Mandatory Assignment 1, MEK 4250

Jørgen D. Tyvand

March 17, 2016

Exercise 1

We are considering the problem

$-\Delta u = f$ in Ω

$u = 0$ for $x = 0$ and $x = 1$

$\frac{\partial u}{\partial n} = 0$ for $y = 0$ and $y = 1$

We are asked to compute $f = -\Delta u$, assuming $u = sin(\pi kx)cos(\pi ly)$. This is done as follows

$f = -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = (k\pi)^2 sin(\pi kx)cos(\pi ly) + (l\pi)^2 sin(\pi kx)cos(\pi ly) = ((k\pi)^2 + (l\pi)^2)u$

a)

I assume that the purpose of "Computing the H^p norm of u" suggests finding an analytical expression for H^p, rather than an actual computation with a program, as no value is given for p.

Looking at the general expression for the H^p norm, we have the following expression

$$H^p = \left[\int_\Omega \sum_{i \le p} |\nabla^i u|^2 \,\mathrm{d}x\right]^{\frac{1}{2}}$$

The term we want to look at is the increasing powers of the gradient/divergence term $\nabla^i u$. This will alternate between being a scalar and a vector, but the term is squared in the above expression, giving a scalar value. We get

$\nabla^0 u = u = sin(\pi kx)cos(\pi ly)$

$$\int_\Omega u^2 \,\mathrm{d}y\,\mathrm{d}x = \int_\Omega sin^2(\pi kx)cos^2(\pi ly)\,\mathrm{d}y\,\mathrm{d}x = \frac{1}{4}$$

$\nabla^1 u = [\pi k cos(\pi kx)cos(\pi ly), -\pi l sin(\pi kx)sin(\pi ly)]$

$$\int_\Omega (\nabla u)^2 \,\mathrm{d}y\,\mathrm{d}x = \int_\Omega \left[\pi^2 k^2 cos^2(\pi kx)cos^2(\pi ly) + \pi^2 l^2 sin^2(\pi kx)sin^2(\pi ly)\right] \mathrm{d}y\,\mathrm{d}x = \frac{\pi^2(k^2+l^2)}{4}$$

$\nabla^2 u = \nabla \cdot \nabla u = -\pi^2 k^2 sin(\pi kx)cos(\pi ly) - \pi^2 l^2 sin(\pi kx)cos(\pi ly) = -\pi^2(k^2+l^2)u$

$$\int_\Omega (\nabla^2 u)^2 \,\mathrm{d}y\,\mathrm{d}x = \pi^4(k^2+l^2)^2 \int_\Omega u^2 \,\mathrm{d}y\,\mathrm{d}x = \frac{\pi^4(k^2+l^2)^2}{4}$$

$\nabla^3 u = \nabla(\nabla^2 u) = [-(\pi^3)k(k^2+l^2)cos(\pi kx)cos(\pi ly), \pi^3 l(k^2+l^2)sin(\pi kx)sin(\pi ly)]$

$$\int_\Omega (\nabla^3 u)^2 \,\mathrm{d}y\,\mathrm{d}x = \pi^6(k^2+l^2)^3 \int_\Omega \left[cos^2(\pi kx)cos^2(\pi ly) + sin^2(\pi kx)sin^2(\pi ly)\right] \mathrm{d}y\,\mathrm{d}x$$

$$= \frac{\pi^6(k^2+l^2)^3}{4}$$

$$\nabla^4 u = \nabla \cdot \nabla^3 u = \pi^4 k^2(k^2+l^2)sin(\pi kx)cos(\pi ly) + \pi^4 l^2(k^2+l^2)sin(\pi kx)cos(\pi ly)$$

$$= \pi^4(k^2+l^2)^2 u$$

$$\int_\Omega (\nabla^4 u)^2 \,\mathrm{d}y\,\mathrm{d}x = \pi^8(k^2+l^2)^4 \int_\Omega u \,\mathrm{d}y\,\mathrm{d}x = \frac{\pi^8(k^2+l^2)^4}{4}$$

As we can see, there is a pattern to the value of the integral, and inserted into the expression for the H^p norm, we get

$$H^p = \frac{1}{2}\left[\sum_{i\leq p} \pi^{2i}(k^2+l^2)^i\right]^{\frac{1}{2}}$$

a)

The program for computing the L_2 and H^1 errors is listed below

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

k_vals = [1, 10, 100]
l_vals = [1, 10, 100]

for i in [1,2]:
        for h in [8, 16, 32, 64]:
                mesh = UnitSquareMesh(h,h)
                print('h = %d, P%d elements:\n' % (h,i))
                V = FunctionSpace(mesh, 'Lagrange', i)

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

                bc0 = DirichletBC(V, Constant(0.0), 'x[0] < DOLFIN_EPS')
                bc1 = DirichletBC(V, Constant(0.0), 'x[0] > 1-DOLFIN_EPS')
                bcs = [bc0, bc1]


                L2_errs = np.zeros((3,3))
                H1_errs = np.zeros((3,3))
                for k in range(3):
                        for l in range(3):

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

                                a = inner(grad(u),grad(v))*dx
                                L = f*v*dx

                                u_ = Function(V)

                                solve(a == L, u_, bcs)

                                u_exp = Expression('sin(pi*k*x[0])*cos(pi*l*x[1])',\
                                        k=k_vals[k],l=l_vals[l])
                                u_ex = interpolate(u_exp,V)

                                L2_errs[k][l] = errornorm(u_, u_ex, 'l2', degree_rise=3)
                                H1_errs[k][l] = errornorm(u_, u_ex, 'h1', degree_rise=3)
```

A printout of this program (not included above) gives the following results for the different combinations

```
h = 8, P1 elements:

L2 norm:

                l = 1           l = 10          l = 100
k =    1        0.032775        0.679790        191.311299
k =   10        0.679028        0.667057        39.001974
k =  100        193.304690      43.850560       159.356420


H1 norm:

                l = 1           l = 10          l = 100
k =    1        0.436592        16.149851       2760.519420
k =   10        16.403729       26.481453       1062.025335
k =  100        2769.298737     1081.558408     3226.285685


h = 8, P2 elements:

L2 norm:

                l = 1           l = 10          l = 100
k =    1        0.000569        0.329865        289.591947
k =   10        0.331676        0.435611        32.740710
k =  100        289.695710      32.023576       293.246151


H1 norm:

                l = 1           l = 10          l = 100
k =    1        0.033185        9.011149        3779.659405
k =   10        8.910024        19.124524       1121.761657
k =  100        3780.341160     1119.416192     5321.280870


h = 16, P1 elements:

L2 norm:

                l = 1           l = 10          l = 100
k =    1        0.008463        0.245283        262.037348
k =   10        0.240959        0.365538        22.107864
k =  100        262.277492      23.046334       246.861840


H1 norm:

                l = 1           l = 10          l = 100
k =    1        0.218166        9.179273        3505.998736
k =   10        9.097867        17.546447       871.935651
k =  100        3507.090063     873.593192      4686.200990


h = 16, P2 elements:

L2 norm:

                l = 1           l = 10          l = 100
k =    1        0.000069        0.025310        91.898771
k =   10        0.025276        0.089599        8.702181
k =  100        91.923500       9.070227        90.474916


H1 norm:

                l = 1           l = 10          l = 100
k =    1        0.008389        2.207236        1219.798089
k =   10        2.183449        6.920349        386.281221
k =  100        1219.859911     383.828456      1648.496211
```

```
h = 32, P1 elements:

L2 norm:

               l = 1          l = 10         l = 100
k =    1       0.002133       0.078653       3.079839
k =   10       0.078315       0.178190       2.650246
k =  100       2.931370       2.447102       2.696856


H1 norm:

               l = 1          l = 10         l = 100
k =    1       0.109055       4.623558       281.577212
k =   10       4.605848       10.602375      269.120343
k =  100       281.670147     266.895911     376.364409


h = 32, P2 elements:

L2 norm:

               l = 1          l = 10         l = 100
k =    1       0.000009       0.002889       5.903254
k =   10       0.002879       0.010206       5.120845
k =  100       5.906873       5.124357       4.722304


H1 norm:

               l = 1          l = 10         l = 100
k =    1       0.002106       0.572304       556.318121
k =   10       0.568733       1.977957       520.797098
k =  100       556.632761     521.094856     689.092405


h = 64, P1 elements:

L2 norm:

               l = 1          l = 10         l = 100
k =    1       0.000534       0.020911       4.688718
k =   10       0.020920       0.054904       4.019684
k =  100       4.687891       4.013733       3.588809


H1 norm:

               l = 1          l = 10         l = 100
k =    1       0.054524       2.283389       433.257497
k =   10       2.280790       5.439857       405.332937
k =  100       433.699674     405.623107     540.466888


h = 64, P2 elements:

L2 norm:

               l = 1          l = 10         l = 100
k =    1       0.000001       0.000351       1.803051
k =   10       0.000350       0.001140       1.579982
k =  100       1.802916       1.580365       1.470963


H1 norm:

               l = 1          l = 10         l = 100
k =    1       0.000527       0.144695       186.599439
k =   10       0.144221       0.518416       178.944455
k =  100       186.504757     178.853083     288.597177
```

c)

We start by rewriting the expression for the error estimate (using the first one given in the exercise as our example)

$||u - u_h||_1 \leq C_\alpha h^\alpha$

Assigning the constant A to the norm for brevity and taking the logarithm

$A \leq C_\alpha h^\alpha$

$ln(A) \leq ln(C_\alpha h^\alpha)$

$ln(A) \leq ln(C_\alpha) + \alpha ln(h)$

This is a linear equation $y = a + bx$ with $ln(C_\alpha)$ as the intercept and α as the slope.

The below program calculatates the values for α and C for both cases, and checks if the error estimate is valid

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

for k in [1, 10, 100]:
        for err_type in ['H1', 'L2']:
                for i in [1,2]:

                        x = []
                        y = []
                        for N in [8,16,32,64]:

                                mesh = UnitSquareMesh(N,N)

                                h = 1.0/N

                                V = FunctionSpace(mesh, 'Lagrange', i)

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

                                bc0 = DirichletBC(V, Constant(0.0), 'x[0] < DOLFIN_EPS')
                                bc1 = DirichletBC(V, Constant(0.0), 'x[0] > 1-DOLFIN_EPS')
                                bcs = [bc0, bc1]


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

                                a = inner(grad(u),grad(v))*dx
                                L = f*v*dx

                                u_ = Function(V)

                                solve(a == L, u_, bcs)

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

                                A = errornorm(u_ex, u_, err_type, degree_rise=3)

                                x.append(ln(h))
                                y.append(ln(A))


                        x = np.array(x)
                        y = np.array(y)

                        Arr = np.vstack([x, np.ones(len(x))]).T
                        alpha, lnC = np.linalg.lstsq(Arr, y)[0]
                        print('%s, k = %3d, P%d elements:' % (err_type, k, i))
                        print('alpha/beta = %.3f, C = %9.3f' % (alpha, exp(lnC)))
                        #print('alpha/beta = %.3f, C = %9.3f' % (alpha, lnC))
                        RHS = exp(lnC)*h**alpha
                        print('Errornorm = %.16f, Ch^alpha/beta = %.16f' % (A, RHS))
                        if A < RHS:
                                print('Error estimate is valid!\n')
                        else:
                                print('Error estimate is NOT valid!\n')
```

The values for the two cases and the result of error estimate validations are listed below

```
H1, k =   1, P1 elements:

alpha/beta = 1.000, C =     3.495

N = 8: Errornorm = 0.43659, Ch^alpha/beta = 0.43653
Error estimate is NOT valid for N = 8!

N = 16: Errornorm = 0.21817, Ch^alpha/beta = 0.21820
Error estimate is valid for N = 16!

N = 32: Errornorm = 0.10906, Ch^alpha/beta = 0.10907
Error estimate is valid for N = 32!

N = 64: Errornorm = 0.05452, Ch^alpha/beta = 0.05452
Error estimate is NOT valid for N = 64!


H1, k =   1, P2 elements:

alpha/beta = 1.992, C =     2.096

N = 8: Errornorm = 0.03318, Ch^alpha/beta = 0.03327
Error estimate is valid for N = 8!

N = 16: Errornorm = 0.00839, Ch^alpha/beta = 0.00836
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 0.00211, Ch^alpha/beta = 0.00210
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 0.00053, Ch^alpha/beta = 0.00053
Error estimate is valid for N = 64!


L2, k =   1, P1 elements:

alpha/beta = 1.980, C =     2.031

N = 8: Errornorm = 0.03278, Ch^alpha/beta = 0.03305
Error estimate is valid for N = 8!

N = 16: Errornorm = 0.00846, Ch^alpha/beta = 0.00838
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 0.00213, Ch^alpha/beta = 0.00212
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 0.00053, Ch^alpha/beta = 0.00054
Error estimate is valid for N = 64!


L2, k =   1, P2 elements:

alpha/beta = 3.015, C =     0.299

N = 8: Errornorm = 0.00057, Ch^alpha/beta = 0.00057
Error estimate is NOT valid for N = 8!

N = 16: Errornorm = 0.00007, Ch^alpha/beta = 0.00007
Error estimate is valid for N = 16!

N = 32: Errornorm = 0.00001, Ch^alpha/beta = 0.00001
Error estimate is valid for N = 32!

N = 64: Errornorm = 0.00000, Ch^alpha/beta = 0.00000
Error estimate is NOT valid for N = 64!
```

```
H1, k =  10, P1 elements:

alpha/beta = 0.758, C =   135.960

N = 8: Errornorm = 26.48145, Ch^alpha/beta = 28.12922
Error estimate is valid for N = 8!

N = 16: Errornorm = 17.54645, Ch^alpha/beta = 16.63692
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 10.60237, Ch^alpha/beta = 9.83984
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 5.43986, Ch^alpha/beta = 5.81973
Error estimate is valid for N = 64!


H1, k =  10, P2 elements:

alpha/beta = 1.742, C =   782.070

N = 8: Errornorm = 19.12452, Ch^alpha/beta = 20.88578
Error estimate is valid for N = 8!

N = 16: Errornorm = 6.92035, Ch^alpha/beta = 6.24289
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 1.97796, Ch^alpha/beta = 1.86604
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 0.51842, Ch^alpha/beta = 0.55777
Error estimate is valid for N = 64!


L2, k =  10, P1 elements:

alpha/beta = 1.185, C =     8.891

N = 8: Errornorm = 0.66706, Ch^alpha/beta = 0.75728
Error estimate is valid for N = 8!

N = 16: Errornorm = 0.36554, Ch^alpha/beta = 0.33318
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 0.17819, Ch^alpha/beta = 0.14659
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 0.05490, Ch^alpha/beta = 0.06450
Error estimate is valid for N = 64!


L2, k =  10, P2 elements:

alpha/beta = 2.887, C =   211.265

N = 8: Errornorm = 0.43561, Ch^alpha/beta = 0.52215
Error estimate is valid for N = 8!

N = 16: Errornorm = 0.08960, Ch^alpha/beta = 0.07060
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 0.01021, Ch^alpha/beta = 0.00955
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 0.00114, Ch^alpha/beta = 0.00129
Error estimate is valid for N = 64!
```

```
H1, k = 100, P1 elements:

alpha/beta = 1.137, C = 45954.859

N = 8: Errornorm = 3226.28569, Ch^alpha/beta = 4319.43930
Error estimate is valid for N = 8!

N = 16: Errornorm = 4686.20099, Ch^alpha/beta = 1963.93007
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 376.36441, Ch^alpha/beta = 892.94491
Error estimate is valid for N = 32!

N = 64: Errornorm = 540.46689, Ch^alpha/beta = 405.99745
Error estimate is NOT valid for N = 64!


H1, k = 100, P2 elements:

alpha/beta = 1.387, C = 87018.317

N = 8: Errornorm = 5321.28087, Ch^alpha/beta = 4862.00991
Error estimate is NOT valid for N = 8!

N = 16: Errornorm = 1648.49621, Ch^alpha/beta = 1858.73535
Error estimate is valid for N = 16!

N = 32: Errornorm = 689.09241, Ch^alpha/beta = 710.59031
Error estimate is valid for N = 32!

N = 64: Errornorm = 288.59718, Ch^alpha/beta = 271.65706
Error estimate is NOT valid for N = 64!


L2, k = 100, P1 elements:

alpha/beta = 2.293, C = 31760.852

N = 8: Errornorm = 159.35642, Ch^alpha/beta = 269.60999
Error estimate is valid for N = 8!

N = 16: Errornorm = 246.86184, Ch^alpha/beta = 54.99847
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 2.69686, Ch^alpha/beta = 11.21929
Error estimate is valid for N = 32!

N = 64: Errornorm = 3.58881, Ch^alpha/beta = 2.28865
Error estimate is NOT valid for N = 64!


L2, k = 100, P2 elements:

alpha/beta = 2.718, C = 99528.778

N = 8: Errornorm = 293.24615, Ch^alpha/beta = 349.59800
Error estimate is valid for N = 8!

N = 16: Errornorm = 90.47492, Ch^alpha/beta = 53.14255
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 4.72230, Ch^alpha/beta = 8.07822
Error estimate is valid for N = 32!

N = 64: Errornorm = 1.47096, Ch^alpha/beta = 1.22797
Error estimate is NOT valid for N = 64!
```

Exercise 2

The problem in this exercise is

$$-\mu\Delta u + u_x = 0 \text{ in } \Omega$$

$$u = 0 \text{ for } x = 0 \qquad\qquad u = 1 \text{ for } x = 1$$

$$\frac{\partial u}{\partial n} = 0 \text{ for } y = 0 \text{ and } y = 1$$

a)

We wish to derive an expression for the analytical solution. We assume that the solution is a product of two functions dependent on only one parameter, so that we can use separation of variables

$$u = X(x)Y(x)$$

Inserting this into the problem equation, we get

$$-\mu(X''Y'' + XY'') + X'Y = 0$$

Dividing by XY we get

$$-\mu\left(\frac{X''}{Y} + \frac{Y''}{Y}\right) + \frac{X'}{X} = 0$$

We see that we can get two expressions that are only dependent on x or y, so these can be said to be equal to a common constant λ. We start by looking at the expression for $Y(y)$

$$\mu\frac{Y''}{Y} = -\lambda^2$$

$$\mu Y'' - \lambda Y = 0$$

$$Y'' - \frac{\lambda}{\mu}Y = 0$$

The solution of this equation is $Y(y) = Acos\left(\frac{\lambda}{\sqrt{\mu}}y\right) + Bsin\left(\frac{\lambda}{\sqrt{\mu}}y\right)$

The derivative is $Y'(y) = -\frac{A\lambda}{\sqrt{\mu}}sin\left(\frac{\lambda}{\sqrt{\mu}}y\right) + \frac{B\lambda}{\sqrt{\mu}}cos\left(\frac{\lambda}{\sqrt{\mu}}y\right)$

Using the boundary conditions, we see that both constants become zero if $\lambda \neq 0$. Therefore $Y(y) = C$, and we can set $Y(y) = 1$ for convenience.

Next we look at the equation for X, using the above value $\lambda = 0$

$$-\mu \frac{X''}{X} + \frac{X'}{X} = 0$$

$$-muX'' + X' = 0$$

Using the boundary conditions, and writing out for $u(x)$ seeing that $Y(y) = 1$, we get the solution (similar to the one given in the beginning of chapter 6 of the lecture notes)

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

b)

The following program computes the errors for $mu = 1, 0.1, 0.01, 0.001, 0.0001$ at N = 8, 16, 32 and 64

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

muvals = [1., 0.1, 0.01]
nvals = [8, 16, 32, 64]
for i in [1,2]:
        for err_type in ['H1', 'L2']:
                print('     %s error for P%d elements\n' % (err_type, i))

                errs = np.zeros((3,4))

                print('                      N = 8              N = 16             N = 32             N = 64')
                ncount = 0
                for N in nvals:

                        mesh = UnitSquareMesh(N,N)
                        V = FunctionSpace(mesh, 'Lagrange', i)
                        V_1 = FunctionSpace(mesh, 'Lagrange', i+2)

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

                        bc0 = DirichletBC(V, Constant(0), 'x[0] < DOLFIN_EPS')
                        bc1 = DirichletBC(V, Constant(1), 'x[0] > 1 - DOLFIN_EPS')
                        bcs = [bc0, bc1]

                        u_ = Function(V)

                        f = Constant(0.0)

                        mucount = 0
                        for mu in muvals:
                                a = mu*inner(grad(u),grad(v))*dx + u.dx(0)*v*dx
                                L = f*v*dx

                                solve(a == L, u_, bcs)

                                u_ex = interpolate(Expression('(exp(x[0]/mu) - \
                                        1)/(exp(1./mu) - 1)', \
                                        mu=mu),V_1)

                                errs[mucount][ncount] = (errornorm(u_ex,u_, err_type))

                                mucount += 1
                        ncount += 1

                for j in range(3):
                        print('mu = ' + '{:.1e}'.format(muvals[j]) + '   ' + \
                                '{:.3e}'.format(errs[j][0]) + \
                                '        ' + '{:.3e}'.format(errs[j][1]) + '       ' + \
                                '{:.3e}'.format(errs[j][2]) + \
                                '        ' + '{:.3e}'.format(errs[j][3]))
                print('\n')
```

Below is a printout of the computed errors for $\mu = 1$, 0.1 and 0.01. The errors for 0.001 and 0.0001 are computed as NaN, so these have not been included here.

```
                H1 error for P1 elements

                       N = 8          N = 16         N = 32         N = 64
mu = 1.0e+00        3.752e-02      1.877e-02      9.383e-03      4.692e-03
mu = 1.0e-01        7.671e-01      3.981e-01      2.010e-01      1.008e-01
mu = 1.0e-02        7.238e+00      6.684e+00      5.007e+00      2.969e+00


                L2 error for P1 elements

                       N = 8          N = 16         N = 32         N = 64
mu = 1.0e+00        1.402e-03      3.508e-04      8.770e-05      2.193e-05
mu = 1.0e-01        2.375e-02      6.177e-03      1.561e-03      3.915e-04
mu = 1.0e-02        2.379e-01      1.039e-01      3.819e-02      1.126e-02


                H1 error for P2 elements

                       N = 8          N = 16         N = 32         N = 64
mu = 1.0e+00        5.970e-04      1.504e-04      3.772e-05      9.448e-06
mu = 1.0e-01        1.183e-01      3.164e-02      8.066e-03      2.028e-03
mu = 1.0e-02        5.140e+00      3.604e+00      1.705e+00      5.661e-01


                L2 error for P2 elements

                       N = 8          N = 16         N = 32         N = 64
mu = 1.0e+00        1.151e-05      1.448e-06      1.817e-07      2.277e-08
mu = 1.0e-01        2.245e-03      3.038e-04      3.884e-05      4.888e-06
mu = 1.0e-02        8.513e-02      3.039e-02      7.598e-03      1.326e-03
```

c)

The following program checks the validity of the error estimates (note that only the three first values for μ are used as only these give results)

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

muvals = [1., 0.1, 0.01]
nvals = [8,16,32,64]

for mu in muvals:
        for err_type in ['H1', 'L2']:
                for i in [1,2]:

                        x = []
                        y = []
                        Alist = []
                        for N in nvals:

                                mesh = UnitSquareMesh(N,N)

                                h = 1.0/N

                                V = FunctionSpace(mesh, 'Lagrange', i)
                                V_1 = FunctionSpace(mesh, 'Lagrange', i+2)

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

                                bc0 = DirichletBC(V, Constant(0.0), 'x[0] < DOLFIN_EPS')
                                bc1 = DirichletBC(V, Constant(1.0), 'x[0] > 1-DOLFIN_EPS')
                                bcs = [bc0, bc1]


                                f = Constant(0.0)

                                a = mu*inner(grad(u),grad(v))*dx + u.dx(0)*v*dx
                                L = f*v*dx

                                u_ = Function(V)

                                solve(a == L, u_, bcs)


                                u_ex = interpolate(Expression('(exp(x[0]/mu) - \
                                        1)/(exp(1./mu) - 1)', \
                                        mu=mu),V_1)

                                A = errornorm(u_ex, u_, err_type, degree_rise=3)
```

```
                    Alist.append(A)
                    x.append(ln(h))
                    y.append(ln(A))


            Alist = np.array(Alist)
            x = np.array(x)
            y = np.array(y)

            Arr = np.vstack([x, np.ones(len(x))]).T
            alpha, lnC = np.linalg.lstsq(Arr, y)[0]
            print('%s, mu = %.3e, P%d elements:\n' % (err_type, mu, i))
            print('alpha/beta = %.3f, C = %9.3f' % (alpha, exp(lnC)))
            #print('alpha/beta = %.3f, C = %9.3f' % (alpha, lnC))

            counter = 0
            for N in nvals:
                    h = 1./N
                    RHS = exp(lnC)*h**alpha
                    print('N = %d: Errornorm = %.3e, Ch^alpha/beta = %.3e' \
                            % (N, Alist[counter], RHS))
                    if Alist[counter] < RHS:
                            print('Error estimate is valid for N = %d!\n' % N)
                    else:
                            print('Error estimate is NOT valid for N = %d!\n' % N)
                    counter += 1
            print('\n')
```

Below is a printout of the validity of the error estimates

```
H1, mu = 1.000e+00, P1 elements:

alpha/beta = 1.000, C =     0.300

N = 8: Errornorm = 3.752e-02, Ch^alpha/beta = 3.752e-02
Error estimate is valid for N = 8!

N = 16: Errornorm = 1.877e-02, Ch^alpha/beta = 1.876e-02
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 9.383e-03, Ch^alpha/beta = 9.383e-03
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 4.692e-03, Ch^alpha/beta = 4.692e-03
Error estimate is valid for N = 64!



H1, mu = 1.000e+00, P2 elements:

alpha/beta = 1.994, C =     0.038

N = 8: Errornorm = 5.970e-04, Ch^alpha/beta = 5.979e-04
Error estimate is valid for N = 8!

N = 16: Errornorm = 1.504e-04, Ch^alpha/beta = 1.501e-04
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 3.772e-05, Ch^alpha/beta = 3.768e-05
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 9.448e-06, Ch^alpha/beta = 9.460e-06
Error estimate is valid for N = 64!



L2, mu = 1.000e+00, P1 elements:

alpha/beta = 2.000, C =     0.090

N = 8: Errornorm = 1.402e-03, Ch^alpha/beta = 1.403e-03
Error estimate is valid for N = 8!

N = 16: Errornorm = 3.508e-04, Ch^alpha/beta = 3.507e-04
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 8.770e-05, Ch^alpha/beta = 8.769e-05
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 2.193e-05, Ch^alpha/beta = 2.193e-05
Error estimate is valid for N = 64!
```

```
L2, mu = 1.000e+00, P2 elements:

alpha/beta = 2.994, C =     0.006

N = 8: Errornorm = 1.151e-05, Ch^alpha/beta = 1.152e-05
Error estimate is valid for N = 8!

N = 16: Errornorm = 1.448e-06, Ch^alpha/beta = 1.447e-06
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 1.817e-07, Ch^alpha/beta = 1.816e-07
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 2.277e-08, Ch^alpha/beta = 2.279e-08
Error estimate is valid for N = 64!


H1, mu = 1.000e-01, P1 elements:

alpha/beta = 0.977, C =     5.907

N = 8: Errornorm = 7.671e-01, Ch^alpha/beta = 7.745e-01
Error estimate is valid for N = 8!

N = 16: Errornorm = 3.981e-01, Ch^alpha/beta = 3.935e-01
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 2.010e-01, Ch^alpha/beta = 1.999e-01
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 1.008e-01, Ch^alpha/beta = 1.016e-01
Error estimate is valid for N = 64!


H1, mu = 1.000e-01, P2 elements:

alpha/beta = 1.957, C =     7.045

N = 8: Errornorm = 1.183e-01, Ch^alpha/beta = 1.204e-01
Error estimate is valid for N = 8!

N = 16: Errornorm = 3.164e-02, Ch^alpha/beta = 3.100e-02
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 8.066e-03, Ch^alpha/beta = 7.984e-03
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 2.028e-03, Ch^alpha/beta = 2.056e-03
Error estimate is valid for N = 64!


L2, mu = 1.000e-01, P1 elements:

alpha/beta = 1.975, C =     1.459

N = 8: Errornorm = 2.375e-02, Ch^alpha/beta = 2.400e-02
Error estimate is valid for N = 8!

N = 16: Errornorm = 6.177e-03, Ch^alpha/beta = 6.102e-03
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 1.561e-03, Ch^alpha/beta = 1.552e-03
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 3.915e-04, Ch^alpha/beta = 3.947e-04
Error estimate is valid for N = 64!


L2, mu = 1.000e-01, P2 elements:

alpha/beta = 2.950, C =     1.056

N = 8: Errornorm = 2.245e-03, Ch^alpha/beta = 2.291e-03
Error estimate is valid for N = 8!

N = 16: Errornorm = 3.038e-04, Ch^alpha/beta = 2.965e-04
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 3.884e-05, Ch^alpha/beta = 3.838e-05
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 4.888e-06, Ch^alpha/beta = 4.967e-06
Error estimate is valid for N = 64!
```

```
H1, mu = 1.000e-02, P1 elements:

alpha/beta = 0.427, C =    19.638

N = 8: Errornorm = 7.238e+00, Ch^alpha/beta = 8.076e+00
Error estimate is valid for N = 8!

N = 16: Errornorm = 6.684e+00, Ch^alpha/beta = 6.006e+00
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 5.007e+00, Ch^alpha/beta = 4.466e+00
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 2.969e+00, Ch^alpha/beta = 3.321e+00
Error estimate is valid for N = 64!


H1, mu = 1.000e-02, P2 elements:

alpha/beta = 1.063, C =    56.604

N = 8: Errornorm = 5.140e+00, Ch^alpha/beta = 6.209e+00
Error estimate is valid for N = 8!

N = 16: Errornorm = 3.604e+00, Ch^alpha/beta = 2.972e+00
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 1.705e+00, Ch^alpha/beta = 1.423e+00
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 5.661e-01, Ch^alpha/beta = 6.811e-01
Error estimate is valid for N = 64!


L2, mu = 1.000e-02, P1 elements:

alpha/beta = 1.465, C =     5.508

N = 8: Errornorm = 2.379e-01, Ch^alpha/beta = 2.619e-01
Error estimate is valid for N = 8!

N = 16: Errornorm = 1.039e-01, Ch^alpha/beta = 9.487e-02
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 3.819e-02, Ch^alpha/beta = 3.437e-02
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 1.126e-02, Ch^alpha/beta = 1.245e-02
Error estimate is valid for N = 64!


L2, mu = 1.000e-02, P2 elements:

alpha/beta = 2.001, C =     6.533

N = 8: Errornorm = 8.513e-02, Ch^alpha/beta = 1.018e-01
Error estimate is valid for N = 8!

N = 16: Errornorm = 3.039e-02, Ch^alpha/beta = 2.542e-02
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 7.598e-03, Ch^alpha/beta = 6.350e-03
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 1.326e-03, Ch^alpha/beta = 1.586e-03
Error estimate is valid for N = 64!
```

d)

To implement the SUPG method we replace the test function v with a new test function $w = v + \beta u_x$. Inserted into the weak form for our problem, we get

$$\mu \int_\Omega \nabla u \nabla w \,\mathrm{d}x + \int_\Omega u_x w \,\mathrm{d}x = \mu \int_\Omega \nabla u \nabla (v + \beta u_x) \,\mathrm{d}x + \int_\Omega u_x (v + \beta u_x) \,\mathrm{d}x =$$

$$\mu \int_\Omega \nabla u \nabla v \,\mathrm{d}x + \beta\mu \int_\Omega \nabla u \nabla u_x \,\mathrm{d}x + \int_\Omega u_x v \,\mathrm{d}x + \int_\Omega u_x^2 \,\mathrm{d}x = 0$$

Below is a printout of the computed values for α and C, as well as a error estimate validity check for each case

```
mu = 1.000e+00, P1 elements:

alpha/beta = 1.131, C =      0.254

N = 8: Errornorm = 2.472e-02, Ch^alpha/beta = 2.419e-02
Error estimate is NOT valid for N = 8!

N = 16: Errornorm = 1.080e-02, Ch^alpha/beta = 1.105e-02
Error estimate is valid for N = 16!

N = 32: Errornorm = 4.948e-03, Ch^alpha/beta = 5.044e-03
Error estimate is valid for N = 32!

N = 64: Errornorm = 2.351e-03, Ch^alpha/beta = 2.303e-03
Error estimate is NOT valid for N = 64!


mu = 1.000e+00, P2 elements:

alpha/beta = 0.475, C =      2.378

N = 8: Errornorm = 8.799e-01, Ch^alpha/beta = 8.849e-01
Error estimate is valid for N = 8!

N = 16: Errornorm = 6.405e-01, Ch^alpha/beta = 6.365e-01
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 4.598e-01, Ch^alpha/beta = 4.578e-01
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 3.276e-01, Ch^alpha/beta = 3.293e-01
Error estimate is valid for N = 64!


mu = 1.000e-01, P1 elements:

alpha/beta = 1.062, C =      3.528

N = 8: Errornorm = 3.836e-01, Ch^alpha/beta = 3.878e-01
Error estimate is valid for N = 8!

N = 16: Errornorm = 1.886e-01, Ch^alpha/beta = 1.858e-01
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 8.921e-02, Ch^alpha/beta = 8.899e-02
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 4.234e-02, Ch^alpha/beta = 4.263e-02
Error estimate is valid for N = 64!
```

```
mu = 1.000e-01, P2 elements:

alpha/beta = 0.276, C =      1.806

N = 8: Errornorm = 9.853e-01, Ch^alpha/beta = 1.018e+00
Error estimate is valid for N = 8!

N = 16: Errornorm = 8.683e-01, Ch^alpha/beta = 8.410e-01
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 7.187e-01, Ch^alpha/beta = 6.947e-01
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 5.552e-01, Ch^alpha/beta = 5.739e-01
Error estimate is valid for N = 64!


mu = 1.000e-02, P1 elements:

alpha/beta = 0.395, C =      2.764

N = 8: Errornorm = 1.083e+00, Ch^alpha/beta = 1.215e+00
Error estimate is valid for N = 8!

N = 16: Errornorm = 1.031e+00, Ch^alpha/beta = 9.235e-01
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 7.957e-01, Ch^alpha/beta = 7.022e-01
Error estimate is NOT valid for N = 32!

N = 64: Errornorm = 4.735e-01, Ch^alpha/beta = 5.339e-01
Error estimate is valid for N = 64!


mu = 1.000e-02, P2 elements:

alpha/beta = 0.247, C =      2.859

N = 8: Errornorm = 1.681e+00, Ch^alpha/beta = 1.710e+00
Error estimate is valid for N = 8!

N = 16: Errornorm = 1.487e+00, Ch^alpha/beta = 1.441e+00
Error estimate is NOT valid for N = 16!

N = 32: Errornorm = 1.202e+00, Ch^alpha/beta = 1.214e+00
Error estimate is valid for N = 32!

N = 64: Errornorm = 1.019e+00, Ch^alpha/beta = 1.023e+00
Error estimate is valid for N = 64!
```