Mek4250 - Mandatory assignment 1

Andreas Thune

16.03.2016

Exercise 1

a) The H^p norm of a function $u(x, y)$ of two variables on $\Omega = (0, 1)^2$ is defined as follows:

$$||u||^2_{H^p(\Omega)} = \sum_{i=0}^{p} \sum_{j=0}^{i} \binom{i}{j} ||\frac{\partial^i u}{\partial^j x \partial^{i-j} y}||^2_{L^2(\Omega)} \tag{1}$$

In our case $u(x, y) = sin(k\pi x)cos(l\pi y)$. We easily see that the derivative of this function can be expressed with the following formula:

$$\frac{\partial^i u(x, y)}{\partial^j x \partial^i y} = (k\pi)^j (l\pi)^i f_j(k\pi x) g_j(l\pi y) \tag{2}$$

were f_j is the j-th derivative of $sin(x)$ and g_i is the i-th derivative of $cos(y)$. Now lets look at the L^2 norm of (2).

$$\begin{aligned} ||\frac{\partial^i u(x, y)}{\partial^j x \partial^i y}||^2_{L^2(\Omega)} &= (k\pi)^{2j} (l\pi)^{2i} \int \int_\Omega f_j(k\pi x)^2 g_i(l\pi y)^2 dxdy \\ &= (k\pi)^{2j} (l\pi)^{2i} \int_0^1 f_j(k\pi x)^2 dx \int_0^1 g_i(l\pi y)^2 dy \end{aligned}$$

Since f_j^2 and g_i^2 are:

$$f_j(x)^2 = \begin{cases} sin^2(x) & \text{j even} \\ cos^2(x) & \text{j odd} \end{cases}$$

and

$$g_i(y)^2 = \begin{cases} cos^2(y) & \text{i even} \\ sin^2(y) & \text{i odd} \end{cases}$$

and since

$$\int_0^1 sin^2(l\pi y) dy = \int_0^1 cos^2(l\pi y) dy = \frac{1}{2}$$

we get the following expression for the square of the L^2 norm of a general derivative of u:

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

If we plug this into (1), we get

$$||u||^2_{H^p(\Omega)} = \frac{1}{4} \sum_{i=0}^{p} \sum_{j=0}^{i} \binom{i}{j} (k\pi)^{2j} (l\pi)^{2(i-j)}$$

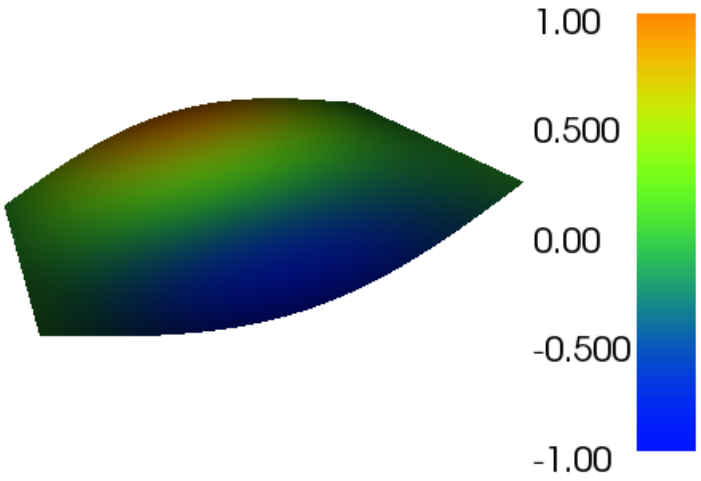

Figure 1: Plot of numerical solution u for $k = l = 1$ and $h = 64$. See that everything lies between -1 and 1 as one would expect from a product between sine and cosine functions. Notice also that the solution is smooth

b) To solve the equation numerically I need to derive the variational form of the equation. Since our boundary conditions are homogeneous Neumann and Dirichlet, I will ignore them completely. Our equation is then:

$$-\triangle u = f$$

Multiply this with testfunction v, and integrate over $\Omega = (0,1)^2$:

$$\begin{aligned} -\int_\Omega \triangle uvdx &= \int_\Omega fvdx \\ \iff \int_\Omega \bigtriangledown u \cdot \bigtriangledown vdx &= \int_\Omega fvdx \end{aligned}$$

This is our variatonal form, that we write in FEniCS. To get it we use partial integration. We also need to find $f = -\triangle u$. Since we know u this is simple.

$$\begin{aligned} f &= -\triangle u = -\triangle sin(k\pi x)cos(l\pi y) \\ &= ((k\pi)^2 + (l\pi)^2)sin(k\pi x)cos(l\pi y) \end{aligned}$$

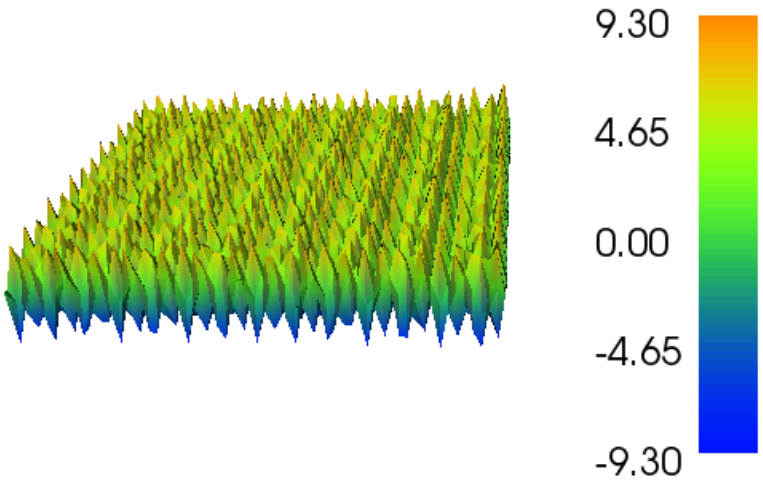

Figure 2: Plot of numerical solution u for $k = l = 100$ and $h = 64$. Now the values of u exceed both -1 and 1, and it also does not look smooth. This is due to aliasing.

In this exercise we were supposed to find the L^2 and H^1 error when solving our equation numerical for both elements of order 1 and 2, and for different values of k and l. To make it short I will only present my results for $k = 1$. The code for this exercise is added at the end, and the results are added at the end of the code. What we see from the error, is that it is small for k and l small, but big when k or l get big. This is especially true for the H^1 error. The reason for this is that big k and l means that our exact solution will "go up and down" very quickly, and since we are using a quite coarse mesh, we will get aliasing. To illustrate this I have added the plots of our numerical solutions for $k = l = 1$ and $k = l = 100$ when $\frac{1}{h} = 64$ and using $P1$ elements.

c) Since I implemented the norm in a), I have measured the convergence rate using all equations and errors, and also the convergence rate for each problem separately. This means I have tried to fit

$$\frac{||u - u_h||_{H^p}}{||u||_{H^{q+1}}} \quad \text{to} \quad Ch^\alpha \tag{3}$$

where u is exact solution to the equation for all l and k, and u_h is the same

numerically. I also calculated convergence rate while holding k and l fixed, using the following:

$$||u - u_h||_{H^p} \text{ to } Ch^{\alpha} \tag{4}$$

Using $P1$ elements I got the following result when estimating convergence for (3):

	Convergence rate	Constant
L^2	1.776496	0.041508
H^1	0.911748	0.181145

For $P2$ the result was:

	Convergence rate	Constant
L^2	2.570886	0.000820
H^1	1.491997	0.003164

Theoretically we would expect

$$\frac{||u - u_h||_p}{||u||_{q+1}} \leq Ch^{q+1-p}$$

In the expression above q represent the order of the element used in our finite element method. We see that our numerical estimates of the convergence rate in all cases are lower than what we would expect. This could indeed be a result of the aliasing we get because of our coarse mesh. As I said I also checked convergence for (k, l) pairs separately, and this is the L^2 convergence rates I got for $P1$ elements :

	$k = 1$	$k = 10$	$k = 100$
$l = 1$	1.980241	1.665838	2.253570
$l = 10$	1.663255	1.190830	1.298707
$l = 100$	2.265711	1.367577	2.302735

One remark to these numbers, are that even though the convergence rate is good for k and l around 100, the constant is in these cases really big, and it is therefore simple to get good convergence. The bad convergence occurs around k and l equal to 10. This is probably because this is a borderline case where our mesh is almost good enough to avoid aliasing. The rest of the results is found in the code.

Exercise 2

a) Assume our solution is on the form $u(x, y) = X(x)Y(y)$. If we plug this into our equation and assume that both X and Y are nonzero, we get:

$$-\mu(X''Y + XY'') + X'Y = 0 \iff \frac{-\mu X'' + X'}{X} - \mu\frac{Y''}{Y} = 0$$
$$\iff -\mu X'' + X' = \lambda X \text{ and } \mu Y'' = \lambda Y$$

Now lets look at the boundary conditions, starting withe the Dirichlet conditions:

$$u(0,y) = 0 \iff X(0)Y(y) = 0 \Rightarrow X(0) = 0$$

Since $Y(y) = 0$ would be a contradiction.

$$u(1,y) = 1 \iff X(1)Y(y) = 1 \Rightarrow Y(y) = 1/X(1)$$

This means that $Y(y)$ is a constant. This does not contradict our Neumann boundary conditions, since they say that the y-derivative is zero at $y = 0$ and $y = 1$. This means that our PDE is really an ODE on the form:

$$\begin{cases} -\mu X''(x) + X'(x) = 0 \\ X(0) = 0,\ X(1) = 1 \end{cases}$$

This gives us:

$$\mu X'(x) = X(x) + C \iff (X(x)e^{\frac{-x}{\mu}})' = Ce^{\frac{-x}{\mu}} \tag{5}$$

$$\iff X(x) = C' + De^{\frac{x}{\mu}} \tag{6}$$

Our boundary terms yields

$$C' = -D$$
$$C' + De^{\frac{1}{\mu}} = 1$$

The solution to this system is:

$$C' = \frac{1}{1 - e^{\frac{1}{\mu}}}$$
$$D = -\frac{1}{1 - e^{\frac{1}{\mu}}}$$

Putting this into the general solution gives us:

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

and

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

b)To solve the equation numerically I need to derive the variational form of the equation. Since our boundary conditions are a mix of homogeneous

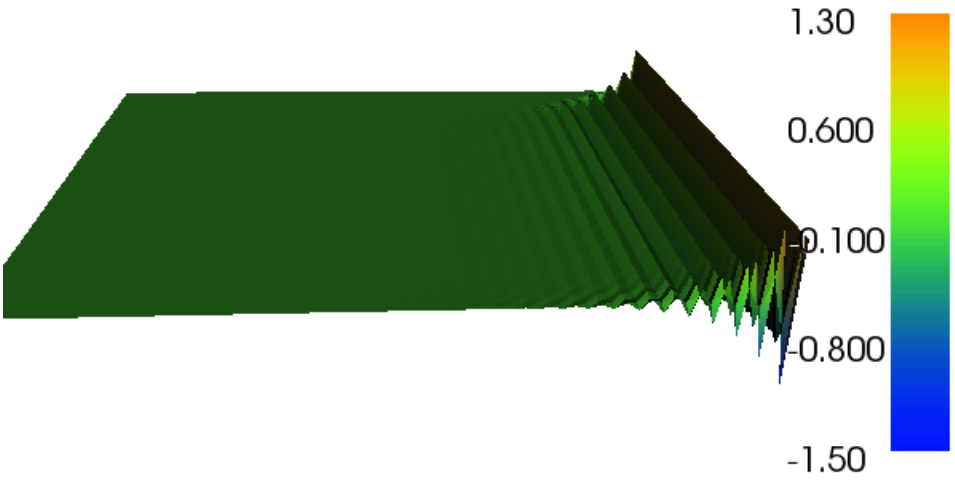

Figure 3: Plot of numerical solution u for $\mu = 0.001$ and $h = 64$, using normal Galerkin method. Notice oscillations close to $x = 1$.

Neumann and non-homogeneous Dirichlet, I will ignore them completely. Our equation is then:

$$-\mu \triangle u + u_x = f$$

Multiply this with testfunction v, and integrate over $\Omega = (0,1)^2$

$$\int_\Omega (-\mu \triangle u + u_x)vdx = \int_\Omega fvdx$$
$$\iff \int_\Omega \bigtriangledown u \cdot \bigtriangledown vdx + \int_\Omega u_x vdx = \int_\Omega fvdx$$

This is our variatonal form, that we write in FEniCS. To get it we use partial integration.

The code is attached at the end together with the results. Could add, that to be able to define the exact solution for small μ, I did the following approximation trick:

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

This makes sense since $e^{-\frac{1}{\mu}} \approx 0$ when μ is small.

As we see from the error results we get big errors in both norms when μ is small. To illustrate what happens I have plotted the numerical solution for $\mu = 0.001$ and $h = 64$. What we observe is oscillations close to $x = 1$, where we now that the exact solution grows very fast.

c) Lets look at the convergence rate α and constant C, I get from L^2 and H^1 for different μ.

	$\mu = 1$	$\mu = 0.1$	$\mu = 0.01$	$\mu = 0.001$	$\mu = 0.0001$
L^2 α	1.999763	1.975224	1.467166	1.299193	1.802562
L^2 C	0.044866	0.735446	3.339322	12.052507	301.737979
H^1 α	0.999856	0.978303	0.462191	0.184035	0.698340
H^1 C	0.212219	4.229330	19.327090	44.796474	989.689148

As we see from these results, we get what we would expect for big μ, but when we let μ get smaller, the convergence rate gets worse and the constant gets big. See that the convergence gets better again for the last μ value, but this is because error is so big that the convergence rate almost doesn't matter. The problem is that we get big oscillation near $x = 1$. We see this by plotting, or that the $H1$ error gets big.

d) Want to implement the streamline upwind Petrov-Galerkin method. The notes describe how to do this when we use first order Lagrange elements. The only thing we have to add to our variational form is the term:

$$\beta \int_\Omega (v \cdot \bigtriangledown u)(v \cdot \bigtriangledown w) dx = \beta \int_\Omega u_x w_x dx$$

An important question is how to choose β. I choose to use $\beta \approx h$. This seemed to work. However, when I use a changing β, I can no longer use the sd norm to calculate the convergence rate of the method.

As above the code is attached below, and the error and convergence of the method is in the code as well. See that the L^2 error and H^1 error is much smaller for SUPG method then for the regular Galerkin method when μ is small. However, the convergence rate for L^2 is now a lot worse, and for H^1, the H^1 norm actually increases. To illustrate how much better SUPG is I have added a plot of the solution to the same problem I plotted above using SUPG. This looks a lot closer to the true solution, then what what we got earlier, i.e. no numerical artifacts.

Code for Exercise 1

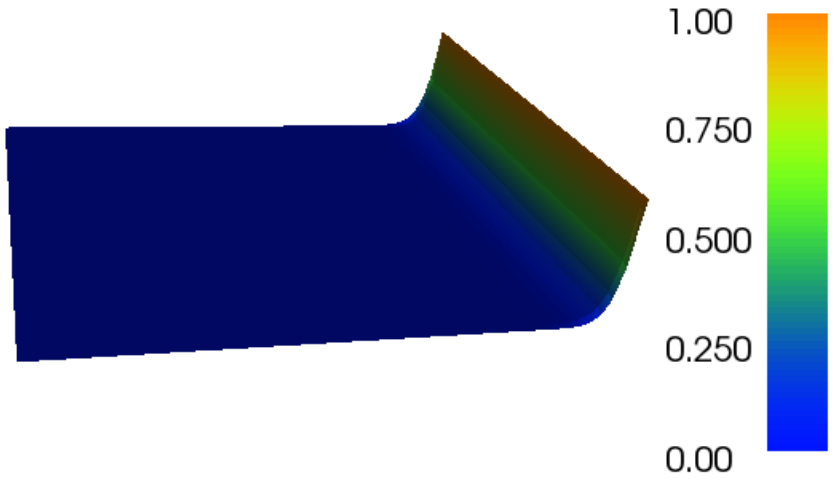

Figure 4: Plot of numerical solution u for $\mu = 0.001$ and $h = 64$ using the SUPG method. Notice that the graph is smooth, even at $x = 1$.

```
from dolfin import *
from numpy import pi,matrix,sqrt,diagflat,zeros,vstack,ones,log,array,size
#from scipy.special import factorial as fac
from math import factorial as fac
from scipy import linalg


#This function is an implementation of the expression for the norm found
#in exerccise 1a.
def Hp_norm(p,k,l):
    s = 0

    for i in range(p+1):
        for j in range(i+1):
            s = s + (k*pi)**(2*(i-j))*(l*pi)**(2*j)*fac(i)/(fac(j)*fac(i-

    return sqrt(0.25*s)


#handeling the boundary, which is dirichlet only for
#x=0 and x=1
```

```
def Dirichlet_boundary(x, on_boundary):
    if on_boundary:
        if x[0] == 0 or x[0] == 1:
            return True
        else:
            return False
    else:
        return False

#lots of lists to store errors and convergence reates;
L2 = [[],[]]
H1 = [[],[]]
h_val = [[],[]]
con = [[ [],[] ],[ [],[] ]]

L2_print =[[[],[],[]],[[],[],[]]]
H1_print =[[[],[],[]],[[],[],[]]]


#solving the equation for a lot of different parameters, p being order of
#elements and k and l having the same meaning as in exercise
for p in [1,2]:
    for k in [1,10,100]:
        for l in [1,10,100]:

            #even more lists to store errors
            l2 = []
            h1 = []
            hv = []

            #Size of mesh parameter
            for h in [8,16,32,64]:

                #define mesh
                mesh = UnitSquareMesh(h,h)


                #define the functionspace and the space where the
                #exact solution should be
                V = FunctionSpace(mesh,'Lagrange',p)
                V2 = FunctionSpace(mesh,'Lagrange',p+3)

                #f= -laplace(ue)
                f = Expression('pi*pi*sin(%e*pi*x[0])*cos(%e*pi*x[1])*%e'
```

```
                %(k,l,l**2+k**2))

#ue given
ue = Expression('sin(%e*pi*x[0])*cos(%e*pi*x[1])'%(k,l))
#Dirichlet boundary function
g = Constant(0)


# define the weak formulation
u = TrialFunction(V)
v = TestFunction(V)

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


#give the boundary
bc = DirichletBC(V,g,Dirichlet_boundary)

#solve the equation
U = Function(V)
solve(a==L,U,bc,solver_parameters={"linear_solver": "cg"}

#interpolate the exact solutionb into the V2 space
Ue = interpolate(ue,V2)

#measure the error in L2 and h1 norm
A =errornorm(U,Ue)
B = errornorm(U,Ue,'H1')

#store the error in different ways
L2[p-1].append(A/Hp_norm(p+1,k,l))
H1[p-1].append(B/Hp_norm(p+1,k,l))
h_val[p-1].append( mesh.hmin())

l2.append(A)
h1.append(B)
hv.append(mesh.hmin())
if k==1:
    L2_print[p-1][int(log(l)/log(10))].append(A)
    H1_print[p-1][int(log(l)/log(10))].append(B)

if k==1 and l==1 and h==64 and p==1:
    plot(U)
```

```
                    interactive()
                if k==100 and l==100 and h==64 and p==1:
                    plot(U)
                    interactive()

            #calculate convergence using least square for each set of
            #parameters.
            Q1 = vstack([log(array(hv)),ones(len(hv))]).T
            con[p-1][0].append(linalg.lstsq(Q1, log(array(l2)))[0])
            con[p-1][1].append(linalg.lstsq(Q1, log(array(h1)))[0])


#calculate and print convergence for both p1 and p2 elements using all
#the results.
print "****************************"
for i in range(2):

    Q1 = vstack([log(array(h_val[i])),ones(len(h_val[i]))]).T

    L2LS = linalg.lstsq(Q1, log(array(L2[i])))[0]
    H1LS = linalg.lstsq(Q1, log(array(H1[i])))[0]
    print
    print "-----------------oreder =%d------------------------ "%( i+1)
    print "L2 convergence =%f and constant=%f" %(L2LS[0],exp(L2LS[1]))
    print
    print "H1 convergence =%f and constant=%f" %(H1LS[0],exp(H1LS[1]))
print "****************************"
print

#print out the errrors for k=1
for i in range(2):
    print
    print "Error for element order %d" %(i+1)
    print
    for j in range(3):
        print"----------------------"
        print "<<<<<<<error for k=1 and l=%d>>>>>>>" % 10**j
        print
        print "L2 error: ", L2_print[i][j]
        print
        print "H1 error: ", H1_print[i][j]
        print"----------------------"
print
```

```
for i in range(2):
    print "***************************"
    print "convergence rates for elements of order %d" %(i+1)
    print "***************************"
    for j in range(9):
        cL2 =con[i][0][j]
        cH1 =con[i][1][j]
        print "--------------------"
        print "k=%d and l=%d" %(10**(j%3),10**(j/3))
        print
        print "L2 convergence =%f and constant=%f" %(cL2[0],exp(cL2[1]))
        print
        print "H1 convergence =%f and constant=%f" %(cH1[0],exp(cH1[1]))
        print "--------------------"
```

```
"""
terminal> python Exercise1.py
***************************

-----------------oreder =1-------------------------
L2 convergence =1.776496 and constant=0.041508

H1 convergence =0.911748 and constant=0.181145

-----------------oreder =2-------------------------
L2 convergence =2.570886 and constant=0.000820

H1 convergence =1.491997 and constant=0.003164
***************************


Error for element order 1

--------------------
<<<<<<<error for k=1 and l=1>>>>>>>

L2 error:  [0.032766238358843174, 0.008462150927576493, 0.002133162850145

H1 error:  [0.43611616866425396, 0.218104587006901, 0.10904724617721379,
```

```
_____
_____
<<<<<<<error for k=1 and l=10>>>>>>>

L2 error:  [0.6722330538591342, 0.2446180327889003, 0.07860246093967867,

H1 error:  [15.899204235889268, 9.124783303768643, 4.615527564530228, 2.2
_____
_____
<<<<<<<error for k=1 and l=100>>>>>>>

L2 error:  [191.99035176204282, 262.5175888557975, 3.0125832896210762, 4.

H1 error:  [2761.707144295842, 3506.665166616119, 312.2994681818063, 432.
_____

Error for element order 2

_____
<<<<<<<error for k=1 and l=1>>>>>>>

L2 error:  [0.0005687944394087238, 6.932977495255205e-05, 8.6111120541648

H1 error:  [0.03314085488599447, 0.008386636058147628, 0.002105368509165 3
_____
_____
<<<<<<<error for k=1 and l=10>>>>>>>

L2 error:  [0.3263570278412481, 0.025207667472855206, 0.00288629390508115

H1 error:  [8.791728328179158, 2.1875920019489614, 0.5710006773964853, 0.
_____
_____
<<<<<<<error for k=1 and l=100>>>>>>>

L2 error:  [289.3910615468301, 91.85959608956985, 5.776418252203544, 1.79

H1 error:  [3775.694834292425, 1225.1062671080413, 553.7172353393878, 183
_____

**************************
convergence rates for elements of order 1
**************************
_____
```

k=1 and l=1

L2 convergence =1.980241 and constant=1.021877

H1 convergence =0.999942 and constant=2.466974

---

k=10 and l=1

L2 convergence =1.665838 and constant=12.995948

H1 convergence =0.938442 and constant=84.342502

---

k=100 and l=1

L2 convergence =2.253570 and constant=14999.996858

H1 convergence =1.151324 and constant=26031.249021

---

k=1 and l=10

L2 convergence =1.663255 and constant=12.830047

H1 convergence =0.944926 and constant=85.907784

---

k=10 and l=10

L2 convergence =1.190830 and constant=6.013075

H1 convergence =0.740449 and constant=98.137533

---

k=100 and l=10

L2 convergence =1.298707 and constant=358.875264

H1 convergence =0.576514 and constant=2884.015411

---

k=1 and l=100

```
L2 convergence =2.265711 and constant=15356.475296

H1 convergence =1.152285 and constant=26134.262986
--------------------
--------------------
k=10 and l=100

L2 convergence =1.367577 and constant=442.285878

H1 convergence =0.585193 and constant=2964.549325
--------------------
--------------------
k=100 and l=100

L2 convergence =2.302735 and constant=14499.701518

H1 convergence =1.116652 and constant=30563.304129
--------------------
**************************
convergence rates for elements of order 2
**************************
--------------------
k=1 and l=1

L2 convergence =3.015059 and constant=0.104979

H1 convergence =1.991671 and constant=1.048373
--------------------
--------------------
k=10 and l=1

L2 convergence =3.271422 and constant=83.036484

H1 convergence =1.971545 and constant=265.580690
--------------------
--------------------
k=100 and l=1

L2 convergence =2.598631 and constant=30848.599689

H1 convergence =1.422809 and constant=42797.018952
--------------------
--------------------
k=1 and l=10
```

```
L2 convergence =3.275512 and constant=83.936394

H1 convergence =1.967469 and constant=260.533783
--------------------
--------------------
k=10 and l=10

L2 convergence =2.874872 and constant=74.397249

H1 convergence =1.705687 and constant=375.021456
--------------------
--------------------
k=100 and l=10

L2 convergence =1.392888 and constant=327.276881

H1 convergence =0.766741 and constant=3781.919625
--------------------
--------------------
k=1 and l=100

L2 convergence =2.598767 and constant=30869.372506

H1 convergence =1.423052 and constant=42827.465163
--------------------
--------------------
k=10 and l=100

L2 convergence =1.388960 and constant=325.309665

H1 convergence =0.765313 and constant=3760.090085
--------------------
--------------------
k=100 and l=100

L2 convergence =2.721865 and constant=38942.470253

H1 convergence =1.413687 and constant=57275.977178
--------------------


""""
```

Code for Exercise 1

```
from dolfin import *
from numpy import pi,matrix, sqrt, diagflat,zeros,vstack,ones,log,array,e
#from scipy.special import factorial as fac
from math import factorial as fac
from scipy import linalg




#function to define the Dirichlet boundary
def Dirichlet_boundary(x, on_boundary):
    if on_boundary:
        if x[0] == 0 or x[0] == 1:
            return True
        else:
            return False
    else:
        return False

#lists to store errors and convergence
con = []
errorL2 = []
errorH1 = []

#Solve the equaton for different mu
for my in [1,0.1,0.01,0.001,0.0001]:

    #more error storing
    L2 = []
    H1 = []
    h_val = []

    #size of mesh
    for h in [8,16,32,64]:

        #define mesh and functionspaces
        mesh = UnitSquareMesh(h,h)
        V = FunctionSpace(mesh,'Lagrange',1)
        V2 = FunctionSpace(mesh,'Lagrange',1+3)
```

```
        #define right hand side function
        f = Constant(0)

        #define exact solution. do trick to properly get what I want when
        if my>0.001:
            ue = Expression('(1-exp(x[0]/%e))/(1-exp(1/%e))'%(my,my))
        else:
            ue = Expression('exp((x[0]-1)/%e)'%my)

        #g is function on the boundary u(0,y)=0 u(1,y)=1
        g = Expression('x[0]')


        #define variational formulatin
        u = TrialFunction(V)
        v = TestFunction(V)

        MY = Constant(my)

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


        #define boudary conditions
        bc = DirichletBC(V,g,Dirichlet_boundary)

        #solve the equation
        U = Function(V)
        solve(a==L,U,bc)

        #find the error in L2 and H1 norm
        Ue = interpolate(ue,V2)

        L2.append(errornorm(U,Ue))
        H1.append(errornorm(U,Ue,'H1'))
        h_val.append( mesh.hmax())

        if my==0.001 and h==64:
            plot(U)
            interactive()
    #calculate the convergence rate for each mu
    errorL2.append(L2)
    errorH1.append(H1)
    Q = vstack([log(array(h_val)),ones(len(h_val))]).T
```

```
    con.append(linalg.lstsq(Q, log(array(L2)))[0])
    con.append(linalg.lstsq(Q, log(array(H1)))[0])


for i in range(5):
    print "----------------my=%e----------------------" % (0.1**i)
    print "L2 Error: ", errorL2[i]
    print
    print "H1 Error: ", errorH1[i]
    print
    print "L2 convergence=%f , constant=%f " %(con[2*i][0],exp(con[2*i][1
    print
    print "H1 convergance=%f , constant=%f " %(con[2*i+1][0],exp(con[2*i+
    print "-----------------------------------------------"
    print

"""
terminal> python Exercise2.py

----------------my=1.000000e+00----------------------
L2 Error:  [0.0014024911398510243, 0.00035075775846229513, 8.769838157263

H1 Error:  [0.037522413566527565, 0.01876559984230818, 0.0093833789669410

L2 convergence=1.999763 , constant=0.044866

H1 convergance=0.999856 , constant=0.212219
-----------------------------------------------

----------------my=1.000000e-01----------------------
L2 Error:  [0.023747305281734955, 0.006176861546736365, 0.001561325220945

H1 Error:  [0.7692373698326008, 0.39838931594977284, 0.2010768389408248,

L2 convergence=1.975224 , constant=0.735446

H1 convergance=0.978303 , constant=4.229330
-----------------------------------------------

----------------my=1.000000e-02----------------------
L2 Error:  [0.2389653886603793, 0.10398976824872899, 0.03814192062101558,

H1 Error:  [7.7969983422840405, 7.008644352418786, 5.086479716611928, 2.9
```

```
L2 convergence=1.467166  , constant=3.339322

H1 convergance=0.462191  , constant=19.327090
-----------------------------------------------

----------------my=1.000000e-03----------------
L2 Error:  [1.4205871191242874, 0.45478843938509783, 0.19169796112898158,

H1 Error:  [36.08584314601011, 25.48364599592582, 23.46005066643907, 24.2

L2 convergence=1.299193  , constant=12.052507

H1 convergance=0.184035  , constant=44.796474
-----------------------------------------------

----------------my=1.000000e-04----------------
L2 Error:  [13.576604419546138, 3.89939174630185, 0.9717085725730065, 0.3

H1 Error:  [313.9722174102841, 175.99443692797345, 99.33859332033042, 75.

L2 convergence=1.802562  , constant=301.737979

H1 convergance=0.698340  , constant=989.689148
-----------------------------------------------


"""
```

Code for Exercise 1

```
from dolfin import *
from numpy import pi,matrix, sqrt, diagflat,zeros,vstack,ones,log,array
from scipy import linalg


#Defining the Dirichlet boundary
def Dirichlet_boundary(x, on_boundary):
    if on_boundary:
        if x[0] == 0 or x[0] == 1:
            return True
        else:
```

```
            return False
    else:
        return False

#lists for storing errors and stuff
con = []
errorL2 = []
errorH1 = []


#solve equation for diffrent mu
for my in [1,0.1,0.01,0.001,0.0001]:

    #more lists for storing
    L2 = []
    H1 = []
    h_val = []

    #size of mesh
    for h in [8,16,32,64]:

        #define mesh and
        mesh = UnitSquareMesh(h,h)
        beta = mesh.hmax()
        V = FunctionSpace(mesh,'Lagrange',1)
        V2 = FunctionSpace(mesh,'Lagrange',1+3)


        #right hand side in equation
        f = Constant(0)

        #exact solution + trick
        if my==1:
            ue = Expression('(1-exp(x[0]))/(1-exp(1))')
        else:
            ue = Expression('exp((x[0]-1)/%e)'%my)

        #boundary function u(0,y)=0 u(1,y)=1
        g = Expression('x[0]')


        #setting up variational formulation
        u = TrialFunction(V)
        v = TestFunction(V)
```

```
            MY = Constant(my)
            Beta = Constant(beta)

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


            #boundary
            bc = DirichletBC(V,g,Dirichlet_boundary)

            #solve system
            U = Function(V)
            solve(a==L,U,bc)

            #calculate the error etc
            Ue = interpolate(ue,V2)


            L2.append(errornorm(U,Ue))
            H1.append(errornorm(U,Ue,'H1'))
            h_val.append( mesh.hmax())
            if my==0.001 and h==64:
                plot(U)
                interactive()

        errorL2.append(L2)
        errorH1.append(H1)
        #find convergence rate for each mu using least square.

        Q = vstack([log(array(h_val)),ones(len(h_val))]).T
        con.append(linalg.lstsq(Q, log(array(L2)))[0])
        con.append(linalg.lstsq(Q, log(array(H1)))[0])

    for i in range(5):
        print "----------------------my=%f----------------------" % (0.1**i)
        print "L2 Error: ", errorL2[i]
        print
        print "H1 Error: ", errorH1[i]
        print
        print "L2 convergence=%f, Constant=%f "% (con[2*i][0],exp(con[2*i][1]
        print
        print "H1 convergance%f, Constant=%f "%( con[2*i+1][0], exp(con[2*i+
        print "------------------------------------------------------"
```

```
    print
"""
terminal> python SUPG_Exersise.py
--------------------my=1.000000--------------------
L2 Error:  [0.01416838752283591, 0.0073698629559538304, 0.003766948027708

H1 Error:  [0.058124707898786016, 0.030553163128422527, 0.015691211257080

L2 convergence=0.965154, Constant=0.076007

H1 convergance0.956887, Constant=0.308063
----------------------------------------------------


--------------------my=0.100000--------------------
L2 Error:  [0.19247496357120064, 0.11447613838596869, 0.06286471681033422

H1 Error:  [1.291133318726959, 0.8952364831423733, 0.5543188017552529, 0.

L2 convergence=0.849216, Constant=0.866334

H1 convergance0.679910, Constant=4.414590
----------------------------------------------------


--------------------my=0.010000--------------------
L2 Error:  [0.28346579869290073, 0.19326250254022131, 0.12513237445344977

H1 Error:  [5.62404968238124, 6.118072592987696, 5.569677858352143, 4.491

L2 convergence=0.627636, Constant=0.863301

H1 convergance0.110835, Constant=7.365285
----------------------------------------------------


--------------------my=0.001000--------------------
L2 Error:  [0.28120110865768316, 0.200020969882777, 0.14226112534474597,

H1 Error:  [6.016784489921939, 8.508281095933974, 12.03054605209179, 16.5

L2 convergence=0.492342, Constant=0.660256

H1 convergance-0.486715, Constant=2.604413
----------------------------------------------------


--------------------my=0.000100--------------------
```

```
L2 Error:  [0.28047072407456436, 0.1989474319782039, 0.14076212733913698,
H1 Error:  [6.015360380217439, 8.50450651604375, 12.027273462389585, 17.0
L2 convergence=0.497769, Constant=0.664928
H1 convergance -0.499934, Constant=2.529147
```

"""