Mandatory assignment 2
MEK4250

Christian Pedersen

May 6, 2016

Exercise 1

Exercise 7.1

We are to prove the three conditions (7.14)-(7.16) in the lecture notes for $V_h = H_0^1$ and $Q_h = L^2$. I will use the Cauchy-Schwarz inequality (C-S)

$$| < u, v > | \leq ||u||\ ||v|| \tag{1}$$

and the Poincare inequality (PC)

$$||u||_{L^2} \leq C||\nabla u||_{L^2} = C|u|_{H^1} \tag{2}$$

Condition (7.14):

- show that $a(u_h, v_h) \leq C_1||u_h||_{H^1}||v_h||_{H^1}$

$$\begin{aligned} a(u_h, v_h) &= \int \nabla u_h : \nabla v_h dx \\ &\leq \left(\left(\int \nabla u_h : \nabla v_h \right)^2 \right)^{1/2} \\ &= |\langle \nabla u_h, \nabla v_h \rangle| \\ &\overset{\text{C-S}}{\leq} ||\nabla u_h||_{L^2}||\nabla v_h||_{L^2} \\ &= |u_h|_{H^1}|v_h|_{H^1} \\ &\leq ||u_h||_{H^1}||v_h||_{H^1} \end{aligned} \tag{3}$$

In the last inequality I used that a semi norm will yield a lesser value than a full norm.

Condition (7.15):

- show that $b(u_h, q_h) \leq C_2 ||q_h||_{L^2} ||u_h||_{H^1}$. We start with

$$\begin{aligned} b(u_h, q_h) &= \int q_n \nabla \cdot u_h dx \\ &\leq \left(\left(\int q_n \nabla \cdot u_h dx \right)^2 \right)^{1/2} \\ &= | < q_h, \nabla \cdot u_h > | \\ &\overset{\text{C-S}}{\leq} ||q_h||_{L^2} ||\nabla \cdot u_h||_{L^2} \end{aligned} \tag{4}$$

Since we have $||q_h||_{L^2}$ on both sides all we have to do is prove that $||\nabla \cdot u_h||_{L^2} \leq ||u_h||_{H^1}$. Treating u_h as a 2D vector with x and y components u and v. Raising the degree on both sides and writing the terms out gives us left side

$$||\nabla \cdot u_h||_{L^2}^2 = || \left(\frac{\partial u}{\partial y} \right)^2 + 2 \frac{\partial u}{\partial x} \frac{\partial v}{\partial y} + \left(\frac{\partial v}{\partial y} \right)^2 ||_{L^2}^2 \tag{5}$$

and right side

$$||u_h||_{H^1}^2 = ||u_h^2 + \left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial u}{\partial y} \right)^2 + \left(\frac{\partial v}{\partial x} \right)^2 + \left(\frac{\partial v}{\partial y} \right)^2 ||_{L^2}^2 \tag{6}$$

A trick we introduce is to add something to the left hand side. First we add $\left(\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} \right)^2$. This gives us a left hand side looking like

$$||2 \left(\frac{\partial u}{\partial x} \right)^2 + 2 \left(\frac{\partial v}{\partial y} \right)^2 ||_{L^2}^2 \tag{7}$$

Now we add $2 \left(u_h^2 + \left(\frac{\partial u}{\partial y} \right)^2 + \left(\frac{\partial v}{\partial x} \right)^2 \right)$. The left hand side now looks like

$$2||u_h^2 + \left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial u}{\partial y} \right)^2 + \left(\frac{\partial v}{\partial x} \right)^2 + \left(\frac{\partial v}{\partial y} \right)^2 ||_{L^2}^2 \tag{8}$$

Now we see that by adding many positive terms to the left hand side, we have managed to obtain the expression on the right hand side times two. This proves the boundedness and gives us $C_2 = \frac{1}{2}$.

Condition (7.16):

- show that $a(u_h, q_h) \geq C_3 ||u_h||^2_{L^2}$

$$\begin{aligned} a(u_h, u_h) &= \int \nabla u_h : \nabla u_h \\ &= ||\nabla u_h||^2_{L^2} \\ &= ||\nabla u_h||^2_{L^2} \\ &\overset{\text{PC}}{\geq} C_3 ||u_h||^2_{L^2} \end{aligned} \tag{9}$$

where $C_3 = 1/C^2$ and C is from equation 2.

Exercise 7.6

Equation set

$$\begin{aligned} -\Delta u - \nabla p &= f \qquad \text{in} \quad \Omega = (0,1)^2 \\ \nabla \cdot u &= 0 \qquad \text{in} \quad \Omega = (0,1)^2 \end{aligned} \tag{10}$$

with $u_{exact} = (sin(\pi y),\ \ cos(\pi x))$ and $p_{exact} = sin(2\pi x)$. This gives us a source term

$$f = \left[\pi^2 sin(\pi y) - 2\pi cos(2\pi x),\ \ \pi^2 cos(\pi x)\right] \tag{11}$$

The optimal convergence rate that we want to obtain is on the form

$$||u - u_h||_1 + ||p - p_h||_0 \leq Ch^k ||u||_{k+1} + Dh^{l+1} ||p||_{l+1} \tag{12}$$

where k is the order of the velocity polynomials and l is the order of the pressure polynomials. The H1 error for velocity and the L2 error for the pressure can be seen in figure 1.

We see from the figures that we the error has a unexpected behaviour for some of the polynomial pairs when h gets big. To understand what is going on we need to take a look at the wanted convergence rate.

P4-P3

For this combination of elements we are expecting that the error norm should behave something like

$$||u - u_h||_1 + ||p - p_h||_0 \leq C_0 h^4 \left(||u||_5 + ||p||_4\right) \tag{13}$$

Assuming that the norms on the right hand side is of the same order we see that both of the error norms on the left hand side should be expected to have a convergence rate of 4. This is also the result we got from the numerical calculation.

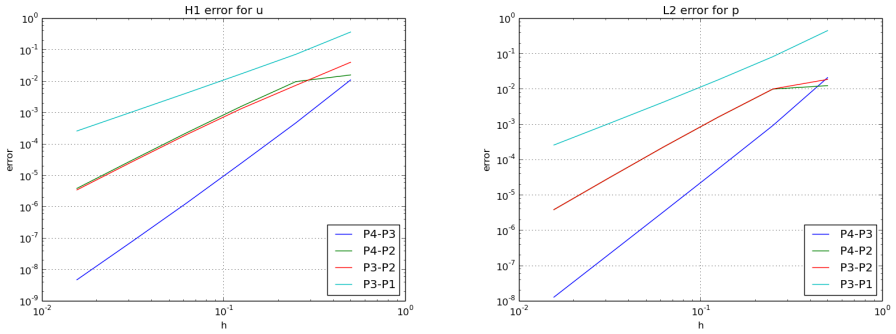

Figure 1: H1 error for u and L2 error for p.

Table 1: Error and convergence rate for P4-P3 elements.

N	2	4	8	16	32	64
H1 error	0.01069	0.0004594	2.368e-05	1.271e-06	7.509e-08	4.62e-09
Convergence rate	-	4.541	4.278	4.219	4.082	4.023
L2 error	0.02061	0.0009062	5.379e-05	3.258e-06	2.017e-07	1.257e-08
Convergence rate	-	4.507	4.075	4.045	4.014	4.004

P4-P2

Table 2: Error norms and convergence rates for P4-P2 elements.

N	2	4	8	16	32	64
H1 error	0.0154	0.009467	0.001534	0.0002202	2.927e-05	3.758e-06
Convergence rate	-	0.7016	2.625	2.801	2.911	2.961
L2 error	0.01221	0.009755	0.001562	0.0002227	2.945e-05	3.771e-06
Convergence rate	-	0.3235	2.643	2.811	2.919	2.966

For this combination the error norms should behave something like

$$||u - u_h||_1 + ||p - p_h||_0 \leq Ch^4||u||_5 + Dh^3||p||_4 \tag{14}$$

Assuming that both coefficients and norms on the right hand side is of the same order, we see that when h goes towards zero, the first term on the right hand side will go towards zero much faster than the second term. This tells us that the error norms should behave something like h^3 and give us a third order convergence. This is also what we get from calculations.

P3-P2

Table 3: Error norms and convergence rates for P3-P2 elements.

N	2	4	8	16	32	64
H1 error	0.03913	0.007193	0.001293	0.000193	2.603e-05	3.361e-06
Convergence rate	-	2.443	2.476	2.744	2.891	2.953
L2 error	0.01835	0.009826	0.001565	0.000223	2.947e-05	3.772e-06
Convergence rate	-	0.9007	2.65	2.811	2.919	2.966

For this combination we expect

$$||u - u_h||_1 + ||p - p_h||_0 \leq C_0 h^3 \left(||u||_4 + ||p||_3\right) \tag{15}$$

Both terms on the right hand side behaves like h^3 so we expect third order convergence. And that is what we get.

P3-P1

Table 4: Error norms and convergence rates for P3-P1 elements.

N	2	4	8	16	32	64
H1 error	0.3591	0.07011	0.01668	0.004066	0.001012	0.0002533
Convergence rate	-	2.357	2.072	2.036	2.006	1.999
L2 error	0.4413	0.08066	0.01778	0.004171	0.001023	0.0002544
Convergence rate	-	2.452	2.182	2.092	2.028	2.007

Here we expect

$$||u - u_h||_1 + ||p - p_h||_0 \leq Ch^3 ||u||_4 + Dh^2 ||p||_2 \tag{16}$$

Here, as for the P4-P2 elements, we see that as h goes towards zero, the first term on the right hand side goes to zero much faster and both error norms will behave something like h^2. And second order convergence is what we get.

Exercise 7.7

Still solving Stokes flow but now with

$$\begin{aligned} u &= [sin(\pi y), \;\; cos(\pi x)] \\ p &= sin(2\pi x) \end{aligned} \tag{17}$$

we also get

$$\begin{aligned} f &= -\Delta u - \nabla p \\ &= \left[2\pi cos(2\pi x) - \pi^2 sin(\pi y), \;\; -\pi^2 cos(\pi x)\right] \end{aligned} \tag{18}$$

To calculate the wall shear stress we use Cauchy's stress tensor

$$\tau = -pI + \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{19}$$

To calculate the stress force we use

$$F_{shear} = \vec{n} \cdot \tau \cdot \vec{n_t} \tag{20}$$

where $\vec{n}$ is the respective walls normal vector and $\vec{n_t}$ is its tangential vector. In figure 2 we can see the error in the wall shear force for the lower wall (y=0).

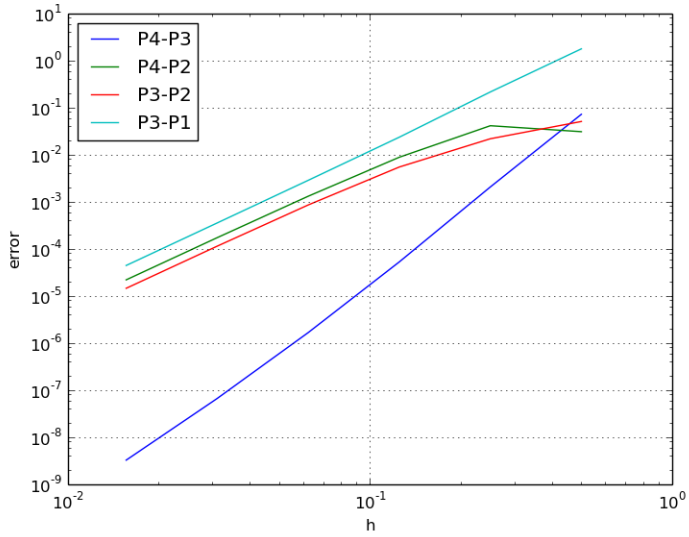

Figure 2: Figure displays error of the wall shear force against the lower wall (y=0) against mesh size h.

In table 5 we see the convergence rates for the wall shear force.

The convergence rates goes towards the same as in the previous exercise except for the P3-P1 elements. Not sure why this is but for wall shear stress the change in velocity is the dominant term in the stress tensor. Maybe this makes does has an impact when we get into linear elements. This should be examined more at another time.

Table 5: Convergence rate, α_i, for the wall shear stress over the lower wall (y=0). As i increases the mesh is finer.

	Convergence rate			
α	P4-P3	P4-P2	P3-P2	P3-P1
α_1	5.1279	-0.4209	1.2186	3.0412
α_2	5.2881	2.2238	1.9942	3.1922
α_3	4.9845	2.7499	2.6755	3.0419
α_4	4.6502	2.9249	2.9082	3.0087
α_5	4.3539	2.9770	2.9741	3.0024

Exercise 2

Mathematical model

The equation set we are to solve are

$$\begin{aligned} -\mu\Delta u - \lambda\nabla\nabla\cdot u &= f \qquad \text{in} \quad \Omega = (0,1)^2 \\ u &= u_e \qquad \text{on} \quad \partial\Omega \end{aligned} \tag{21}$$

with $u_e = \left(\frac{\partial\phi}{\partial y}, -\frac{\partial\phi}{\partial x}\right)$ where $\phi = sin(\pi xy)$. This gives by construction $\nabla \cdot u_e = 0$.

Part a

Calculating $u_e = (\pi x cos(\pi xy), -\pi y cos(\pi xy))$ and inserting in equation 21 with $\nabla \cdot u_e = 0$, gives

$$\begin{aligned} f &= -\mu\Delta u_e - \lambda\nabla\nabla\cdot u_e \\ &= -\mu\Delta u_e - 0 \\ &= \mu\left(2\pi^2 y sin(\pi xy) + \pi^3 x cos(\pi xy)(x^2+y^2)\right)\vec{i} \\ &\quad + \mu\left(2\pi^2 x sin(\pi xy) + \pi^3 y cos(\pi xy)(x^2+y^2)\right)\vec{j} \end{aligned} \tag{22}$$

Part b and c

Running a straight forward implementation of the equation set and solving for u gives the results seen in table 6 and 7.

We see that for both 1st and 2nd order polynomials the convergence rate is not what we want. This is because locking is occurring. Locking means that we get less deformation in our numerical calculations that what we

Table 6: Error and convergence rate for 1st order polynomials.

	Numerical error				
λ \ N	8	16	32	64	Convergence rate
1	0.0712804	0.0185788	0.00469644	0.00117743	1.99593
100	0.299714	0.164058	0.0607608	0.0176536	1.78318
10000	0.444626	0.456282	0.432982	0.351944	0.298963

Table 7: Error and convergence rate for 2nd order polynomials.

	Numerical error				
λ \ N	8	16	32	64	Convergence rate
1	0.00208792	0.000252388	3.12521e-05	3.89715e-06	3.00346
100	0.0144104	0.00149903	0.000119506	8.74676e-06	3.77219
10000	0.0299025	0.00717555	0.00157727	0.0002722	2.53469

get in real life. This problem can be fixed by implementing a new function $p = \lambda \nabla \cdot u$ and solving the system with a mixed space function. This gives us a new set of equations which written on a weak form looks like

$$\begin{aligned} -\mu \Delta u - \nabla p &= f \\ \nabla \cdot u &= 0 \end{aligned} \tag{23}$$

Solving this set, now with a 2nd order polynomial for u and a 1st order polynomial for p, gives us the results seen in table 8.

This gives us a good convergence rate for all λ.

Table 8: Error and convergence rates with P2 elements for u and P1 elements for p.

	Numerical error				
λ \ N	8	16	32	64	Convergence rate
1	0.00198965	0.000248935	3.11405e-05	3.89363e-06	2.99961
100	0.00197179	0.000248309	3.1121e-05	3.89305e-06	2.99892
10000	0.00197181	0.00024831	3.1121e-05	3.89305e-06	2.99892

Code

Exercise 7.6

```
from dolfin import *
import matplotlib.pyplot as plt
import numpy as np
set_log_active(False)

N = [2, 4, 8, 16, 32, 64]
u_order = [4, 4, 3, 3]
p_order = [3, 2, 2, 1]

for o in range(len(u_order)):
    error_u = np.zeros(len(N))
    error_p = np.zeros(len(N))
    h = np.zeros(len(N))
    print ''
    print 'P%s-P%s' %(u_order[o], p_order[o])
    print '-'*80
    for i in range(len(N)):
        mesh = UnitSquareMesh(N[i], N[i])

        V = VectorFunctionSpace(mesh, 'CG', u_order[o])
        V2 = VectorFunctionSpace(mesh, 'CG', u_order[o]+1)
        Q = FunctionSpace(mesh, 'CG', p_order[o])
        Q2 = FunctionSpace(mesh, 'CG', p_order[o]+1)

        VQ = MixedFunctionSpace([V, Q])

        class Left(SubDomain):
```

```
    def inside(self, x, on_boundary):
        return near(x[0], 0) and on_boundary

class velocity_boundaries(SubDomain):
    def inside(self, x, on_boundary):
        return x[0] < DOLFIN_EPS \
               or x[1] > 1 - DOLFIN_EPS \
               or x[1] < DOLFIN_EPS

boundaries = FacetFunction('size_t', mesh)
boundaries.set_all(0)

vel_bound = velocity_boundaries()
vel_bound.mark(boundaries, 1)
#plot(boundaries)

u_exact = Expression(('sin(pi*x[1])', 'cos(pi*x[0])'))
p_exact = Expression('sin(2*pi*x[0])')

f = Expression(('pi*pi*sin(pi*x[1]) - 2*pi*cos(2*pi*x[0])', 'pi*pi


noslip = DirichletBC(VQ.sub(0), u_exact, boundaries, 1)
bc = [noslip]


up = TrialFunction(VQ)
u, p = split(up)
vq = TestFunction(VQ)
v, q = split(vq)

eq1 = inner(grad(u), grad(v))*dx + inner(p, div(v))*dx \
      - inner(f,v)*dx
eq2 = inner(div(u), q)*dx

equation = eq1 + eq2

up_ = Function(VQ)

solve(lhs(equation) == rhs(equation), up_, bc)
```

```
    u_, p_ = up_.split()

    u_exact = interpolate(u_exact, V2)
    p_exact = interpolate(p_exact, Q2)

    error_u[i] = errornorm(u_exact, u_, 'h1')
    error_p[i] = errornorm(p_exact, p_)
    h[i] = 1./N[i]

print 'Velocity:'
for i in range(len(N)):
    convergence = 0
    if i == 0:
        print 'N = ',N[i],', H1 error = ',error_u[i]
    if i > 0:
        convergence = np.log(abs(error_u[i]/error_u[i-1])) \
                      / np.log(abs(h[i] / h[i-1]))
        print 'N = ',N[i],', H1 error = ',error_u[i],', convergence rate

print ''
print 'Pressure:'
for i in range(len(N)):
    convergence = 0
    if i == 0:
        print 'N = ',N[i],', L2 error = ',error_p[i]
    if i > 0:
        convergence = np.log(abs(error_p[i]/error_p[i-1])) \
                      / np.log(abs(h[i] / h[i-1]))
        print 'N = ',N[i],', L2 error = ',error_p[i],', convergence rate

plt.figure(1)
plt.loglog(h, error_u, label='P%s-P%s'%(u_order[o] ,p_order[o]))
plt.title('H1 error for u')
plt.xlabel('h'), plt.ylabel('error')
plt.grid('on')
plt.legend(loc='lower right')
plt.figure(2)
plt.loglog(h, error_p, label='P%s-P%s'%(u_order[o] ,p_order[o]))
plt.title('L2 error for p')
```

```
    plt.xlabel('h'), plt.ylabel('error')
    plt.grid('on')
    plt.legend(loc='lower right')
plt.xlabel('h'), plt.ylabel('error')
plt.show()
```

Exercise 7.7

```
from dolfin import *
import matplotlib.pyplot as plt
import numpy as np
set_log_active(False)


N = [2, 4, 8, 16, 32, 64]
h = [1./i for i in N]
error = np.zeros(len(N))
u_pol = [4, 4, 3, 3]
p_pol = [3, 2, 2, 1]

for deg in range(len(u_pol)):
    print ''
    print 'P%s-P%s' %(u_pol[deg], p_pol[deg])
    print '-'*80
    for i in range(len(N)):
        mesh = UnitSquareMesh(N[i], N[i])
        V = VectorFunctionSpace(mesh, 'CG', u_pol[deg])
        V2 = VectorFunctionSpace(mesh, 'CG', u_pol[deg]+1)
        Q = FunctionSpace(mesh, 'CG', p_pol[deg])
        Q2 = FunctionSpace(mesh, 'CG', p_pol[deg]+1)

        VQ = MixedFunctionSpace([V, Q])

        up = TrialFunction(VQ)
        u, p = split(up)
        vq = TestFunction(VQ)
        v, q = split(vq)

        u_ex = Expression(('sin(pi*x[1])', 'cos(pi*x[0])'))
```

```
p_ex = Expression('sin(2*pi*x[0])')

boundaries = FacetFunction('size_t', mesh)
boundaries.set_all(0)

class left(SubDomain):
    def inside(self, x, on_boundary):
        return x[0] < DOLFIN_EPS

class right(SubDomain):
    def inside(self, x, on_boundary):
        return x[0] > 1.0 - DOLFIN_EPS

class bottom(SubDomain):
    def inside(self, x, on_boundary):
        return x[1] < DOLFIN_EPS

class top(SubDomain):
    def inside(self, x, on_boundary):
        return x[1] > 1.0 - DOLFIN_EPS

left = left()
left.mark(boundaries, 1)
right = right()
right.mark(boundaries, 2)
bottom = bottom()
bottom.mark(boundaries, 3)
top = top()
top.mark(boundaries, 4)
#plot(boundaries, interactive=True)

left_vel = DirichletBC(VQ.sub(0), u_ex, left)
bottom_vel = DirichletBC(VQ.sub(0), u_ex, bottom)
top_vel = DirichletBC(VQ.sub(0), u_ex, top)
outlet_pressure = DirichletBC(VQ.sub(1), p_ex, right)

bc = [left_vel, bottom_vel, top_vel, outlet_pressure]

f = Expression(('2*pi*cos(2*pi*x[0]) - pi*pi*sin(pi*x[1])', '-pi*p
```

```
        eq1 = inner(grad(u), grad(v))*dx + p*div(v)*dx + inner(f, v)*dx
        eq2 = div(u)*q*dx

        equation = eq1-eq2

        up_ = Function(VQ)
        solve(lhs(equation) == rhs(equation), up_, bc)

        u_, p_ = split(up_)

        # cauchy stress tensor numerical
        tau = -p_*Identity(2) + 0.5*(grad(u_) + grad(u_).T)
        n = FacetNormal(mesh)
        ds = Measure('ds', subdomain_data=boundaries)
        stress = dot(tau, n)

        # cauchy stress tensor exact
        uex = interpolate(u_ex, V2)
        pex = interpolate(p_ex, Q2)
        tau_ex = -pex*Identity(2) + 0.5*(grad(uex) + grad(uex).T)
        stress_ex = dot(tau_ex, n)

        error[i] = assemble((stress_ex[0]-stress[0])**2*ds(1))
        error[i] =  sqrt(error[i])

    for i in range(len(N)):
        if i == 0:
            print 'N = ',N[i],', error = ',error[i]
        else:
            convergence = np.log(error[i]/error[i-1]) \
                        / np.log(h[i]/h[i-1])
            print 'N = ',N[i],', error = ',error[i],', convergence rate = ',c

    plt.loglog(h, error, label='P%s-P%s'%(u_pol[deg], p_pol[deg]))
plt.xlabel('h'), plt.ylabel('error')
plt.grid('on')
plt.legend(loc='upper left')
plt.show()
```

Exercise 2

```
from dolfin import *
import numpy as np

set_log_level(ERROR)

mu = Constant(1.0)

def functionspace(N, Lambda, Degree):
    print 'Single functionspace'
    print '----------------------'
    print 'numerical error for:'
    for deg in Degree:
        print ''
        print 'Degree of polynomial:',deg
        for lmbda in Lambda:
            error = np.zeros(len(N))
            h = np.zeros(len(N))
            for n in range(len(N)):
                mesh = UnitSquareMesh(N[n], N[n])

                V = VectorFunctionSpace(mesh, 'CG', deg)
                V2 = VectorFunctionSpace(mesh, 'CG', deg+1)

                f = Expression(('mu*pi*pi*(2*x[1]*sin(pi*x[0]*x[1]) + pi*
                                '-mu*pi*pi*(2*x[0]*sin(pi*x[0]*x[1]) + pi

                ue = Expression(('pi*x[0]*cos(pi*x[0]*x[1])', '-pi*x[1]*co
                ue = interpolate(ue, V2)
                uE = project(ue, V)
                class Boundaries(SubDomain):
                    def inside(self, x, on_boundary):
                        return on_boundary

                boundaries = Boundaries()

                bc = DirichletBC(V, ue, boundaries)
```

```
                mf = FacetFunction('size_t', mesh)
                mf.set_all(0)
                boundaries.mark(mf, 2)
                #plot(mf, interactive=True)

                u = TrialFunction(V)
                v = TestFunction(V)

                a = mu*inner(grad(u), grad(v))*dx \
                   + lmbda*inner(div(u), div(v))*dx
                L = dot(f, v)*dx

                u_ = Function(V)
                solve(a == L, u_, bc)

                error[n] = errornorm(u_, ue)#, norm_type='l2')
                h[n] = 1./N[n]
                if n == 0:
                    print 'Lambda=',lmbda,', n=',N[n],', error=',error[n]
                else:
                    convergence_rate = np.log(abs(error[n]/error[n-1]))/np
                    print 'Lambda=',lmbda,', n=',N[n],', error=',error[n],

            #plot(u_)
            #plot(uE)
            #interactive(True)
            print ''


def mixedfunctionspace(N, Lambda, Degree):
    print 'Mixed functionspace'
    print '--------------------'
    print 'numerical error for:'
    for deg in Degree:
        print ''
        print 'Degree of polynomial:',deg
        for lmbda in Lambda:
            error = np.zeros(len(N))
            h = np.zeros(len(N))
            for n in range(len(N)):
```

```
mesh = UnitSquareMesh(N[n], N[n])

V = VectorFunctionSpace(mesh, 'CG', deg)
V2 = VectorFunctionSpace(mesh, 'CG', deg+1)
P = FunctionSpace(mesh, 'CG', 1)
W = MixedFunctionSpace([V, P])

up = Function(W)
vq = TestFunction(W)
u, p = split(up)
v, q = split(vq)

f = Expression(('mu*pi*pi*(2*x[1]*sin(pi*x[0]*x[1]) + pi*
                '-mu*pi*pi*(2*x[0]*sin(pi*x[0]*x[1]) + pi

uex = Expression(('pi*x[0]*cos(pi*x[0]*x[1])', '-pi*x[1]*c
ue = interpolate(uex, V2)

class Boundaries(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary

boundaries = Boundaries()

bc = DirichletBC(W.sub(0), uex, boundaries)

mf = FacetFunction('size_t', mesh)
mf.set_all(0)
boundaries.mark(mf, 2)
#plot(mf, interactive=True)

eq1 = mu*inner(grad(u), grad(v))*dx + \
      Constant(lmbda)*inner(p, div(v))*dx - inner(f,v)*dx
eq2 = 1./Constant(lmbda)*inner(p, q)*dx - \
       inner(div(u), q)*dx

eq = eq1 - eq2

solve(eq == 0, up, bc)
u, p = up.split()
```

```
                error[n] = errornorm(u, ue)
                h[n] = 1./N[n]
                convergence_rate = 0
                if n == 0:
                    print 'Lambda=',lmbda,', n=',N[n],', error=',error[n]
                else:
                    convergence_rate = np.log(abs(error[n]/error[n-1]))/np
                    print 'Lambda=',lmbda,', n=',N[n],', error=',error[n],

            #plot(u)
            #plot(ue)
            #interactive(True)
            print ''

N = [8, 16, 32, 64]
Lambda = [1, 100, 10000]
Degrees = [1, 2]

functionspace(N, Lambda, Degrees)
mixedfunctionspace(N, Lambda, Degrees)
```