nabla v dx \leq (\int (\nabla u \nabla v)^2 dx)^{\frac{1}{2}} =$$

Using Cauchy- Schwartz

$$||\nabla u \nabla v||_{L^2} \leq C||\nabla u||_{L^2}||\nabla v||_{L^2} =$$

$$C|u|_{H_0^1}|v|_{H_0^1} \leq C||u||_{H_0^1}||v||_{H_0^1}$$

The last line we can see is true since the H1 seminorm has to be smaller than the H1 norm since it contains the L2 norm of u in addition to the gradient of u. Next we want to show:

$$b(q,u) \leq C||u||_{H_0^1}||q||_{L^2}$$

$$\int q\nabla * u dx \leq (\int (q\nabla * u)^2 dx)^{\frac{1}{2}} \leq ||q||_{L^2}||\nabla * u||_{L^2} \text{used Cauchy Schwartz}$$

Since we now have sorted the q we can focus on the divergence

$$||\nabla * u||_{L^2} \leq ||\nabla * u||_{L^2} + ||\frac{\partial u_y}{\partial x} + \frac{\partial u_x}{\partial y}||_{L^2} = ||\nabla u||_{L^2} \leq C||u||_{H^1}$$

we use the same trick as before and get

$$\int q\nabla * u dx \leq C||u||_{H_0^1}||q||_{L^2}$$

Lastly we show:

$$a(u,u) \geq ||u||^2_{H_0^1}$$

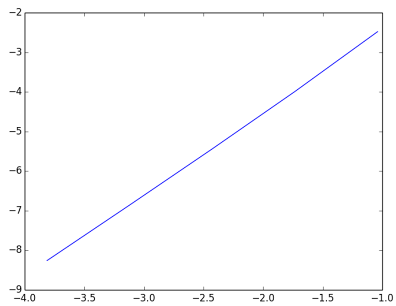
$$
\begin{aligned}
||u||^2_{H^1_0} = ||u||^2_{L^2}||\nabla u||^2_{L^2} & \\
\text{using Poincares inequality} \quad & \leq D||\nabla u||^2_{L^2} + ||\nabla u||^2_{L^2} = ||\nabla u||^2_{L^2}(D+1) \\
& = C||\nabla u||^2_{L^2} \\
& \leq \int \nabla u : \nabla u dx
\end{aligned}
$$

7.6

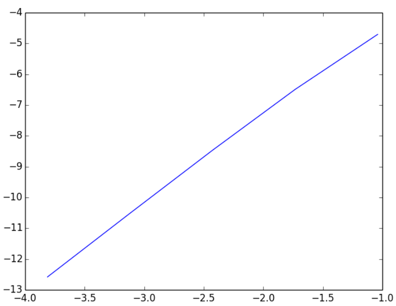
$$||u - u_h||_1 + ||p - p_h||_0 \leq Ch^{\alpha}||u|| + Ch^{\beta}||p||_{\beta+1}$$

In the 4 figures we see log log plots of u from the different mixes of function spaces. We seem to get good convergence rates for all the combinations but best when the spaces are only one degree apart.

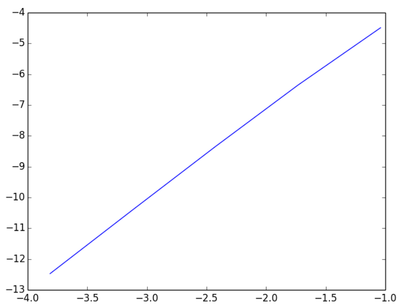
Elements	α_u	α_p
P4-P3	4.23752218115	4.07080574134
P4-P2	2.88750262912	2.91633169585
P3-P2	2.85257225007	2.91774992053
P3-P1	2.08150448230	2.15170883599



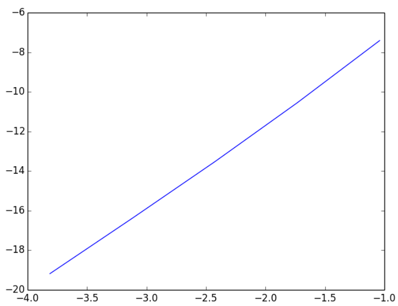
Figur 1: P3-P1



Figur 2: P3-P2



Figur 3: P4-P2

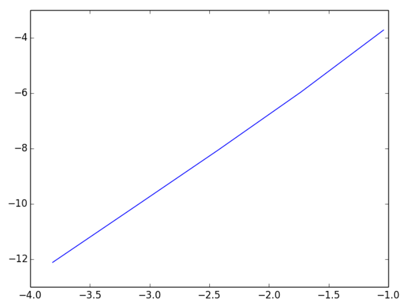


Figur 4: P4-P3,

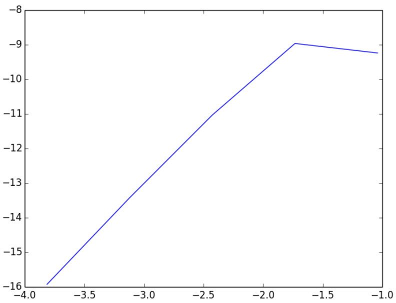
7.7

To calculate the wall shear stress i calculated $\epsilon = 0.5 * (\nabla u + \nabla u^T)$, i focused the wall shear stress on the bottom wall and calculated it by hand and got $\epsilon = 0.5 * (\pi - 2) \approx 0.57$. The convergence was calculated in a similar fashion as 7.6.

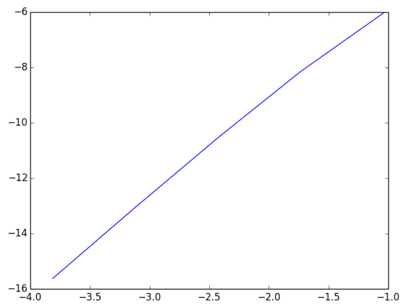
Elements	α_{stress}
P4-P3	4.08979685189
P4-P2	3.47692041535
P3-P2	2.5719146479
P3-P1	3.01745034613



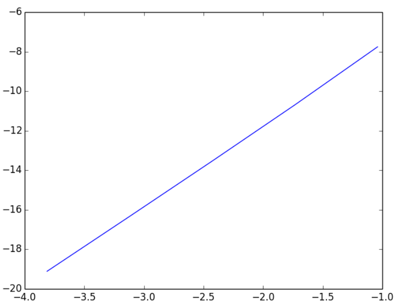
Figur 5: P3-P1



Figur 6: P3-P2



Figur 7: P4-P2



Figur 8: P4-P3,

2

a)

$$-\mu\nabla u - \lambda\nabla\nabla * u = f$$

$$u_e = (\pi x\cos(\pi xy), -\pi y cos(\pi xy))$$

I used sympy to calculate the laplacian of u_e since the divergence term is zero by construction. Giving

$$f = \left(\pi^2(\pi*x*(x^2+y^2)*cos(\pi xy)+2ysin(\pi xy)), \pi^2(\pi*y*(x^2+y^2)*cos(\pi xy)+2ysin(\pi xy))\right)$$

```
from sympy import *
x, y, pi = symbols('x y pi')
#print ((x+y)**2 * (x+1)).expand()
#(pi*x*cos(pi*x*y), -pi*y*cos(pi*x*y))
f1 = simplify(diff(pi*x*cos(pi*x*y), x,x) + diff(pi*x*cos(pi*x*y), y,y))

f2 = simplify(diff(-pi*y*cos(pi*x*y), x,x) + diff(-pi*y*cos(pi*x*y), y,y))

f = lambdify((x, y), [f1, f2])
print f

#k, m, n = symbols('k m n', integer=True)
#f, g, h = map(Function, 'fgh')
```

b)

The equation was solved on the form:

$$\nu(\nabla u, \nabla v) - \lambda(\nabla\nabla * u, v) = (f, v)$$

Errornorm for u P1

N / λ	1	100	1000
8	6.917632e-02	6.917632e-02	6.917632e-02
16	1.782853e-02	1.782853e-02	1.782853e-02
32	4.492167e-03	4.492167e-03	4.492167e-03
64	1.125256e-03	1.125256e-03	1.125256e-03

Errornorm for u P2

N / λ	1	100	1000
8	1.492836e-02	3.750042e+00	1.426643e+00
16	3.947793e-03	6.060935e-01	1.681095e+00
32	1.001643e-03	6.508181e-01	3.957783e+00
64	2.513488e-04	2.416408e+00	1.143778e+00

We can see that this is quite bad error norms. The second time I used that trick of calculating the PDE as a dual function space creating a new function from $P = \nabla * u$ giving two equations to solve:

$$-\mu\nabla u - \lambda\nabla P = f$$

$$P = \nabla * u$$

Errornorm for u P2-P1

N / λ	1	100	1000
8	1.989689e-03	1.970400e-03	1.971845e-03
16	2.489791e-04	2.483019e-04	2.483559e-04
32	3.117027e-05	3.114920e-05	3.115123e-05
64	3.919508e-06	3.918951e-06	3.919020e-06

Errornorm for u P3-P2

N / λ	1	100	1000
8	8.157349e-05	8.028915e-05	8.027183e-05
16	4.984589e-06	4.921852e-06	4.921303e-06
32	3.405847e-07	3.378894e-07	3.378800e-07
64	1.295475e-07	1.295024e-07	1.295015e-07

c)

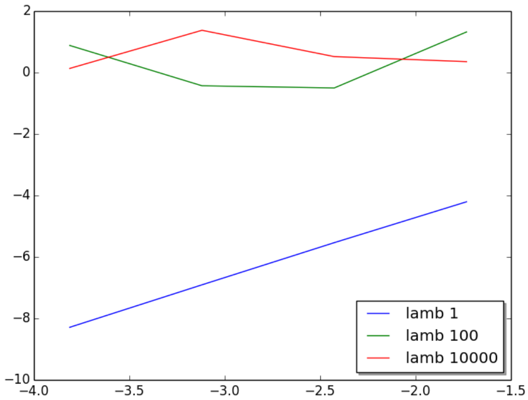
Convergence for single functionspace

λ	α
1	1.96553403788
100	0.179941425079
10000	-0.0278840148111

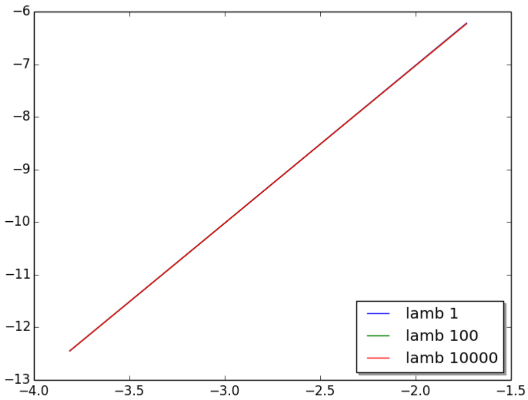
Convergence for mixed functionspace

λ	α
1	2.99607463617
100	2.99162444897
10000	2.99195596264

In the first approximation I calculated the PDE with a single functionspace and calculating the $\lambda\nabla\nabla u$ straight into the program. We can see from the table of α that we get a very bad convergence rate, which indicates that locking is occuring. This is also evident when we look at the log log plots. Where we get a nice straight line for $\lambda = 1$ and bad results for the others. The second time I used that trick of calculating the PDE as a dual function space like before with P We see now from the figure that we get straight lines for all λ



Figur 9: Log plot without trick



Figur 10: Log plot with trick

Code

7.6 Code

```
from dolfin import *
import numpy as np
import matplotlib.pyplot as plt
set_log_active(False)
def solve_this_shit(N,degree_u,degree_p):
    mesh = UnitSquareMesh(N,N)

    V = VectorFunctionSpace(mesh,"CG",degree_u)
    Q = FunctionSpace(mesh,"CG",degree_p)
    VQ = V*Q
```

```python
u,p = TrialFunctions(VQ)
v,q = TestFunctions(VQ)


class Up(SubDomain):
    def inside(self, x, on_boundary):
        return near(x[1],1)
class Down(SubDomain):
    def inside(self, x, on_boundary):
        return near(x[1],0)
class Left(SubDomain):
    def inside(self, x, on_boundary):
        return near(x[0],0)
class Right(SubDomain):
    def inside(self, x, on_boundary):
        return near(x[0],1)



u_e = Expression(("sin(pi*x[1])","cos(pi*x[0])"))
p_e = Expression("sin(2*pi*x[0])")

up = Up()
down = Down()
left = Left()
right = Right()
bound = FacetFunction("size_t",mesh)
bound.set_all(0)
up.mark(bound,0)
down.mark(bound,1)
left.mark(bound,2)
right.mark(bound,3)
#plot(bound, interactive=True)
ds = Measure("ds",subdomain_data=bound)
n = FacetNormal(mesh)
ds = ds[bound]


bc1 = DirichletBC(VQ.sub(0), u_e, up)
bc2 = DirichletBC(VQ.sub(1), p_e , up)
bc3 = DirichletBC(VQ.sub(0), u_e, down)
bc4 = DirichletBC(VQ.sub(1), p_e , down)
bc5 = DirichletBC(VQ.sub(0), u_e, left)
bc6 = DirichletBC(VQ.sub(1), p_e , left)
bc7 = DirichletBC(VQ.sub(0), u_e, right)
bc8 = DirichletBC(VQ.sub(1), p_e , right)
bcs = [bc1,bc2,bc3,bc4,bc5,bc6,bc7,bc8]

f = Expression(("pi*pi*sin(pi*x[1])-2*pi*cos(2*pi*x[0])","pi*pi*cos(pi*x[0]
a = inner(grad(u), grad(v))*dx + div(u)*q*dx + div(v)*p*dx - inner(f, v)*dx
```

```
    up = Function(VQ)
    solve(lhs(a)==rhs(a),up,bcs)
    u_,p_ = up.split()


    """
    #print
    plot(u_exact,interactive= True)
    plot(u_, interactive = True)
    plot(p_exact,interactive = True)
    plot(p_, interactive = True)"""


    R = VectorFunctionSpace(mesh, 'R', 0)
    c = TestFunction(R)
    tau = 0.5*(grad(u_)+grad(u_).T)
    n = FacetNormal(mesh)
    forces = assemble(dot(dot(tau, n), c)*ds(1)).array()
    print "x-direction = {}, y-direction = {}".format(*forces)
    #print forces
    #force = forces[0]
    """u_ex = project(u_e,V)
    R1 = VectorFunctionSpace(mesh, 'R', 0)
    c1 = TestFunction(R1)
    tau = 0.5*(grad(u_ex)+grad(u_ex).T)
    n = FacetNormal(mesh)
    forces_ex = assemble(dot(dot(tau, n), c1)*ds(1)).array()
    print "x-direction = {}, y-direction = {}".format(*forces_ex)"""

    #print forces_ex
    #force_exact = forces_ex[1]

    forces_ex=-0.5*(-2+pi)
    return u_,u_e,p_,p_e,mesh,V,Q#, forces[0],forces_ex

N = [4,8,16,32,64]
def convergence_force(N,degree_u,degree_p):
    b = np.zeros(len(N))
    a = np.zeros(len(N))
    d = np.zeros(len(N))

    for i in range(len(N)):
        u_,ue,p_,pe,mesh,V,Q,forces,forces_ex = solve_this_shit(N[i],degree_u,d
        e_u = abs(forces_ex-forces)
        #e_u = errornorm(forces_ex,forces, norm_type = "l2",degree_rise = 3)

        print "N:",N[i] ," force_error ",e_u
        print "------------------------------------------"
        a[i] = np.log(mesh.hmin())
```

```
        b[i] = np.log(e_u)

    plt.plot(a,b,label = "P%.f-P%.f"%(degree_u,degree_p))

    A = np.vstack([a,np.ones(len(a))]).T
    alpha,c = np.linalg.lstsq(A,b)[0]
    print "force-alpha: ", alpha
    print "force-c: ", c
    plt.show()
#convergence_force(N,4,3)


def convergence_e(N,degree_u,degree_p):
    b = np.zeros(len(N))
    a = np.zeros(len(N))
    d = np.zeros(len(N))

    for i in range(len(N)):
        u_,ue,p_,pe,mesh,V,Q = solve_this_shit(N[i],degree_u,degree_p)
        e_u = errornorm(ue, u_, norm_type = "h1",degree_rise = 3)
        e_p = errornorm(pe, p_, norm_type = "l2",degree_rise = 3)

        #print "N : ",N ,"lambda: ",lamb
        print "N:",N[i] ," u-errornorm: ",e_u
        print "N:",N[i] ," p-errornorm: ",e_p
        print "------------------------------------------"
        a[i] = np.log(mesh.hmin())
        b[i] = np.log(e_u)
        d[i] = np.log(e_p)

    plt.plot(a,b)
    #plt.plot(d,b)
    A = np.vstack([a,np.ones(len(a))]).T
    alpha,c = np.linalg.lstsq(A,b)[0]
    alpha_p,c_p = np.linalg.lstsq(A,d)[0]
    print "u-alpha: ", alpha
    print "u-c: ", c
    print "--------------"
    print "p-alpha: ", alpha_p
    print "p-c: ", c_p
    plt.show()
convergence_e(N,4,3)
```

2 Code

```
from dolfin import *
import numpy as np
import matplotlib.pyplot as plt

set_log_active(False)
def solve_me(N ,mu,lamb):
        mesh = UnitSquareMesh(N,N)
        V = VectorFunctionSpace(mesh,"CG",3)
        W = VectorFunctionSpace(mesh,"CG",4)
        Q = FunctionSpace(mesh,"CG",2)
        VQ = V*Q
        up = TrialFunction(VQ)
        u, p = split(up)
        vq = TestFunction(VQ)
        v, q = split(vq)

        class Boundary(SubDomain):
                def inside(self, x, on_boundary):
                        return on_boundary
        u_e = Expression(("pi*x[0]*cos(pi*x[0]*x[1])","-pi*x[1]*cos(pi*x[0]*x[1
        u_exact = project(u_e,W)
        #plot(u_e_plot,interactive=True)

        f = Expression(("pi*pi*(pi*x[0]*(x[0]*x[0]+x[1]*x[1])*cos(pi*x[0]*x[1])
                        " -pi*pi*(pi*x[1]*(x[0]*x[0]+x[1]*x[1])*cos(pi*x[0]*x[
        boundary = Boundary()
        bound = FacetFunction("size_t",mesh)
        boundary.mark(bound,1)
        #plot(bound,interactive=True)

        bc1 = DirichletBC(VQ.sub(0), u_e, boundary)
        #bc2 = DirichletBC(Q, 0, boundary)
        bcs = [bc1]
        mu = Constant(mu); lamb = Constant(lamb)

        a1 = mu*inner(grad(u),grad(v))*dx + p*div(v)*dx
        a2 = - (1.0/lamb)*p*q*dx  + dot(div(u),q)*dx
        L = inner(f,v)*dx
        u_p_ = Function(VQ)
        solve(a1+a2==L,u_p_,bcs)
        u_, p_ = u_p_.split()
        #plot(u_,interactive=True)
        return u_, u_exact,mesh

def solve_me_bad(N,mu,lamb):
        mesh = UnitSquareMesh(N,N)
        V = VectorFunctionSpace(mesh,"CG",2)
        W = VectorFunctionSpace(mesh,"CG",3)
```

```
    u = TrialFunction(V)
    v = TestFunction(V)
    class Boundary(SubDomain):
        def inside(self, x, on_boundary):
            return on_boundary
    u_e = Expression(("pi*x[0]*cos(pi*x[0]*x[1])","-pi*x[1]*cos(pi*x[0]*x[1
    u_exact = project(u_e,W)
    f = Expression(("pi*pi*(pi*x[0]*(x[0]*x[0]+x[1]*x[1])*cos(pi*x[0]*x[1])
                    " -pi*pi*(pi*x[1]*(x[0]*x[0]+x[1]*x[1])*cos(pi*x[0]*x[
    boundary = Boundary()
    bound = FacetFunction("size_t",mesh)
    boundary.mark(bound,1)
    #plot(bound,interactive=True)

    bc1 = DirichletBC(V, u_e, boundary)
    #bc2 = DirichletBC(Q, 0, boundary)
    bcs = [bc1]
    mu = Constant(mu)
    lamb = Constant(lamb)
    a = mu*inner(grad(u),grad(v))*dx - lamb*inner(grad(div(u)),v)*dx
    L = inner(f,v)*dx
    u_ = Function(V)
    solve(a==L,u_,bcs)
    return u_exact, u_, mesh



N = [8,16,32,64]
mu = 1.0
b = np.zeros(len(N))
a = np.zeros(len(N))
for lamb in[1,100,10000]:
    for i in range(len(N)):
        u ,u_exact,mesh = solve_me(N[i],mu,lamb)
        e = errornorm(u_exact, u, norm_type = "l2",degree_rise = 2)
        #print "N : ",N ,"lambda: ",lamb
        print "N: %e Lambda: %e Errornorm: %e"%(N[i],lamb ,e)
        #plot(u_exact)
        #plot(u)
        #interactive()
        b[i] = np.log(e)#/u_norm)
        a[i] = np.log(mesh.hmin())#1./(N[i]))
        #print u_norm
        #u_norm = sum(u_ex*u_ex)**0.5
    plt.plot(a,b, label =("lamb %.f "%(lamb)))
    axes = plt.gca()
    legend = axes.legend(loc='lower right', shadow=True)
    A = np.vstack([a,np.ones(len(a))]).T
    alpha,c = np.linalg.lstsq(A,b)[0]
```

```
        print "lambda: ",lamb
        print "alpha: ", alpha
        print "c: ", c
        print "-------------"
plt.show()
```