Oblig 1

Greger Svenn

March 17, 2016

Code and report can also be found on GitHub:

https://github.com/andergsv/MEK4250.git

Ex. 1

a)

Given

$$u = cos(k\pi x)sin(l\pi y) \tag{1}$$

we can compute the H^p norm using definition 2.13 from the lecture notes that states

$$||u||_{H^p} = \sqrt{\sum_{|\alpha|\leq p} \int_\Omega |\partial^\alpha u|^2 dx} \equiv \sqrt{\sum_{|\alpha|\leq p} ||\partial^\alpha u||^2_{L^2(\Omega)}} \tag{2}$$

The L^2 norm of the weak derivative. Since a strong derivative of u exists, Lemma 2.2 says this is a weak derivative.

$$||\frac{\partial^{i+j}u}{\partial x^i \partial y^j}||^2_{L^2} = (k\pi)^{2i}(l\pi)^{2j} \int_{x=0}^1 \int_{y=0}^1 f(x)^2 g(y)^2 dydx \tag{3}$$

where $f(x)$ and $g(y)$ is either $cos(...)$ or $sin(...)$ depending on the derivative. Then we either get

$$\begin{aligned} \int_0^1 sin^2(k\pi x)dx &= \frac{x}{2} - \frac{sin(2k\pi x}{4\pi k}\bigg|_0^1 = \frac{1}{2} \\ \int_0^1 cos^2(k\pi x)dx &= \frac{x}{2} + \frac{sin(2k\pi x}{4\pi k}\bigg|_0^1 = \frac{1}{2} \end{aligned} \tag{4}$$

Same goes for y, and we get

$$||\frac{\partial^{i+j}u}{\partial x^i \partial y^j}||^2_{L^2} = \frac{1}{4}(k\pi)^{2i}(l\pi)^{2j} \tag{5}$$

Results from (5) into (2)

$$||u||_{H^p} = \frac{1}{2}\sum_{i=0}^{p}\sum_{j=0}^{p-i}(k\pi)^i(l\pi)^j \tag{6}$$

b)

Numerical error, P1:

h	l	k	L^2	H^1
8	1	1	0.03278	0.43659
		10	0.67903	16.40373
		100	193.30469	2769.29874
	10	1	0.67979	16.14985
		10	0.66706	26.48145
		100	43.85056	1081.55841
	100	1	191.31130	2760.51942
		10	39.00197	1062.02533
		100	159.35642	3226.28568
16	1	1	0.00846	0.21817
		10	0.24096	9.09787
		100	262.27749	3507.09006
	10	1	0.24528	9.17928
		10	0.36554	17.54645
		100	23.04633	873.59319
	100	1	262.03735	3505.99874
		10	22.10786	871.93565
		100	246.86184	4686.20099
32	1	1	0.00213	0.10906
		10	0.07831	4.60585
		100	2.93137	281.67015
	10	1	0.07865	4.62356
		10	0.17819	10.60237
		100	2.44710	266.89592
	100	1	3.07984	281.57721
		10	2.65025	269.12035
		100	2.69686	376.36441
64	1	1	0.00053	0.05452
		10	0.02092	2.28079
		100	4.68789	433.69968
	10	1	0.02091	2.28339
		10	0.05490	5.43986
		100	4.01373	405.62311
	100	1	4.68872	433.25750
		10	4.01968	405.33294
		100	3.58881	540.46687

Numerical error, P2:

h	l	k	L^2	H^1
8	1	1	0.00057	0.03318
		10	0.33168	8.91002
		100	289.69571	3780.34116
	10	1	0.32986	9.01115
		10	0.43561	19.12452
		100	32.02358	1119.41619
	100	1	289.59195	3779.65941
		10	32.74071	1121.76166
		100	293.24615	5321.28087
16	1	1	0.00007	0.00839
		10	0.02528	2.18345
		100	91.92350	1219.85991
	10	1	0.02531	2.20724
		10	0.08960	6.92035
		100	9.07023	383.82845
	100	1	91.89877	1219.79809
		10	8.70218	386.28122
		100	90.47492	1648.49621
32	1	1	0.00001	0.00211
		10	0.00288	0.56873
		100	5.90687	556.63276
	10	1	0.00289	0.57230
		10	0.01021	1.97795
		100	5.12436	521.09486
	100	1	5.90325	556.31812
		10	5.12085	520.79710
		100	4.72230	689.09240
64	1	1	0.00000	0.00058
		10	0.00035	0.14421
		100	1.80292	186.50476
	10	1	0.00035	0.14469
		10	0.00114	0.51839
		100	1.58037	178.85308
	100	1	1.80305	186.59945
		10	1.57998	178.94446
		100	1.47096	288.59719

c)

Considering

$$||u - u_h||_1 \leq C_\alpha h^\alpha \tag{7}$$

and

$$||u - u_h||_0 \leq C_\beta h^\beta \tag{8}$$

The expected convergence rate for L^2 error is 2 for for first order polynomials and 3 for second order. For H^1 we expect 1 for P1 and 2 for P2.
. Rewriting (7) and (8) as follows

$$log(||u - u_h||_p) = log(C) + \alpha log(h) \tag{9}$$

we can find the convergence rates, β and α, and C using least square method.

I've got the following estimates for C_α, C_β, α and β for all k and l

error norm		β	C_β
L^2	P1	1.77219875142	199.641306082
	P2	2.57094220334	393.651742315

error norm		α	C_α
H^1	P1	0.922950793267	892.842058168
	P2	1.49059681803	1512.36220975

For $k = l = 1$:

error norm		β	C_β
L^2	P1	1.98036792982	1.0223300868
	P2	3.0153973461	0.105098152287

error norm		α	C_α
H^1	P1	1.00043640421	2.47126276095
	P2	1.9922681715	1.05057103854

The convergence rates when $k = l = 1$ is as expected. For frequencies lager than number of elements we get bad results.

Ex. 2

a)

$$-\mu\Delta u + u_x = 0 \tag{10}$$

$$u(0,y) = 0, u(1,y) = 1 \tag{11}$$

$$\frac{\partial u(x,0)}{\partial n} = \frac{\partial u(x,1)}{\partial n} = 0 \tag{12}$$

(10) can be solved by separation of variables, $u(x,y) = X(x)Y(y)$:

$$\begin{aligned} -\mu X''Y - \mu XY'' + X'Y = 0 \\ \frac{X''}{X} - \frac{1}{\mu}\frac{X'}{X} = -\frac{Y''}{Y} = \lambda \end{aligned} \tag{13}$$

Solving $Y'' + \lambda Y = 0$ with boundary conditions, gives us the following solution

$$Y(y) = A\cos(\sqrt{\lambda}x) \tag{14}$$

If (14) are to fulfil the BC at $y = 1$, $\sqrt{\lambda}$ needs to be of the form $n\pi$ where $n = 0, 1, 2...$
By choosing $\sqrt{\lambda} = 0$, we get

$$Y = A \tag{15}$$

Since Y is a constant our problem is now reduced to an ODE of the form

$$X'' - \frac{1}{\mu} = 0 \tag{16}$$

which has the solution

$$X(x) = \frac{1 - e^{\frac{x}{\mu}}}{1 - e^{\frac{1}{\mu}}} \tag{17}$$

Now $u = AX(x)$. Combining (17) and (11) we get $A = 1$. This means

$$u = X(x) = \frac{1 - e^{\frac{x}{\mu}}}{1 - e^{\frac{1}{\mu}}} \tag{18}$$

b), c)

μ	h	L2	H1	α_{L2}	α_{H1}
1.0	8	0.00140247771262	0.0375213286623	1.99975900004	0.999843163008
	16	0.000350756920914	0.0187654641213		
	32	8.76983292529e-05	0.00938336199838		
	64	2.1925161506e-05	0.00469176097199		
0.1	8	0.0237537287128	0.767085862122	1.97534705635	0.97703269627
	16	0.0061772884875	0.398103773843		
	32	0.00156135223788	0.201040560876		
	64	0.000391472930963	0.100776905796		
0.01	8	0.23793404436	7.23834618036	1.464865403	0.427313340823
	16	0.103935591962	6.68437984751		
	32	0.0381861200365	5.00716129578		
	64	0.0112593871861	2.969491496		
0.0015	8	0.973209783522	25.4293684955	1.23061747096	0.149569580962
	16	0.343073702629	19.3988155863		
	32	0.15386822893	18.2879241561		
	64	0.0740360746411	18.3564801216		
0.001	-	NaN	NaN	NaN	NaN
0.0001	-	NaN	NaN	NaN	NaN

We can see, from the table above, that we get good results for large μ.
As $\mu \to 0$ the results gets worse.

d)

When introducing false diffusion as $\mu \to 0$ using the *Streamkine diffusion/Petrov-Galerkin* method, we now expect a convergence rate of 0.5 for L^2 and 1.5 for H^1.

μ	h	L2	H1	L2 α	H1 α
1.0	8	0.00817327386546	0.0445180433635	1.02400069631	0.98816006649
	16	0.00396967669845	0.0225772790049		
	32	0.00195639004967	0.0113698584674		
	64	0.000971209372729	0.00570543611502		
0.1	8	0.116799775258	1.0067518284	0.928034747153	0.808047892246
	16	0.0633185477355	0.622054684915		
	32	0.0331298022384	0.351779744504		
	64	0.0169821128053	0.188199945818		
0.01	8	0.190538358416	4.58810309565	0.690381217075	0.117182252988
	16	0.129228319526	5.39419191334		
	32	0.0795713397148	4.89193806113		
	64	0.0454392595226	3.61573898911		
0.0015	8	0.184075039798	4.6876624363	0.48364535762	-0.480191675456
	16	0.131288176206	6.63832047093		
	32	0.0943653123066	9.39242690883		
	64	0.0672192167908	12.6635884144		

The convergence is as expected for small μ for L2, not for H1. Not sure why.

Code

```
from dolfin import *

set_log_active(False)
import numpy as np


def Hp(k, l, p):
  s = 0
  for i in range(p+1):
    for j in range(p+1-i):
      s += 0.5 * (k*np.pi)**i * (l*np.pi)**j
  return s

def bc1(x,on_boundary):
  return (near(x[0], 0) or near(x[0], 1)) and on_boundary


k1 = [1, 10, 100]
l1 = [1, 10, 100]
h1 = [8, 16, 32, 64]
Pe = [1,2]

H1 = []
L2 = []

for p in Pe:
  bd = open('table%s.txt' %p,'w')
  bd.write('\\begin{tabular}{ |c|c|c|c|c| }\n')
  bd.write('\hline%cline{1-1} \cline{4-5}\n')
  bd.write('h & l & k &$L^2$&$H^1$ \\\\\n')
  bd.write('\hline \n')

  H = []
  L = []
  meshsize = []
  for h in h1:
    bd.write('\multirow{9}{*}{\\textbf{%s}}' %h)
    for l in l1:
      bd.write(' & \multirow{3}{*}{%s} ' %l)
      for k in k1:

```

```
      mesh = UnitSquareMesh(h, h)
      meshsize.append(mesh.hmin())

      V = FunctionSpace(mesh, 'CG', p)
      V2 = FunctionSpace(mesh, 'CG', p+2)

      bc = DirichletBC(V, Constant(0), bc1)
      u = TrialFunction(V)
      v = TestFunction(V)

      f = Expression( 'pi*pi * sin(k*pi*x[0])*cos(l*pi*x[1])*(k*k + l*l)', k = k, l = l)

      F = inner(grad(u), grad(v))*dx - f*v*dx

      u_ = Function(V)

      solve(lhs(F) == rhs(F), u_, bc)

      u_ex = Expression('sin(k*pi*x[0])*cos(l*pi*x[1])',k = k, l = l)
      u_e = interpolate(u_ex, V2)

      ud= abs(u_ - u_e)

      #L2_norm = errornorm(u_, u_e)
      L2_norm = sqrt(assemble(ud**2*dx))

      #H1_norm= errornorm(u_, u_e, 'H1')
      H1_norm = sqrt(assemble(ud*ud*dx+ inner(grad(ud), grad(ud))*dx))

      H.append(H1_norm)
      L.append(L2_norm)


      if k ==1:
        bd.write('& %s    & %0.5f   & %0.5f  \\\\ \\cline{3-5}\n' %(k, L2_norm, H1_norm))
      elif k ==100:
        bd.write('&& %s    & %0.5f   & %0.5f  \\\\ \\cline{2-5}\n' %(k, L2_norm, H1_norm))
      else:
        bd.write('&& %s    & %0.5f   & %0.5f  \\\\ \\cline{3-5}\n' %(k, L2_norm, H1_norm))

  bd.write('\\hline \\hline \n')
H1.append(H)
L2.append(L)
H1.append(meshsize)


bd.write('\\end{tabular}')
bd.close()
```

```
for j in [H1, L2]:

  for i in range(2):
    A = np.zeros((2,2))
    b = np.zeros(2)
    A[0][0] = len(H1[i*2])
    A[0][1] = sum(np.log(H1[2*i+1]))

    A[1][0] = sum(np.log(H1[2*i+1]))
    A[1][1] = sum(np.log(H1[2*i+1])**2)
    if j == L2:
      b[1] = sum(np.log(H1[2*i+1])*np.log(j[i]))
      b[0] = sum(np.log(j[i]))
    else:
      b[1] = sum(np.log(H1[2*i+1])*np.log(j[2*i]))
      b[0] = sum(np.log(j[2*i]))
    logC, alpha = np.linalg.solve(A,b)
    print 'alpha/beta:', alpha, 'C:', np.exp(logC)

#print A, hi
```

Ex1.py

```
from dolfin import *
set_log_active(False)
import numpy as np

def alpha(error, hval):
  A = np.zeros((2,2))
  b = np.zeros(2)
  A[0][0] = len(error)
  A[0][1] = sum(np.log(hval))

  A[1][0] = sum(np.log(hval))
  A[1][1] = sum(np.log(hval)**2)

  b[1] = sum(np.log(hval)*np.log(error))
  b[0] = sum(np.log(error))
  logC, alpha = np.linalg.solve(A,b)
  return alpha



def left(x,on_boundary):
  return near(x[0], 0) and on_boundary

def right(x, on_boundary):
  return near(x[0], 1) and on_boundary

mu1 = [1., 0.1, 0.01, 0.0015, 0.001, 0.0001]
h1 = [8, 16, 32, 64]
P = 1

PrintTexTable = False
task_d = True


if PrintTexTable == True:
  print '$\\mu$ & h  &   L2  &  H1  & L2 $\\alpha$ & H1 $\\
    alpha$\\\\ \hline'
#for h in h1:

for mu in mu1:
  t = 0
  H1_error = []
  L2_error = []
  hval = []
  for h in h1:

    mesh = UnitSquareMesh(h, h)

    beta = 0.5*mesh.hmin()

    V = FunctionSpace(mesh, 'CG', P)
    V2 = FunctionSpace(mesh, 'CG', P+2)

```

```
        bc = [DirichletBC(V, Constant(0), left), DirichletBC(V, Constant(1), right)]

        u = TrialFunction(V)
        v = TestFunction(V)
        L = v + beta * v.dx(0) #SUPG Testfunction

        f = Constant(0)

        u_ = Function(V)

        if task_d == True:
            FSD = mu*inner(grad(u), grad(L))*dx + u.dx(0)*L*dx -f*L*dx
            solve(lhs(FSD) == rhs(FSD), u_, bc)
        else:
            F = mu*inner(grad(u), grad(v))*dx + u.dx(0)*v*dx -f*v*dx
            solve(lhs(F) == rhs(F), u_, bc)

        u_ex = Expression('(1-exp(x[0]/mu))/(1-exp(1/mu))', mu = mu)
        u_e = interpolate(u_ex, V2)

        ud= abs(u_ - u_e)
        #print ud

        L2_norm = errornorm(u_, u_e)
        H1_norm= errornorm(u_, u_e, 'H1')


        H1_error.append(H1_norm)
        L2_error.append(L2_norm)
        hval.append(mesh.hmin())

        if PrintTexTable == True:
            if t==0:
                print '\multirow{4}{*} {%s} &' %mu,h,'&',L2_norm,' &', H1_norm,' & \multirow{6}{*} {L2} & \multirow{4}{*} {H1} \\\\ \cline{2-4}'
                t+= 1
            else:
                print '&',h,'&', L2_norm,'&', H1_norm, '&', '&', '\\\\ \cline{2-4}'

        #plot(u_, interactive=True)
    if PrintTexTable == True:
        print '  \hline \hline'

    else:

        L2alpha = alpha(L2_error, hval)
        H1alpha = alpha(H1_error, hval)
        print 'mu:', mu, '     L2alpha:', L2alpha, '     H1alpha:', H1alpha

```

```
    '''
    print mu
    for n in range(len(h1)):
        if n>0:
            print 'L2:', np.log(L2_error[n]/L2_error[n-1]) / np.log(hval[n]/hval[n-1])
            print 'H1:',np.log(H1_error[n]/H1_error[n-1]) / np.log(hval[n]/hval[n-1])
    '''



#plot(ud, interactive=True)
```

Ex2.py