In [ ]:
```
########################################
### Name : Md Ziauddin Ridoy ############
### Matriculation : 220100676 ####################
```

In [ ]:
```
'''I couldn`t uderstand the problem properly,From my understanding Minimal trinangulated graphs
triangles 2 cornersvalue will be fixed i.e x1, x3 and for x2, there is boundary condition
available q|Ω = 1/2 − |x2 − 1/2|...which has minimal surface area to be calculated....'''


'''Considering function q which needs to contionus on Ω/bar and linear on each triangle
and added to that constructing a function A which is Rn->R that defines the surface area
of the graph q'''
```

In [2]:
```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
```

In [8]:
```
x1 = np.linspace(0, 1, 30, endpoint=False)   # start, end, num-points
x1

x3 = np.linspace(0, 1, 30, endpoint=False)
x3
```

Out[8]:
```
array([0.        , 0.03333333, 0.06666667, 0.1       , 0.13333333,
       0.16666667, 0.2       , 0.23333333, 0.26666667, 0.3       ,
       0.33333333, 0.36666667, 0.4       , 0.43333333, 0.46666667,
       0.5       , 0.53333333, 0.56666667, 0.6       , 0.63333333,
       0.66666667, 0.7       , 0.73333333, 0.76666667, 0.8       ,
       0.83333333, 0.86666667, 0.9       , 0.93333333, 0.96666667])
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
def ArmijoLineSearch(f, xk, pk, gfk, phi0, alpha0, rho=0.5, c1=1e-4):

    derphi0 = np.dot(gfk, pk)
    phi_a0 = f(xk + alpha0*pk)

    while not phi_a0 <= phi0 + c1*alpha0*derphi0:
        alpha0 = alpha0 * rho
        phi_a0 = f(xk + alpha0*pk)

    return alpha0, phi_a0
```

In [ ]:
```
def GradientDescent(f_grad, init, alpha=10e-3, tol=1e-3, max_iter=100000):
     # initialize x, f(x), and f'(x)
    xk = init
    fk = f(xk)
    gfk = f_grad(xk)
    gfk_norm = np.linalg.norm(gfk)

    num_iter = 0
    curve_x = [xk]
    curve_y = [fk]
    print('Initial condition: y = {:.4f}, x = {} \n'.format(fk, xk))
    # take steps
    while gfk_norm > tol and num_iter < max_iter:
        # determine direction
        pk = -gfk

        alpha, fk = ArmijoLineSearch(f, xk, pk, gfk, fk, alpha0=alpha)
        xk = xk + alpha * pk
        gfk = f_grad(xk)
        gfk_norm = np.linalg.norm(gfk)
        # increase number of steps by 1, save new x and f(x)
        num_iter += 1
        curve_x.append(xk)
        curve_y.append(fk)
        print('Iteration: {} \t y = {:.4f}, x = {}, gradient = {:.4f}'.
              format(num_iter, fk, xk, gfk_norm))
    # print results
    if num_iter == max_iter:
        print('\nGradient descent does not converge.')
    else:
        print('\nSolution: \t y = {:.4f}, x = {}'.format(fk, xk))

    return np.array(curve_x), np.array(curve_y)
```

In [ ]:
```
def SurfaceArea():
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: