Question 2

```
import distmesh as dm
from matplotlib import projections
from matplotlib.collections import LineCollection, PolyCollection
import numpy as np
import matplotlib.pyplot as plt


def plot_fem_mesh(nodes_x, nodes_y, elements):
   '''
   Code excerpt taken from:
https://stackoverflow.com/questions/52202014/how-can-i-plot-2d-fem-results-using-mat
plotlib
   '''
   plt.cla()
   plt.clf()
   for element in elements:
       x = [nodes_x[element[i]] for i in range(len(element))]
       y = [nodes_y[element[i]] for i in range(len(element))]
       plt.fill(x, y, edgecolor='black', fill=False)
   plt.scatter(nodes_x, nodes_y)
   plt.axis('equal')
   plt.title('Plotting FEM Mesh for circle domain')
   plt.xlabel('x-axis')
   plt.ylabel('y-axis')
   plt.savefig('2a.png')
   # plt.show()

def extract_boundary(p, t):
   boundary_edges = dm.boundedges(p, t)
   boundary_nodes = np.reshape(boundary_edges, -1)
   boundary_nodes = np.unique(boundary_nodes)
   return boundary_nodes

def build_elemental_stiffness_matrix(p, t):
   uvw = np.array([p[1]-p[2], p[2]-p[0], p[0]-p[1]])
   determinant = 0.5 * np.abs(uvw[1, 0] * uvw[2, 1] - uvw[1, 1] * uvw[2, 0])
   assert(determinant != 0)
   dim = t.shape[0]
   Ak = np.zeros((dim, dim))
   for i in range(0, dim):
```

```
        for j in range(0, dim):
            Ak[i, j] = np.dot(uvw[i], uvw[j])
    Ak = 0.25 * 1./determinant * Ak
    return Ak

def FEM_Stiffness_Matrix(p, triangles):
    num_nodes = p.shape[0]
    dim = 3
    A = np.zeros((num_nodes, num_nodes))
    for tr in triangles:
        p_tr = np.array([p[tr[0]], p[tr[1]], p[tr[2]]])
        Ak = build_elemental_stiffness_matrix(p_tr, tr)
        for i in range(0, dim):
            for j in range(0, dim):
                A[tr[i], tr[j]] += Ak[i, j]
    return A

def plot_sparsity_pattern(A):
    plt.cla()
    plt.clf()
    plt.spy(A)
    plt.show()

def enforce_dirichlet(A, boundary_nodes):
    A[boundary_nodes, :] = 0
    A[boundary_nodes, boundary_nodes] = 1.0
    return A

def RHS(p, triangles):
    num_nodes = p.shape[0]
    dim = p.shape[1]+1
    b = np.zeros(num_nodes)
    for tr in triangles:
        uvw = np.array([p[tr[1]]-p[tr[2]], p[tr[2]]-p[tr[0]], p[tr[0]]-p[tr[1]]])
        determinant = uvw[1, 0] * uvw[2, 1] - uvw[1, 1] * uvw[2, 0]
        for i in range(dim):
            b[tr[i]] += 1./6. * determinant
    return b

def plot_solution(x, y, t, u):
    plt.cla()
```

```python
    plt.clf()
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_trisurf(x, y, u, triangles=t, cmap=plt.cm.viridis)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()

def get_exact_solution(p):
    r_squared = p[:, 0] * p[:, 0] + p[:, 1] * p[:, 1]
    u = -0.25 * r_squared + 0.25
    return u

def discretize_and_solve_on_circle(h0=0.2):
    fd = lambda p: np.sqrt((p**2).sum(1))-1.0
    p, t = dm.distmesh2d(fd, dm.huniform, h0, (-1,-1,1,1))
    plot_fem_mesh(p[:, 0], p[:, 1], t)

    x = p[:, 0]
    y = p[:, 1]

    boundary_nodes = extract_boundary(p, t)

    A = FEM_Stiffness_Matrix(p, t)
    A = enforce_dirichlet(A, boundary_nodes)

    b = RHS(p, t)
    b[boundary_nodes] = 0

    u = np.linalg.solve(A, b)
    u_exact = get_exact_solution(p)

    print("h={} u={}".format(h0, np.linalg.norm(u)))

    # plt.cla()
    # plt.clf()
    # fig = plt.figure(figsize=plt.figaspect(0.5))
    # ax = fig.add_subplot(121, projection='3d')
    # ax.plot_trisurf(x, y, u, triangles=t, cmap=plt.cm.viridis)
    # ax.set_title("FEM Solution")
    # ax.set_xlabel("x-axis")
```

```python
    # ax.set_ylabel("y-axis")

    # ax = fig.add_subplot(122, projection='3d')
    # ax.plot_trisurf(x, y, u_exact, triangles=t, cmap=plt.cm.viridis)
    # ax.set_title("Exact Solution")
    # ax.set_xlabel("x-axis")
    # ax.set_ylabel("y-axis")
    # plt.title("Exact solution")
    # plt.savefig("2b_fem_vs_exact.png")

    # plt.cla()
    # plt.clf()
    # fig = plt.figure()
    # ax = fig.add_subplot(111, projection='3d')
    # ax.plot_trisurf(x, y, u-u_exact, triangles=t, cmap=plt.cm.viridis)
    # plt.xlabel("x-axis")
    # plt.ylabel("y-axis")
    # plt.title("Plotting error against exact solution")
    # plt.savefig("2b_error.png")
    # plot_solution(x, y, t, u_exact)

def discretize_and_solve_on_ellipse():
    fd = lambda p: (p[:, 0]**2/(4) + p[:, 1]**2/(1))-1.0
    [p,t]=dm.distmesh2d(fd,dm.huniform,0.2,(-2,-1,2,1))

    x = p[:, 0]
    y = p[:, 1]

    boundary_nodes = extract_boundary(p, t)

    A = FEM_Stiffness_Matrix(p, t)
    A = enforce_dirichlet(A, boundary_nodes)

    b = RHS(p, t)
    b[boundary_nodes] = 0

    u = np.linalg.solve(A, b)

    plt.cla()
    plt.clf()
    fig = plt.figure()
```

```python
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_trisurf(x, y, u, triangles=t, cmap=plt.cm.viridis)
    plt.xlabel("x-axis")
    plt.ylabel("y-axis")
    plt.title("Plotting FEM Solution on Ellipse")
    plt.savefig("2c_ellipse.png")

    # plot_solution(x, y, t, u)

def discretize_and_solve_on_polygon():
    pv = np.array([(-0.4,-0.5),(0.4,-0.2),(0.4,-0.7),(1.5,-0.4),(0.9,0.1),
                   (1.6,0.8),(0.5,0.5),(0.2,1.0),(0.1,0.4),(-0.7,0.7),
                   (-0.4,-0.5)])
    fd = lambda p: dm.dpoly(p, pv)
    p, t = dm.distmesh2d(fd, dm.huniform, 0.1, (-1,-1, 2,1), pv)

    x = p[:, 0]
    y = p[:, 1]

    boundary_nodes = extract_boundary(p, t)

    A = FEM_Stiffness_Matrix(p, t)
    A = enforce_dirichlet(A, boundary_nodes)

    b = RHS(p, t)
    b[boundary_nodes] = 0

    u = np.linalg.solve(A, b)

    plt.cla()
    plt.clf()
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_trisurf(x, y, u, triangles=t, cmap=plt.cm.viridis)
    plt.xlabel("x-axis")
    plt.ylabel("y-axis")
    plt.title("Plotting FEM Solution on Polygon")
    plt.savefig("2c_polygon.png")

    # plot_solution(x, y, t, u)
```

```python
def get_concave_mesh(h0):
    fd = lambda p: dm.ddiff(dm.drectangle(p,-1,1,-1,1), dm.dcircle(p,0,0,0.5))
    fh = lambda p: 0.05+0.3*dm.dcircle(p,0,0,0.5)
    p, t = dm.distmesh2d(fd, fh, h0, (-1,-1,1,1), [(-1,-1),(-1,1),(1,-1),(1,1)])
    return p, t

def discretize_and_solve_on_concave_region(h0):
    p, t = get_concave_mesh(h0)

    x = p[:, 0]
    y = p[:, 1]
    plot_fem_mesh(x, y, t)

    boundary_nodes = extract_boundary(p, t)
    # plt.scatter(p[boundary_nodes, 0], p[boundary_nodes, 1])
    # plt.show()

    A = FEM_Stiffness_Matrix(p, t)
    A = enforce_dirichlet(A, boundary_nodes)

    b = RHS(p, t)
    b[boundary_nodes] = 0

    u = np.linalg.solve(A, b)

    print("h={} Norm of u={}".format(h0, np.linalg.norm(u)))

    plt.cla()
    plt.clf()
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_trisurf(x, y, u, triangles=t, cmap=plt.cm.viridis)
    plt.xlabel("x-axis")
    plt.ylabel("y-axis")
    plt.title("Plotting FEM Solution on Concave region, h={}".format(h0))
    plt.savefig("2e_{}.png".format(h0))

    # plot_solution(x, y, t, u)

if __name__ == '__main__':
    discretize_and_solve_on_circle(h0=0.1)
```

```
    discretize_and_solve_on_circle(h0=0.2)
    discretize_and_solve_on_circle(h0=0.5)
    discretize_and_solve_on_ellipse()
    discretize_and_solve_on_polygon()
    discretize_and_solve_on_concave_region(0.01)
    discretize_and_solve_on_concave_region(0.02)
    discretize_and_solve_on_concave_region(0.05)
```

Question 5c

```
import math
import distmesh as dm
from matplotlib import projections
from matplotlib.collections import LineCollection, PolyCollection
import numpy as np
import matplotlib.pyplot as plt

# Plotting functions
def plot_fem_mesh(nodes_x, nodes_y, elements):
    '''
    Code inspiration:
https://stackoverflow.com/questions/52202014/how-can-i-plot-2d-fem-results-using-mat
plotlib
    '''
    plt.cla()
    plt.clf()
    for element in elements:
        x = [nodes_x[element[i]] for i in range(len(element))]
        y = [nodes_y[element[i]] for i in range(len(element))]
        x = [x[0], x[1], x[3], x[2]]
        y = [y[0], y[1], y[3], y[2]]
        plt.fill(x, y, edgecolor='black', fill=False)
    plt.axis('equal')
    plt.show()

def plot_solution(x, y, t, u):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(x, y, u, cmap=plt.cm.viridis)
    plt.xlabel('x')
```

```
    plt.ylabel('y')
    plt.show()

def plot_sparsity_pattern(A):
    plt.cla()
    plt.clf()
    plt.spy(A)
    plt.show()

# Assemble Stiffness Matrix
def build_elemental_stiffness_matrix(h):
    Ak = np.array( [[1./9.*(6+h*h),    1./18.*(-3+h*h),  1./18.*(-3+h*h),
1./36*(-12+h*h)],
                   [1./18.*(-3+h*h),  1./9.*(6+h*h),    1./36*(-12+h*h),
1./18.*(-3+h*h)],
                   [1./18.*(-3+h*h),  1./36*(-12+h*h),  1./9.*(6+h*h),
1./18.*(-3+h*h)],
                   [1./36*(-12+h*h),  1./18.*(-3+h*h),  1./18.*(-3+h*h),
1./9.*(6+h*h)]])
    return Ak

def FEM_Stiffness_Matrix(p, elements):
    num_nodes = p.shape[0]
    A = np.zeros((num_nodes, num_nodes))
    for ele in elements:
        h = p[ele[3]][0] - p[ele[0]][0]
        Ak = build_elemental_stiffness_matrix(h)
        for i in range(0, 4):
            for j in range(0, 4):
                A[ele[i], ele[j]] += Ak[i, j]
    return A

# Assemble RHS
def RHS(p, elements):
    num_nodes = p.shape[0]
    pi = np.pi
    const = 1./(2.*pi*pi)
    cos = np.cos
    sin = np.sin
    b = np.zeros(num_nodes)
    for ele in elements:
```

```
        xi = p[ele[0]][0]
        xiph = p[ele[3]][0]
        yi = p[ele[0]][1]
        yiph = p[ele[3]][1]
        h = xiph - xi

        b[ele[0]] +=  const * (  cos(pi * xi) - cos(pi * xiph) - cos(pi * yi) +
cos(pi * yiph) - h * pi * sin(pi * xi)   + h * pi * sin(pi * yi))
        b[ele[1]] += -const * ( -cos(pi * xi) + cos(pi * xiph) - cos(pi * yi) +
cos(pi * yiph) + h * pi * sin(pi * xi)   + h * pi * sin(pi * yiph))
        b[ele[2]] +=  const * ( -cos(pi * xi) + cos(pi * xiph) - cos(pi * yi) +
cos(pi * yiph) + h * pi * sin(pi * xiph) + h * pi * sin(pi * yi))
        b[ele[3]] +=  const * ( -cos(pi * xi) + cos(pi * xiph) + cos(pi * yi) -
cos(pi * yiph) + h * pi * sin(pi * xiph) - h * pi * sin(pi * yiph))
    return b

# Construct mesh
def get_linear_index(index, domain):
    return index[0] * domain[1] + index[1]

def get_grid_mesh(num_p= 5, num_el = 16):
    p = np.zeros((int(num_p*num_p), 2))
    t = np.zeros((num_el, 4))
    t = t.astype(np.int32)
    boundary_nodes = []

    x = np.linspace(0, 1, num_p)
    x, y = np.meshgrid(x, x, indexing='ij')

    p[:, 0] = x.reshape(-1)
    p[:, 1] = y.reshape(-1)

    num_cells = num_p - 1

    cell_idx = 0
    for cellx in range(num_cells):
        for celly in range(num_cells):
            dim_idx = 0
            for node1 in range(2):
                for node2 in range(2):
                    node = [cellx+node1, celly+node2]
```

```
                    linear_node_idx = get_linear_index(node, [num_cells+1,
num_cells+1])
                    # put node in element
                    t[cell_idx, dim_idx] = linear_node_idx
                    # Add node if boundary
                    if ((node[0] == 0 or node[0] == num_p-1) or (node[1] == 0 or
node[1] == num_p-1)):
                        boundary_nodes.append(linear_node_idx)
                    dim_idx += 1
            cell_idx += 1

    boundary_nodes = np.array(boundary_nodes)
    boundary_nodes = np.unique(boundary_nodes)

    return p, t, boundary_nodes

# Compute exact solution
def get_exact_solution(x, y):
    const = 1./(1.+np.pi*np.pi)
    return const * (np.cos(np.pi * x) - np.cos(np.pi * y))

def discretize_and_solve_on_mesh(num_elements = 16):
    num_points = int(math.sqrt(num_elements) + 1)
    p, t, _ = get_grid_mesh(num_p=num_points, num_el=num_elements)

    A = FEM_Stiffness_Matrix(p, t)

    b = RHS(p, t)

    u = np.linalg.solve(A, b)
    u_exact = get_exact_solution(p[:, 0], p[:, 1])
    error = np.linalg.norm(u - u_exact, np.inf)

    print("Num Elements = {} Error = |u - u_exact| = {}".format(num_elements, error))

    u = np.reshape(u, (num_points, num_points))
    u_exact = np.reshape(u_exact, (num_points, num_points))
    b = np.reshape(b, (num_points, num_points))

    x = np.linspace(0, 1, num_points)
    x, y = np.meshgrid(x, x, indexing='ij')
```

```
    plt.cla()
    plt.clf()
    fig = plt.figure(figsize=plt.figaspect(0.5))
    ax = fig.add_subplot(121, projection='3d')
    ax.plot_surface(x, y, u, cmap=plt.cm.viridis)
    ax.set_title("FEM Solution for Q5, num_elements={}".format(num_elements))
    ax.set_xlabel("x-axis")
    ax.set_ylabel("y-axis")

    ax = fig.add_subplot(122, projection='3d')
    ax.plot_surface(x, y, u, cmap=plt.cm.viridis)
    ax.set_title("Exact Solution for Q5, num_elements={}".format(num_elements))
    ax.set_xlabel("x-axis")
    ax.set_ylabel("y-axis")
    plt.savefig("5c_fem_vs_exact_{}.png".format(num_elements))

    # plot_solution(x, y, t, u)
    # plot_solution(x, y, t, u_exact)

    return error

if __name__ == '__main__':
    e1 = discretize_and_solve_on_mesh(num_elements=16)
    e2 = discretize_and_solve_on_mesh(num_elements=64)
    e3 = discretize_and_solve_on_mesh(num_elements=256)
    print("Error(h={})/Error(h={}) = {}".format(1./np.sqrt(16), 1./np.sqrt(64),
e1/e2))
    print("Error(h={})/Error(h={}) = {}".format(1./np.sqrt(64), 1./np.sqrt(256),
e1/e2))
```