# MEK 4250 Elementmethod
# Mandatory Assignment

Andreas Slyngstad

17. mars 2016

## Problem 1

In this exercise we are faced with a problem on the domain $\Omega = (0,1)^2$

$$-\nabla u = f \text{ in } \Omega \tag{1}$$
$$\text{u} = 0 \text{ for x} = 0 \text{ and x} = 1 \tag{2}$$
$$\frac{\partial u}{\partial n} = 0 \text{ for } y = 0 \text{ and } y = 1 \tag{3}$$

We know that the analytical solution is on the form

$$u(x,y) = sin(k\pi x)cos(l\pi y)$$

### Exercise A

Given

$$u(x,y) = sin(k\pi x)cos(l\pi y)$$

We can calculate the $H^p$ norm as defined in the lecture notes *Definition 2.13* as follows

$$||u||_{H^p} = \sqrt{\sum_{|\alpha| \leq p} \int_\Omega |\partial^\alpha v|^2 dx}$$

We restrict $k, l$ to be whole numbers $k, l \in \mathbb{Z}$
To find some sort of relation of this sum, we first look at the case $\alpha = 0$

$$\int_0^1 \int_0^1 sin^2(k\pi x)cos^2(l\pi y)dxdy$$

$$-\frac{(-2k\pi + sin(2k\pi))(2l\pi + sin(2l\pi))}{16kl\pi^2} = \frac{1}{4}$$

Exploiting the relations which will appear in the integration of the derivatives

$$\int_0^1 sin^2(k\pi x)dx = \frac{1}{2} \qquad \int_0^1 cos^2(k\pi x)dx = \frac{1}{2}$$

We can express the $H^p$ norm as the sum

$$H^p = \sqrt{\frac{1}{4}\sum_0^p ((k\pi)^2 + (l\pi)^2)^p}$$

## Exericse B

In this exercise we were to calculate the $L_2$ and $H^1$ errors form our numerical experiments. The experiments were calculated on a unit squaremesh for $\frac{1}{h} = [8, 16, 32, 64]$. I chose to limit my exploration of errors to the case where $k = l = [1, 10, 100]$. I still think this limitation shows the significant trends we are supposed to look at. My program yields the following output.

```
###########################################################

#-------------------- 1 degree elements -----------------#

###########################################################


###########################################################

#----------------------- L2 Norm ---------------------#

=============  ========  ========  ======  ======
Values of N           8        16      32      64
=============  ========  ========  ======  ======
k_l = 1          0.0328    0.0085  0.0021  0.0005
k_1 = 10         0.6671    0.3655  0.1782  0.0549
k_l = 100      159.356   246.862   2.6969  3.5888
=============  ========  ========  ======  ======

#----------------------- H1 Norm ---------------------#

=============  =========  =========  ========  ========
Values of N            8         16        32        64
=============  =========  =========  ========  ========
k_l = 1           0.4366     0.2182    0.1091    0.0545
k_1 = 10         26.4815    17.5464   10.6024    5.4399
k_l = 100      3226.29    4686.2     376.364   540.467
=============  =========  =========  ========  ========

###########################################################

#-------------------- Linear Approximation ---------------#


                     Norm = L2     k_l = 1
                   alpha = 1.9804, Constant = 2.0308

Errornorm (u-u_h) < C*h^(alpha) is True for N = 8
Errornorm (u-u_h) < C*h^(alpha) is True for N = 16
Errornorm (u-u_h) < C*h^(alpha) is True for N = 32
Errornorm (u-u_h) < C*h^(alpha) is True for N = 64

                     Norm = H1     k_l = 1
                   alpha = 1.0004, Constant = 3.4954

Errornorm (u-u_h) < C*h^(alpha) is True for N = 8
Errornorm (u-u_h) < C*h^(alpha) is True for N = 16
Errornorm (u-u_h) < C*h^(alpha) is True for N = 32
Errornorm (u-u_h) < C*h^(alpha) is True for N = 64
```

```
###########################################################

#-------------------- 2 degree elements -----------------#

###########################################################


###########################################################

#----------------------- L2 Norm ---------------------#

=============  ========  =======  ======  ======
Values of N           8       16      32      64
=============  ========  =======  ======  ======
k_l = 1          0.0006   0.0001  0       0
k_1 = 10         0.4356   0.0896  0.0102  0.0011
k_l = 100      293.246   90.4749  4.7223  1.471
=============  ========  =======  ======  ======

#----------------------- H1 Norm ---------------------#

=============  =========  =========  ========  ========
Values of N            8         16        32        64
=============  =========  =========  ========  ========
k_l = 1           0.0332     0.0084    0.0021    0.0005
k_1 = 10         19.1245     6.9203    1.978     0.5184
k_l = 100      5321.28    1648.5     689.092   288.597
=============  =========  =========  ========  ========

###########################################################

#-------------------- Linear Approximation ---------------#


                       Norm = L2     k_l = 1
                     alpha = 3.0154, Constant = 0.2989

Errornorm (u-u_h) < C*h^(beta) is True for N = 8
Errornorm (u-u_h) < C*h^(beta) is True for N = 16
Errornorm (u-u_h) < C*h^(beta) is True for N = 32
Errornorm (u-u_h) < C*h^(beta) is True for N = 64

                       Norm = H1     k_l = 1
                     alpha = 1.9923, Constant = 2.0955

Errornorm (u-u_h) < C*h^(alpha) is True for N = 8
Errornorm (u-u_h) < C*h^(alpha) is True for N = 16
Errornorm (u-u_h) < C*h^(alpha) is True for N = 32
Errornorm (u-u_h) < C*h^(alpha) is True for N = 64
```

From the output we observe that both the $L_2$ and $H^1$ norms are increasing for some chosen point N.
For the $L_2$ case the reason for the increasing values is because of the increasing wavenumber in the analytical solution. Since the solution has a period of $\frac{-2\pi}{k}$ in x and $\frac{-2\pi}{l}$ in y, we aren't able to represent the solution correctly due to lack of number of elements for increasing k and l.
For the $H_1$ case we would expect increasing $H_1$ values because the oscillating solution, will result in higher values of the derivative. Hence we would expect higher values for the $H_1$ norm as k and l increase.

### Exericse C

In this exercise we were to evaluate the following error estimates

$$||u - u_h||_1 <= C_\alpha h^\alpha$$
$$||u - u_h||_0 <= C_\beta h^\beta$$

by employing the least square method to estimate $\alpha$, $\beta$ and C. Here I have limited the experiments for $k = l = 1$ because this gives the most reasonable numerical results.

From our lecture notes we expect the $L_2$ estimate of the error to yield an $\alpha$ value one value higher than the order of elements. While the $H_1$ estimate of the error would give $\beta$ same as the order of elements.
From the numerical calculations we get

| | $\alpha$ | $\beta$ | $C_\alpha$ | $C_\beta$ |
|---|---|---|---|---|
| $P1$ | 1.9804 | 1.0004 | 2.0308 | 3.4954 |
| $P2$ | 3.0154 | 1.9923 | 0.2989 | 2.0955 |

The *a priori* estimation of convergence rate seems valid according to my calculations. From my output I also conclude that the error estimates are valid for the case $k = l = 1$ for all number of elements.

## Exercise 2

We are presented with the following system

$$-\mu\Delta u + u_x = 0 \text{ in } \Omega \quad (4)$$
$$u = 0 \text{ for } x = 0 \quad (5)$$
$$u = 1 \text{ for } x = 1 \quad (6)$$
$$\frac{\partial u}{\partial n} = 0 \text{ for } y = 0 \text{ and } y = 1 \quad (7)$$

### Exercise A

By assuming a solution on the form $u(x, y) = X(x)Y(y)$, we get by insertion

$$-\mu(Y(y)X(x)'' + X(x)Y(y)'') + Y(y)X(x)' = 0$$
$$\frac{Y''}{Y} = \frac{X' - \mu X''}{\mu X} = -\lambda^2$$

Where $\lambda$ is some arbitrary constant. Solving for Y we get

$$\lambda Y'' + Y = 0$$
$$Y(y) = Acos(\sqrt{\lambda}y) + Bsin(\sqrt{\lambda}y)$$
$$Y'(y) = -A\sqrt{\lambda}sin(\sqrt{\lambda}y) + B\sqrt{\lambda}cos(\sqrt{\lambda}y)$$

From the boundary conditions, and by assuming $\lambda \neq 0$ we get

$$Y'(0) = 0 + B\sqrt{\lambda} = 0 \;\; B = 0$$
$$Y'(1) = -A\sqrt{\lambda}sin(\sqrt{\lambda}) = 0$$
$$\lambda = n\pi \;\; A = 0$$

Assuming $\lambda = 0$ we get a linear solution

$$\frac{Y''}{Y} = 0$$
$$Y(y) = Ay + B \;\; Y'(0) = A = 0$$
$$Y(y) = B$$

As we can see, the function of Y is just a consant, which is convenient to set as $B = 1$

Now, focusing on the other function of X for $\lambda = 0$ we get

$$X' - \mu X'' = 0$$
$$X(x) = \frac{C}{\mu}e^{\frac{x}{\mu}} + D$$
$$X(0) = \frac{C}{\mu} + D = 0 \quad X(1) = \frac{C}{\mu}e^{\frac{1}{\mu}} + D = 1$$

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

Hence the analytical solution can be expressed as

$$u(x) = \frac{e^{\frac{x}{\mu}} - 1}{e^{\frac{1}{\mu}} - 1} \tag{8}$$

## Exercise B

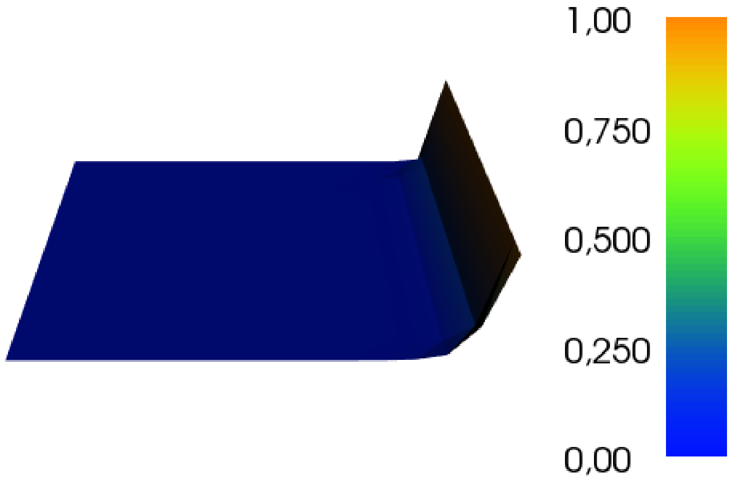
Running the numerical experiments for values
$\mu = [1, 0.1, 0.01, 0.001, 0.0001]$
$h = [8, 16, 32, 64]$

I get the following output

```
###########################################################
#-------------------- 1 degree elements -----------------#
###########################################################


###########################################################
#-------------------- Linear Approximation ---------------#

                Norm = L2     my = 1
                alpha = 1.9998, Constant = 0.0897

Errornorm (u-u_h) < C*h^(alpha) is True for N = 8
Errornorm (u-u_h) < C*h^(alpha) is True for N = 16
Errornorm (u-u_h) < C*h^(alpha) is True for N = 32
Errornorm (u-u_h) < C*h^(alpha) is True for N = 64
                Norm = H1     my = 1
                alpha = 0.9998, Constant = 0.3001

Errornorm (u-u_h) < C*h^(alpha) is True for N = 8
Errornorm (u-u_h) < C*h^(alpha) is True for N = 16
Errornorm (u-u_h) < C*h^(alpha) is True for N = 32
Errornorm (u-u_h) < C*h^(alpha) is True for N = 64
###########################################################
#----------------------- L2 Norm ---------------------#

=============  ==========  ==========  ==========  ==========
Values of N             8          16          32          64
=============  ==========  ==========  ==========  ==========
my = 1           0.001402    0.000351     8.8e-05     2.2e-05
my = 0.1         0.023754    0.006177    0.001561    0.000391
my = 0.01        0.237934    0.103936    0.038186    0.011259
my = 0.001     nan         nan         nan         nan
my = 0.0001    nan         nan         nan         nan
=============  ==========  ==========  ==========  ==========

#----------------------- H1 Norm ---------------------#

=============  ==========  ==========  ==========  ==========
Values of N             8          16          32          64
=============  ==========  ==========  ==========  ==========
my = 1           0.037521    0.018765    0.009383    0.004692
my = 0.1         0.767086    0.398104    0.201041    0.100777
my = 0.01        7.23835     6.68438     5.00716     2.96949
my = 0.001     nan         nan         nan         nan
my = 0.0001    nan         nan         nan         nan
=============  ==========  ==========  ==========  ==========
```

```
##########################################################

#-------------------- 2 degree elements -----------------#

##########################################################


##########################################################

#-------------------- Linear Approximation ---------------#

                 Norm = L2     my = 1
                 alpha = 2.9940, Constant = 0.0058

Errornorm (u-u_h) < C*h^(alpha) is True for N = 8
Errornorm (u-u_h) < C*h^(alpha) is True for N = 16
Errornorm (u-u_h) < C*h^(alpha) is True for N = 32
Errornorm (u-u_h) < C*h^(alpha) is True for N = 64
                 Norm = H1     my = 1
                 alpha = 1.9940, Constant = 0.0378

Errornorm (u-u_h) < C*h^(alpha) is True for N = 8
Errornorm (u-u_h) < C*h^(alpha) is True for N = 16
Errornorm (u-u_h) < C*h^(alpha) is True for N = 32
Errornorm (u-u_h) < C*h^(alpha) is True for N = 64
##########################################################

#----------------------- L2 Norm ---------------------#

=============  ==========  ==========  ==========  ==========
Values of N             8          16          32          64
=============  ==========  ==========  ==========  ==========
my = 1           1.2e-05     1e-06       0           0
my = 0.1         0.002245    0.000304    3.9e-05     5e-06
my = 0.01        0.085126    0.030391    0.007598    0.001326
my = 0.001     nan         nan         nan         nan
my = 0.0001    nan         nan         nan         nan
=============  ==========  ==========  ==========  ==========

#----------------------- H1 Norm ---------------------#

=============  ==========  =========  ==========  ==========
Values of N             8         16          32          64
=============  ==========  =========  ==========  ==========
my = 1           0.000597    0.00015    3.8e-05     9e-06
my = 0.1         0.118321    0.03164    0.008066    0.002028
my = 0.01        5.14048     3.60445    1.70493     0.566126
my = 0.001     nan         nan        nan         nan
my = 0.0001    nan         nan        nan         nan
=============  ==========  =========  ==========  ==========
```

Representation of the calculated solution for $\mu = 0.001$




We observe from the analytical solution (8) that for lower values of $\mu$, python isn't able to represent the exponential $exp(\frac{1}{\mu})$. Hence we get values in the output that python can't produce. Another consequence is that the diffusion term contributes less to the solution . The solution change from one of exponential growth to a sudden steep gradient at the end of the domain. This gives certain effects in the calculated norms as we decrease the value of $\mu$. This sudden gradient will result some larger errors which will effect the L2 norm, and ecspecially the H1 norm as we can see from the output.

## Exericse C

In this exercise we where to evaluate the following error estimates again

$$||u - u_h||_1 <= C_\alpha h^\alpha$$
$$||u - u_h||_0 <= C_\beta h^\beta$$

by employing least square method to estimate $\alpha$, $\beta$ and C. Here I have limited the experiments for $\mu = 1$ because this gives the most reasonable numerical results.

| | $\alpha$ | $\beta$ | $C_\alpha$ | $C_\beta$ |
|---|---|---|---|---|
| $P1$ | 1.9998 | 0.9998 | 0.0897 | 0.3001 |
| $P2$ | 2.9940 | 1.9940 | 0.0058 | 0.0378 |

Using the same arguments as in exercise 1c, we see that the presented results as for convergence rates are satisfying.

## Exericse D

In this exercise we were to implement the Streamwise Upwinding Petrov-Galerkin (SUPG) method. From our lecture notes we know that an alternative errornorm is presented to obtain better error estimates.

$$||u||_{sd} = \left(h||v \cdot \nabla u||^2 + \mu||\nabla u||^2\right)^{\frac{1}{2}}$$
$$||u - u_h|| \leq Ch^{\frac{3}{2}}||u||_2$$

Implementing the SUPG method we exchange the ordinary testfunction $V$ for $L = V + \beta v \nabla V$. This will induce an artificial diffusion term to the system, which will in fact transform the system to a upwind system from from a finite difference point of view. My experiments yields.

```
##########################################################

#-------------------- 1 degree elements -----------------#

##########################################################


##########################################################

#-------------------- Linear Approximation ---------------#

                 Norm      my = 1
                 alpha = 0.5159, Constant = 0.1202

##########################################################

#----------------------- L2 Norm ---------------------#

=============  ==========  ==========  ==========  ==========
Values of N             8          16          32          64
=============  ==========  ==========  ==========  ==========
my = 1           0.030251    0.029669    0.029525    0.029489
my = 0.1         0.317615    0.316281    0.315949    0.315866
my = 0.01        0.422485    0.427869    0.428659    0.428674
my = 0.001     nan         nan         nan         nan
my = 0.0001    nan         nan         nan         nan
=============  ==========  ==========  ==========  ==========

#----------------------- H1 Norm ---------------------#

=============  ==========  ==========  ==========  ==========
Values of N             8          16          32          64
=============  ==========  ==========  ==========  ==========
my = 1           0.105669    0.101119    0.099948    0.099653
my = 0.1         1.69142     1.67803     1.67431     1.67336
my = 0.01        5.08569     6.36207     6.79165     6.85168
my = 0.001     nan         nan         nan         nan
my = 0.0001    nan         nan         nan         nan
=============  ==========  ==========  ==========  ==========
```

```
#-------------------- 2 degree elements -----------------#

##########################################################


##########################################################

#-------------------- Linear Approximation ---------------#

                 Norm = L2     k_l = 1
                 alpha = -3.1504, Constant = 0.0104

                 Norm = H1     k_l = 1
                 alpha = -1.3418, Constant = 0.7286

##########################################################

#----------------------- L2 Norm ---------------------#

=============  ==========  ==========  ==========  ==========
Values of N             8          16          32          64
=============  ==========  ==========  ==========  ==========
my = 1           0.410669    0.411099    0.425731    0.5872
my = 0.1         0.394629    0.392419    0.391779    0.391889
my = 0.01        0.435901    0.437556    0.437833    0.43755
my = 0.001     nan         nan         nan         nan
my = 0.0001    nan         nan         nan         nan
=============  ==========  ==========  ==========  ==========

#----------------------- H1 Norm ---------------------#

=============  =========  =========  =========  =========
Values of N            8         16         32         64
=============  =========  =========  =========  =========
my = 1          13.5928    27.3796    61.1822   230.86
my = 0.1         5.14402    9.25837   17.9106    36.2078
my = 0.01        5.98764    7.00016    8.17057    9.61022
my = 0.001     nan        nan        nan        nan
my = 0.0001    nan        nan        nan        nan
=============  =========  =========  =========  =========
```

From our norms, it seems that the SUPG method is not as accurate as the first implementation. From our print we also observe that the $\alpha$ value for P1 elements is 0.51, which is totally wrong from the estimated value of $\frac{3}{2}$ from our lecture notes. I have tried several approaches to fix this without luck...

```python
############################################
#Author: Andreas Slyngstad
#MEK 4250
#Solving Poission Equation with both Dirichlet
#and Neumann conditions
#############################################

from dolfin import *
import numpy as np
from tabulate import tabulate
#http://www.math.rutgers.edu/~falk/math575/Boundary-conditions.html


class Poission():
    def __init__(self, h):
        self.y = np.zeros(len(h)); self.y1 = np.zeros(len(h))
        self.x = np.zeros(len(h)); self.h_list = h
        self.L2list = []; self.H1list = []
        self.alpha = 0; self.beta = 0
        self.count = 0

    def set_mesh(self,i):
        self.h = i
        self.mesh = UnitSquareMesh(i, i)

    def calc(self, i, k, l, output=True):
        mesh = self.mesh

        #Defining spaces and functions
        V = FunctionSpace(mesh, 'CG', i)
        u = TrialFunction(V)
        v = TestFunction(V)

        class Dirichlet(SubDomain):
            def inside(self, x, on_boundary):
                return on_boundary and( near(x[0], 0) or near(x[0], 1) )

        diri = Dirichlet()
        #Setting boundary values
        boundaries = FacetFunction("size_t", mesh)
        boundaries.set_all(0)
        diri.mark(boundaries,1)
        bc0 = DirichletBC(V, 0, diri)

        #Defining and solving variational problem
        V_1 = FunctionSpace(mesh, 'CG', i+2)
        u_e = interpolate(Expression('sin(k*pi*x[0])*cos(l*pi*x[1])', k=k, l=l),
        f = Expression("((pi*pi*k*k)+(pi*pi*l*l))*sin(pi*k*x[0])*cos(pi*l*x[1])",
        a = inner(grad(u), grad(v))*dx
        L = f*v*dx

        u_ = Function(V)
        solve(a == L, u_, bc0)

        #Norms of the error
        L2 = errornorm(u_e, u_, norm_type='L2', degree_rise = 3)
```

```
        H1 = errornorm(u_e, u_, norm_type='H1', degree_rise = 3)
        self.L2list.append(str(round(L2,4) ))
        self.H1list.append(str(round(H1,4) ))

        if output == True:
            print "----------------------------------"
            print "For %d points and k, l = %d" % (self.h, k)
            print "L2 Norm = %.5f -----  H1 Norm = %.5f" % (L2, H1)
            print
        if k == 1:
            d = mesh.coordinates()
            self.x[self.count] = np.log(1./self.h)
            self.y[self.count] = np.log( L2 )
            self.y1[self.count] = np.log( H1 )
            self.count += 1

    def l_square(self, norm ,fig):
        A = np.zeros((2, 2))
        b = np.zeros(2)

        mid = self.y #hold y values if norm = H1
        test = self.y #Holds L2 errornorms
        if norm == 'H1':
            self.y = self.y1
            test = self.y1 #Holds H! errornorms

        A[0][0] = len(self.h_list)
        A[0][1] = np.sum(self.x); A[1][0] = A[0][1]
        A[1][1] = np.sum(self.x*self.x)
        b[0] = np.sum(self.y)
        b[1] = np.sum(self.y*self.x)

        a, b = np.linalg.solve(A, b)
        self.beta = a ; self.alpha = b

        print
        print '                              Norm = %s      k_l = %d' % (norm ,1)
        print '                            alpha = %.4f, Constant = %.4f \n' % (prob.alp
        for i in range(len(test)):
            print 'Errornorm (u-u_h) < C*h^(alpha) is %s for N = %d' %(test[i]<b*

        if fig == True:
            import matplotlib.pyplot as plt
            plt.figure(1)
            plt.plot(self.x, b*self.x + a, label='Linear approximation')
            plt.plot(self.x, self.y, 'o', label='Points to be approximated')
            plt.legend(loc = 'upper left')
            plt.show()
        self.y = mid

    def make_list(self, h):

        k_1 = ['k_l = 1']; k_10 = ['k_1 = 10']; k_100 = ['k_l = 100']
        for i in range(0, len(self.L2list)-2, 3 ):
            k_1.append(str(self.L2list[i]) )
            k_10.append( str(self.L2list[i+1]) )
```

```
            k_100.append( str(self.L2list[i+2]) )

        table = [k_1, k_10, k_100]
        headers = ['Values of N']
        for i in h:
            headers.append(str(i))

        print '#----------------------- L2 Norm ---------------------#\n'
        print tabulate(table, headers, tablefmt='rst')

        l_1 = ['k_l = 1']; l_10 = ['k_1 = 10']; l_100 = ['k_l = 100']

        for i in range(0, len(self.H1list)-2, 3 ):
            l_1.append(str(self.H1list[i]) )
            l_10.append( str(self.H1list[i+1]) )
            l_100.append( str(self.H1list[i+2]) )
        table = [l_1, l_10, l_100]
        print
        print '#----------------------- H1 Norm ---------------------#\n'
        print tabulate(table, headers, tablefmt='rst')
        print

        self.L2list = []
        self.H1list = []


set_log_active(False) #Removing all logging
kl = [1, 10, 100]
h = [2**(i+3) for i in range(4)]

prob = Poission(h)
for j in [1, 2]:
    print '###########################################################\n'
    print '#-------------------- %d degree elements ----------------#\n' % j
    print '###########################################################\n'
    print
    for i in h:
        for k in kl:
            prob.set_mesh(i)
            prob.calc(j, k, k, output = False)

    print '###########################################################\n'
    print '#-------------------- Linear Approximation ---------------#\n'
    for l in ['L2', 'H1']:
        prob.l_square(l, fig = False)
    print
    print '###########################################################\n'
    prob.make_list(h)
    prob.count = 0
```

```python
###########################################
#Author: Andreas Slyngstad
#MEK 4250
#EXERCISE 2
#Solving Poission Equation with both Dirichlet
#and Neumann conditions
############################################

from dolfin import *
import numpy as np
from tabulate import tabulate

class Poission():
    def __init__(self, h):
        self.y = np.zeros(len(h)); self.y1 = np.zeros(len(h))
        self.x = np.zeros(len(h)); self.h_list = h
        self.L2list = []; self.H1list = []
        self.alpha = 0; self.beta = 0
        self.count = 0

    def set_mesh(self,i):
        self.h = i
        self.mesh = UnitSquareMesh(i, i)

    def calc(self, i, my, output, upwind, imp_norm):
        mesh = self.mesh

        #Defining spaces and functions
        V = FunctionSpace(mesh, 'CG', i)
        u = TrialFunction(V)
        v = TestFunction(V)

        class Left(SubDomain):
            def inside(self, x, on_boundary):
                return on_boundary and near(x[0], 0)

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

        left = Left(); right = Right()
        #Setting boundary values
        boundaries = FacetFunction("size_t", mesh)
        boundaries.set_all(0)
        left.mark(boundaries,1)
        right.mark(boundaries, 2)
        bc0 = DirichletBC(V, 0, left)
        bc1 = DirichletBC(V, 1, right)
        bcs = [bc0, bc1]

        #Defining and solving variational problem
        V_1 = FunctionSpace(mesh, 'CG', i+2)
        u_e = interpolate(Expression('1./(exp(1./my)- 1 ) * (exp(x[0]/my) - 1)',
        f = Constant(0)
        if upwind == True:
            beta_val = 0.5
```

```
            beta = Constant(beta_val)
            v = v + beta*v.dx(0)
            a = my * inner(grad(u), grad(v))*dx + u.dx(0)*v*dx #Standard Galerkin
            L = f*v*dx
        else:
            a = my * inner(grad(u), grad(v))*dx + u.dx(0)*v*dx
            L = f*v*dx

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

        #Norms of the error
        L2 = errornorm(u_e, u_, norm_type='L2', degree_rise = 3)
        H1 = errornorm(u_e, u_, norm_type='H1', degree_rise = 3)

        self.L2list.append(str(round(L2, 6) ))
        self.H1list.append(str(round(H1, 6) ))

        #plot(u_); interactive()

        if output == True:
            print "----------------------------------"
            print "For %d points and my = %d" % (self.h, my)
            print "L2 Norm = %.5f -----  H1 Norm = %.5f" % (L2, H1)
            print

        if my == 1:

            d = mesh.coordinates()
            self.x[self.count] = np.log(1./self.h)

            if imp_norm == True:
                e_x = u_e.dx(0)-u_.dx(0)
                e_y = u_e.dx(1)-u_.dx(1)
                e_x = project(e_x, V); e_y = project(e_y, V)
                i_norm = np.sqrt(mesh.hmin()*norm(e_x, 'l2')**2 + my*(norm(e_x, '
                self.y[self.count] = np.log(i_norm)
            else:
                self.y[self.count] = np.log(L2)
            self.y1[self.count] = np.log( H1 )
            self.count += 1

    def l_square(self, norm, fig):
        A = np.zeros((2, 2))
        b = np.zeros(2)

        mid = self.y
        test = self.y
        if norm == 'H1':
            self.y = self.y1
            test = self.y1

        A[0][0] = len(self.h_list)
        A[0][1] = np.sum(self.x); A[1][0] = A[0][1]
        A[1][1] = np.sum(self.x*self.x)
        b[0] = np.sum(self.y)
```

```
        b[1] = np.sum(self.y*self.x)

        a, b = np.linalg.solve(A, b)
        self.beta = a ; self.alpha = b

        print '                        Norm = %s      k_1 = %d' % (norm ,1)
        print '                        alpha = %.4f, Constant = %.4f \n' % (prob.alpha,
        for i in range(len(test)):
            print 'Errornorm (u-u_h) < C*h^(alpha) is %s for N = %d' %(test[i]<b*

        self.y = mid
        if fig == True:
            import matplotlib.pyplot as plt
            plt.figure(1)
            plt.plot(self.x, b*self.x + a, label='Linear approximation')
            plt.plot(self.x, self.y, 'o', label='Points to be approximated')
            plt.legend(loc = 'upper left')
            plt.show()

    def make_list(self, h):

        k_1 = ['my = 1']; k_10 = ['my = 0.1']; k_100 = ['my = 0.01']
        k_1000 = ['my = 0.001']; k_10000 = ['my = 0.0001']

        for i in range(0, len(self.L2list)-4, 5 ):
            k_1.append(str(self.L2list[i]) )
            k_10.append( str(self.L2list[i+1]) )
            k_100.append( str(self.L2list[i+2]) )
            k_1000.append( str(self.L2list[i+3]) )
            k_10000.append( str(self.L2list[i+4]) )

        table = [k_1, k_10, k_100, k_1000, k_10000]
        headers = ['Values of N']
        for i in h:
            headers.append(str(i))

        print '#----------------------- L2 Norm ---------------------#\n'
        print tabulate(table, headers, tablefmt="rst")

        l_1 = ['my = 1']; l_10 = ['my = 0.1']; l_100 = ['my = 0.01']
        l_1000 = ['my = 0.001']; l_10000 = ['my = 0.0001']

        for i in range(0, len(self.H1list)-4, 5 ):
            l_1.append(str(self.H1list[i]) )
            l_10.append( str(self.H1list[i+1]) )
            l_100.append( str(self.H1list[i+2]) )
            l_1000.append( str(self.H1list[i+3]) )
            l_10000.append( str(self.H1list[i+4]) )
        table = [l_1, l_10, l_100, l_1000, l_10000]
        print
        print '#----------------------- H1 Norm ---------------------#\n'
        print tabulate(table, headers, tablefmt="rst") #fancy_grid
        print

        self.L2list = []
        self.H1list = []
```

```
set_log_active(False) #Removing all logging
my = [1*10**-i for i in range(5)]
h = [2**(i+3) for i in range(4)] #5

prob = Poission(h)
for j in [1, 2]:
    print '############################################################\n'
    print '#-------------------- %d degree elements ------------------#\n' % j
    print '############################################################\n'
    print
    for i in h:
        for m in my:
            prob.set_mesh(i)
            prob.calc(j, m, output = False, upwind = True, imp_norm = True)
    print '############################################################\n'
    print '#-------------------- Linear Approximation ----------------#\n'
    for k in ['L2', 'H1']:
        prob.l_square(k, fig = False)
    print '############################################################\n'
    prob.make_list(h)
    prob.count = 0
```