Appendix

Python code

Task 1b)

```python

#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from dolfin import * #FEniCS
from math import *

for i in [1,2]:

    for N in[8,16,32,64]: # h=(b-a)/N=1/N
        mesh = UnitSquareMesh(N,N)
        V = FunctionSpace(mesh, "Lagrange", i)
        V2 = FunctionSpace(mesh, "Lagrange", i+2)

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

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

        for k in [1,10]:
            f = Expression('2*pi*pi*k*k*sin(pi*k*x[0])*cos(pi*k*x[1])', k=k, degree=i)
            uh = Function(V)

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

            solve(a == L, uh, bcs)

            uexp= Expression('sin(pi*k*x[0])*cos(pi*k*x[1])', k=k, degree=1)
            u_Exact= interpolate(uexp, V2)

            Error=abs(u_Exact-uh)

            L2_norm = errornorm(u_Exact, uh, 'l2', degree_rise=3)
            print 'i=%g ,1/h=%g ,k=%g' % (i, N, k)
            print 'L2 = %g' % (L2_norm)

            H1_norm = errornorm(u_Exact, uh, 'H1', degree_rise=3)
            print 'H1 = %g' % (H1_norm)
            print '\n'
```

Task 1c)

```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

from dolfin import * #FEniCS
import matplotlib.pyplot as plt
from numpy import *
from termcolor import colored
set_log_active(False)

# least squares method:              y =     A   *   x    + b
#                             ln(Error) = ln(h) *   k   + ln(C_k)
for Error in ['l2','H1']:
    print '\n'
    print 'ErrorNorm = %s' % Error
    for i in [1, 2]:
        print '\n'
        print '---------------------------------------------------'
        print 'order = %g' % i
        for k in [1, 10]:
            X=[]
            Y=[]
            print '---------------------------------------------------'
            print 'k=%g' % k
            for N in[8, 16, 32, 64]: # h = (b-a)/N = 1/N
                print 'Number of mesh points = %g' % N

                mesh = UnitSquareMesh(N,N)

                V = FunctionSpace(mesh, "Lagrange", i)
                V2 = FunctionSpace(mesh, "Lagrange", i+2)

                h = 1./N
                print 'h = %g' % h

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

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


                f = Expression('2*pi*pi*k*k*sin(pi*k*x[0])*cos(pi*k*x[1])', k=k, degree=i)
                uh = Function(V)

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

                solve(a == L, uh, bcs)

                uexp= Expression('sin(pi*k*x[0])*cos(pi*k*x[1])', k=k, degree=1)
                u_Exact= interpolate(uexp, V2)

                Error_Norm = errornorm(u_Exact, uh, Error, degree_rise=3)
                print 'Error Norm = %g' % Error_Norm

                X.append(log(h))
                Y.append(log(Error_Norm))

                if N != 8:
                    Xx = array(X)
                    Yy = array(Y)

                    A = vstack([Xx, ones(len(Xx))]).T

                    Conv_rate, lnC = linalg.lstsq(A, Yy)[0]
                    print 'Convergence Rate=%g , C=%g' %(Conv_rate ,exp(lnC))
                    if Error_Norm > (exp(lnC))*h**Conv_rate:
                        print colored('This Error Estimate is --> NOT
                       <-- valid! Err_Norm > C_k*h^k', 'red')


        f = lambda x: lnC + Conv_rate*x
        x = linspace(X[0], X[-1], 101)


        plt.plot(x, f(x), label='Checking k dependency of C_k*h^k')
        plt.legend(loc='upper left')
        plt.grid('on')
        plt.xlabel('log(h)')
        plt.ylabel('log(||u - uh||)')
        filename = "ErrNorm_%s_PolDegree_%d.png" % (Error,i)
        plt.savefig(filename)
        plt.show()
```

Task 2b)

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

from dolfin import * #FEniCS
from math import *
import numpy as np

for Error in ['l2','H1']:
    print '\n'
    print 'Error_Type = %s' % Error
    for i in [1, 2]:
        print '\n'
        print '--------------------------------------------------'
        print 'Polynomial_order = %g' % i
        for N in[8, 16, 32, 64]: # h = (b-a)/N = 1/N
            print '--------------------------------------------------'
            print 'Number of mesh points = %g' % N

            mesh = UnitSquareMesh(N,N)

            V = FunctionSpace(mesh, "Lagrange", i)
            V2 = FunctionSpace(mesh, "Lagrange", i+2)


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

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


            f = Constant(0.0)
            uh = Function(V)

            for mu in [1, 0.1, 0.01]:
                print 'Mu value = %s' % mu

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

                solve(a == L, uh, bcs)

                uexp= Expression('(exp(x[0]/mu)-1)/(exp(1./mu)-1)', mu=mu, degree=1)
                u_Exact= interpolate(uexp, V2)

                Error_Norm = errornorm(u_Exact, uh, Error, degree_rise=3)
                print 'Error Norm = %.3E' % Error_Norm
                print '\n'
```

Task 2c)

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from dolfin import * #FEniCS
from math import *
from numpy import *
from termcolor import colored
set_log_active(False)

for Error in ['l2','H1']:
    print '\n'
    print 'Error_Type = %s' % Error
    for i in [1, 2]:
        X=[]
        Y=[]
        print '\n'
        print '--------------------------------------------------'
        print 'Polynomial_order = %g' % i
        for N in[8, 16, 32, 64]: # h = (b-a)/N = 1/N
            print '--------------------------------------------------'
            print 'Number of mesh points = %g' % N
            print '\n'

            mesh = UnitSquareMesh(N,N)
            h = 1./N

            V = FunctionSpace(mesh, "Lagrange", i)
            V2 = FunctionSpace(mesh, "Lagrange", i+2)


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

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


            f = Constant(0.0)
            uh = Function(V)

            for mu in [1, 0.1, 0.01]:
                print 'Mu value = %s' % mu

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

                solve(a == L, uh, bcs)

                uexp= Expression('(exp(x[0]/mu)-1)/(exp(1./mu)-1)', mu=mu, degree=1)
                u_Exact= interpolate(uexp, V2)

                Error_Norm = errornorm(u_Exact, uh, Error, degree_rise=3)
                print 'Error Norm = %.3E' % Error_Norm

                X.append(log(h))
                Y.append(log(Error_Norm))

                if N != 8:
                    Xx = array(X)
                    Yy = array(Y)

                    A = vstack([Xx, ones(len(Xx))]).T

                    Conv_rate, lnC = linalg.lstsq(A, Yy)[0]
                    print 'Convergence Rate=%g , C=%g' %(Conv_rate ,exp(lnC))
                    if Error_Norm > (exp(lnC))*h**Conv_rate:
                        print colored('This Error Estimate is --> NOT <-- valid! Err_Norm > C_k*h^k', 'red')
                        print '\n'
                    else:
                        print colored('This Error Estimate is valid!', 'green')
                        print '\n'
```

Task 2d)

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

#Oblig 2d

from dolfin import * #FEniCS
from math import *
from numpy import *
from termcolor import colored
set_log_active(False)


for i in [1, 2]:
    X=[]
    Y=[]
    print '\n'
    print '--------------------------------------------------'
    print 'Polynomial_order = %g' % i
    for N in[8, 16, 32, 64]: # h = (b-a)/N = 1/N
        print '--------------------------------------------------'
        print 'Number of mesh points = %g' % N
        print '\n'

        mesh = UnitSquareMesh(N,N)
        h=mesh.hmin()
        Beta = 0.5*h

        V = FunctionSpace(mesh, "Lagrange", i)
        V2 = FunctionSpace(mesh, "Lagrange", i+3)


        u = TrialFunction(V)
        v = TestFunction(V)
        w = v+Beta*v.dx(0)

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


        f = Constant(0.0)
        uh = Function(V)

        for mu in [1, 0.1, 0.01]:
            print 'Mu value = %s' % mu

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

            solve(a == L, uh, bcs)

            uexp= Expression('(exp(x[0]/mu)-1)/(exp(1./mu)-1)', mu=mu, degree=i)
            u_Exact= interpolate(uexp, V2)

            e1 = (uh-u_Exact).dx(0)
            e2 = (uh-u_Exact).dx(1)
            Error_Norm = sqrt(mesh.hmin()*assemble(e1**2*dx)+mu*assemble((e1+e2)**2*dx))
            print 'Error Norm = %.3E' % Error_Norm

            deltau = uh.dx(0).dx(0)+uh.dx(1).dx(1)
            nablau = uh.dx(0)+uh.dx(1)
            u2 = (assemble(uh**2*dx+nablau**2*dx+deltau**2*dx))

            X.append(log(h))
            Y.append(log(Error_Norm/u2))

            if N != 8:

                Xx = array(X)
                Yy = array(Y)

                A = vstack([Xx, ones(len(Xx))]).T

                Conv_rate, lnC = linalg.lstsq(A, Yy)[0]
                print 'Convergence Rate=%g , C=%g' %(Conv_rate ,exp(lnC))

                if Error_Norm/u2 > (exp(lnC))*h**(3/2):
                    print colored('This Error Estimate is --> NOT <-- valid! Err_Norm > C_k*h^k', 'red')
                    print '\n'
                else:
                    print colored('This Error Estimate is valid!', 'green')
                    print '\n'
```