In [ ]:

```python
import numpy as np
import sys

_input = sys.argv[1]
input_str = open(_input).read()

s = input_str.split('\n')
n,int_nodes = [int(i) for i in s[0].split(' ') if i]
x,y = [], []

for i in range(1,n + 1):
    x_i, y_i = [float(p) for p in s[i].split(' ') if p]
    x.append(x_i)
    y.append(y_i)


Nt = int(s[n+1])
t1, t2, t3 = [], [], []
for i in range(n+2,n+2+Nt):
    i1, i2, i3 = [int(p)-1 for p in s[i].split(' ') if p]
    t1.append(i1)
    t2.append(i2)
    t3.append(i3)

p = np.array([[-1,-1],
              [1,0],
              [0,1]]).T

A = np.zeros([int_nodes,int_nodes])
rhs = np.zeros(int_nodes)


for k in range(0, Nt):
    npt = [ t1[k], t2[k], t3[k] ]
    B_T = np.array([[x[npt[1]] - x[npt[0]], x[npt[2]] - x[npt[0]]],
                    [y[npt[1]] - y[npt[0]], y[npt[2]] - y[npt[0]]]])

    ar_triangle = abs(np.linalg.det(B_T)) / 2.0
    b = np.array([[y[npt[2]] - y[npt[0]], y[npt[0]] - y[npt[1]]],
                  [x[npt[0]] - x[npt[2]], x[npt[1]] - x[npt[0]]]])


    alpha = b.dot(p)
    for l in range(0,3):
        i = npt[l]
        if i >= int_nodes:
            continue

        for m in range(l,3):
            j = npt[m]
            if j >= int_nodes:
                continue

            alpha_lm = alpha[:,l].dot(alpha[:,m])
            A[i][j] = A[i][j] + alpha_lm / ( 4 * ar_triangle)

            if i is not j:
                A[j][i] = A[j][i] + alpha_lm / ( 4 * ar_triangle)

        rhs[i] = rhs[i] + ar_triangle * 2 *np.pi * np.pi* np.sin(np.pi *
x[i]) * np.sin(np.pi * y[i]) / 3.0
```

```
In [ ]: calculated = np.linalg.solve(A,rhs)
        actual = np.sin(np.pi * np.array(x[0:int_nodes])) * np.sin(np.pi * np.ar
        ray(y[0:int_nodes]))

        error = np.abs(calculated - actual)
        l2_error  = 0.0
        for k in range(0,Nt):
            npt = [ t1[k], t2[k], t3[k] ]
            B_T = np.array([[x[npt[1]] - x[npt[0]], x[npt[2]] - x[npt[0]]],
                            [y[npt[1]] - y[npt[0]], y[npt[2]] - y[npt[0]]]])

            ar_triangle = abs(np.linalg.det(B_T)) / 2.0
            for l in range(0,3):
                i = npt[l]
                if i >= int_nodes:
                    continue
                l2_error = l2_error + ar_triangle * error[i] * error[i] / 3.0

        print np.sqrt(l2_error)
```