20145271 Geum Kang Hyeon

K-means clustering on the spatial domain

Apply K-means algorithm to the regular grid of a spatial domain in two dimension with varying number of clusters.

The spatial domain can be represented by two matrices where one matrix represents the horizontal index and the other matrix represents the vertical index.

Define a distance between each spatial point (x_i, y_i) and a centroid (c_x^k, c_y^k) for cluster k using L2-norm square and L1-norm.

Visualize the result using color coding scheme that distinguishes different clusters.

Observe the trajectory of centroid during the optimization and the shape of the clusters depending on the distance.

[L1 Norm Energy]

$$\frac{1}{n}\sum_{x\in\Omega}|f(x)-m_c|$$

[L2 Norm Energy]

$$\frac{1}{n}\sum_{x\in\Omega}\|f(x)-m_c\|^2$$

where Ω denotes the coordinate domain and the number of pixels $|\Omega|$ is n, and m_c denotes the centroid for cluster c that is the cluster label of $f(x)$.

[Output Plot]

$g(x)=m_c$ where $label(x)=c$

Each pixel of the output plot $g(x)$ should be its centroid m_c where c is the cluster label of $g(x)$.

Code Explaination

The process consists of 7 main stages.

1. Generate two lists with coordinate information, assign a random label to each pixels
2. Find a centroid coordinate from randomly labeled data.
3. The distance is expressed by the L1 or L2 Norm between the actual coordinate value and the centeroid coordinate.
4. Update labeling with the value of argmin in distance.
5. Calculate energy according to the expression provided above.
6. Repeat until the difference between the previous and current energies is less than 0.01 (about 0.0001%).
7. Displays the plot corresponding to each K, each Norm and points at the centroid coordinate.
8. Displays the plot corresponding to each Energy Data

In [158]:

```
import matplotlib.pyplot as plt
import numpy as np
import math, random
import statistics

def norm(norm_type, source, target):
    if(norm_type == 0):
        return L1_distance(source, target)
    elif(norm_type == 1):
        return L2_distance(source, target)
    elif(norm_type == 2):
        return sum_of_square(source, target)

def L1_distance(source, target):
    sum = 0
    sum += abs(source[:] - target[:])
    return np.sum(sum)

def L2_distance(source, target):
    sum = sum_of_square(source, target)
    return math.sqrt(sum)

def sum_of_square(source, target):
    sum =  0
    sum += (source[:] - target[:]) ** 2
    return np.sum(sum)

def Vectorization(list):
    x = []
    y = []

    for i in range(len(list)):
        x.append(list[i][0])
        y.append(list[i][1])
    return x,y

def plot_data(row, col, random_label, centroid_coord, k, norm_type):

    colorMap = np.zeros(row*col, dtype = float)
    normalizedVector = np.zeros(k, dtype = float)

    temp = np.zeros((1, 2), dtype=float)
    temp[0] = [row, col]

    for i in range(k):
        if(norm_type == 0):
            normalizedVector[i] = (L1_distance([0,0], temp[0]) - L1_distance([0,0],centroid_coor
d[i])) / L1_distance([0,0], temp[0])
        else:
            normalizedVector[i] = (L2_distance([0,0], temp[0]) - L2_distance([0,0],centroid_coor
d[i])) / L2_distance([0,0], temp[0])

    for i in range(row * col):
        colorMap[i] = normalizedVector[random_label[i]]

    pixelMap = np.reshape(colorMap, (row,col))
    x,y = tuple(Vectorization(centroid_coord))

    if(norm_type == 0):
```

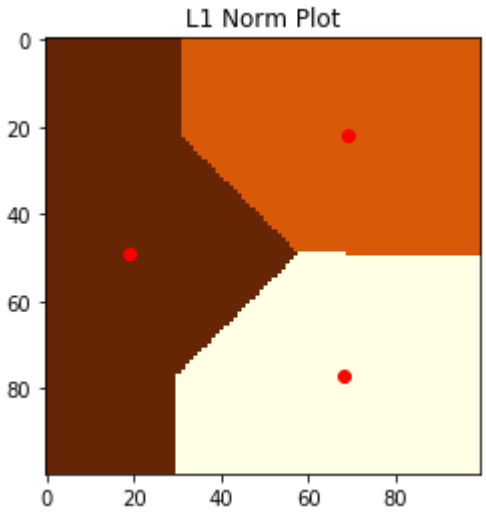
```
        plt.title("L1 Norm Plot")
    else:
        plt.title("L2 Norm Plot")
    plt.plot(y, x, marker='o', color='r', ls='')
    plt.imshow(pixelMap, cmap='YlOrBr')
    plt.show()

def plot_energy(energy_data, norm_type):
    if(norm_type == 0):
        plt.title("L1 Norm Energy")
    else:
        plt.title("L2 Norm Energy")
    plt.plot(energy_data)
    plt.show()


def clustering(row, col, k, norm_type):

    num_pixel = row * col

    coord_x = np.empty(num_pixel, dtype=int)
    coord_y = np.empty(num_pixel, dtype=int)
    random_label = np.empty(num_pixel, dtype=int)
    backup_label = np.empty(num_pixel, dtype=int)

    idx = 0
    for i in range(row):
        for j in range(col):
            coord_x[idx] = i
            coord_y[idx] = j
            random_label[idx] = np.random.randint(0,k,1)
            idx += 1

    energy = 0
    energy_data = []

    for loop in range(100):

        new_energy = 0

        centroid_x = np.zeros(k, dtype=float)
        centroid_y = np.zeros(k, dtype=float)
        centroid_count = np.zeros(k, dtype=int)
        countIdx = np.zeros(k, dtype=int)

        # centroid coord
        if(norm_type == 0):
            for idx in range(k):
                temp_x = []
                temp_y = []
                for i in range(num_pixel):
                    if(random_label[i] == idx):
                        temp_x.append(coord_x[i])
                        temp_y.append(coord_y[i])
                        centroid_count[idx] += 1
                if(centroid_count[idx] > 0):
                    centroid_x[idx] = statistics.median(temp_x)
                    centroid_y[idx] = statistics.median(temp_y)

        else:
            for i in range(num_pixel):
```

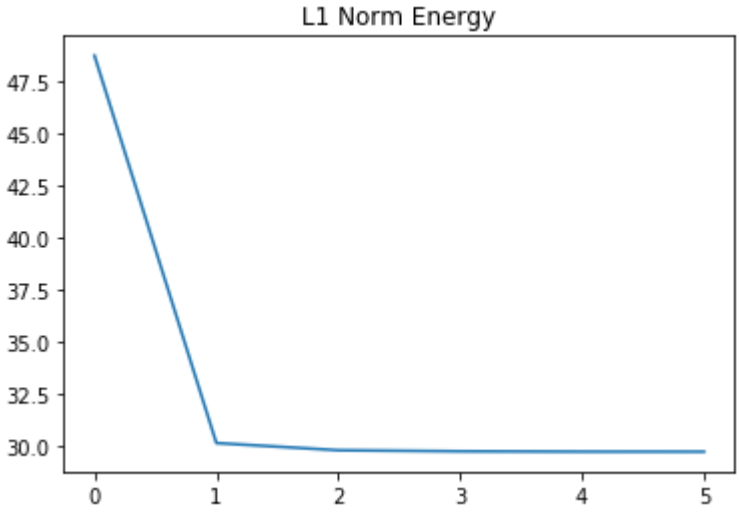
```
                centroid_x[int(random_label[i])] += coord_x[i]
                centroid_y[int(random_label[i])] += coord_y[i]
                centroid_count[int(random_label[i])] += 1

            for i in range(k):
                centroid_x[i] /= centroid_count[i]
                centroid_y[i] /= centroid_count[i]

        zero_check = False
        for i in range(k):
            if(centroid_count[i] == 0):
                centroid_x[i] = np.random.randint(0,row,1)
                centroid_y[i] = np.random.randint(0,col,1)

        if(zero_check == False):
             # compare coord
            for i in range(num_pixel):
                temp = [coord_x[i], coord_y[i]]
                centroid_coord = np.zeros((k, 2), dtype=float)
                distance = np.zeros(k, dtype=float)

                for index in range(k):
                    centroid_coord[index] = [round(centroid_x[index]), round(centroid_y[index])]
                    distance[index] = norm(norm_type, temp, centroid_coord[index])

                label = np.argmin(distance)
                new_energy += norm(norm_type*2, temp, centroid_coord[label])

                backup_label[i] = label

            random_label = backup_label
            # calculate energy
            new_energy /= num_pixel
            energy_data.append(new_energy)

            if energy == 0:
                energy = new_energy
            elif abs(new_energy - energy) < 0.01:
                plot_data(row, col, random_label, centroid_coord, k, norm_type)
                plot_energy(energy_data, norm_type)
                break
            else:
                energy = new_energy
```

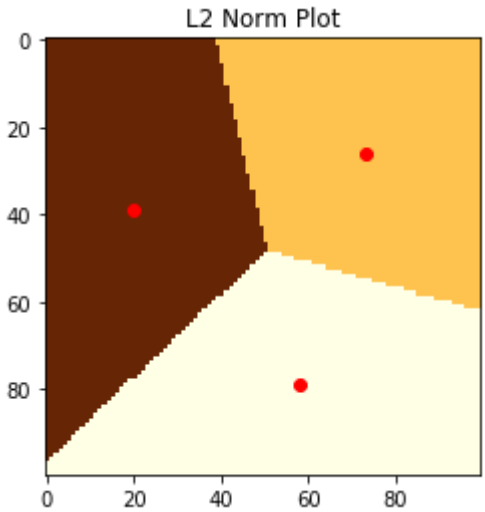
1) k is 3

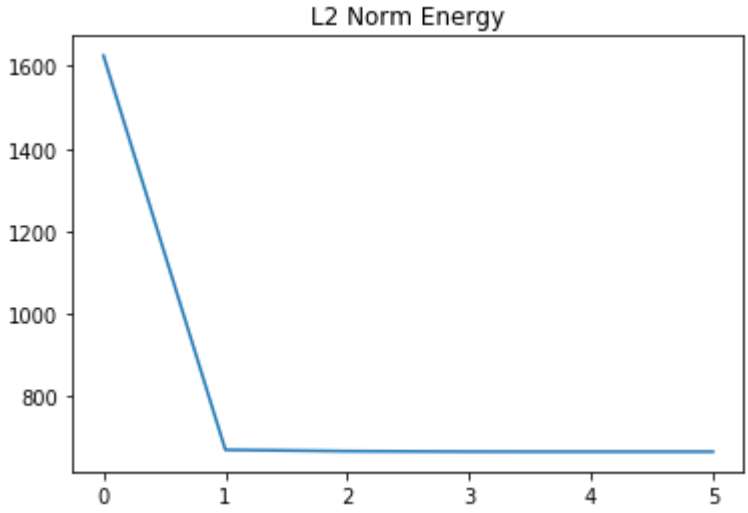
In [159]:

```
clustering(100, 100, 3, 0)
clustering(100, 100, 3, 1)
```


L1 Norm Plot
0
20
40
60
80
0
20
40
60
80


L1 Norm Energy
47.5
45.0
42.5
40.0
37.5
35.0
32.5
30.0
0
1
2
3
4
5

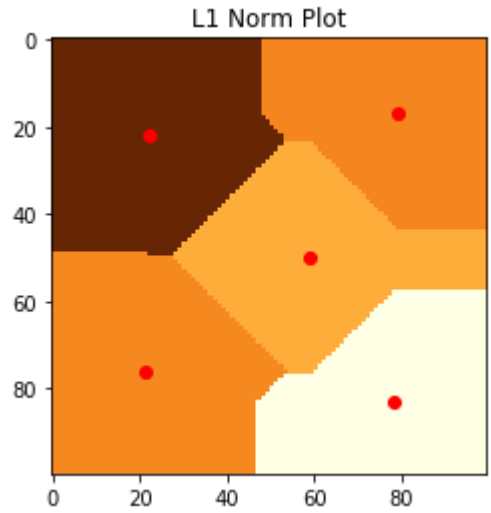

L2 Norm Plot
0
20
40
60
80
0
20
40
60
80

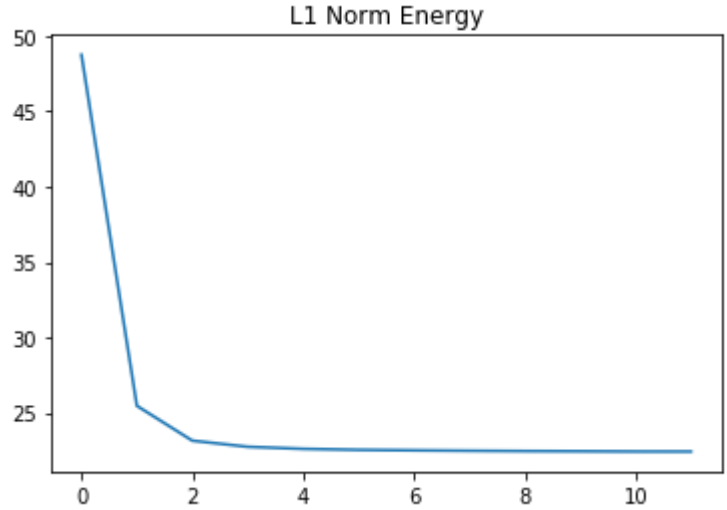


2) k is 5

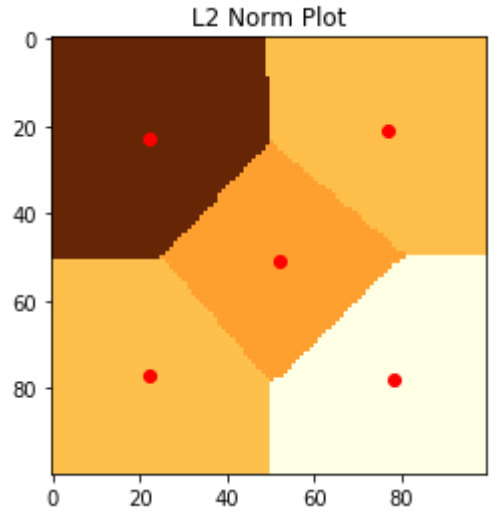
In [162]:

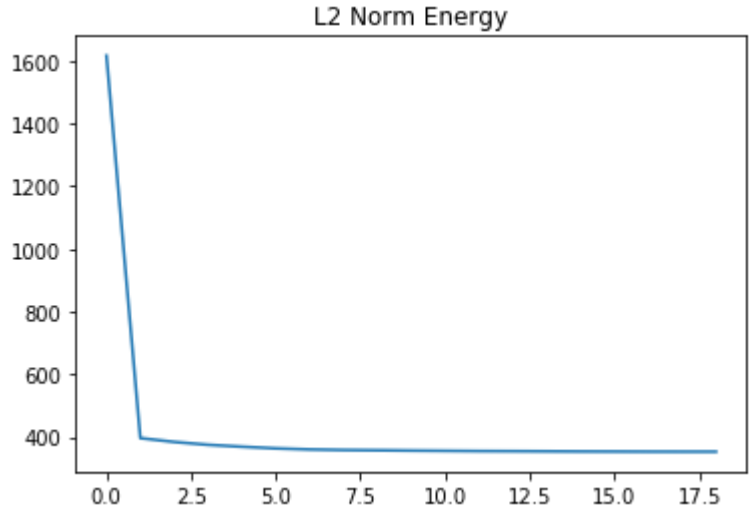
```
clustering(100, 100, 5, 0)
clustering(100, 100, 5, 1)
```






```
C:₩ProgramData₩Anaconda3₩lib₩site-packages₩ipykernel_launcher.py:124: RuntimeWarni
ng: invalid value encountered in double_scalars
C:₩ProgramData₩Anaconda3₩lib₩site-packages₩ipykernel_launcher.py:125: RuntimeWarni
ng: invalid value encountered in double_scalars
```

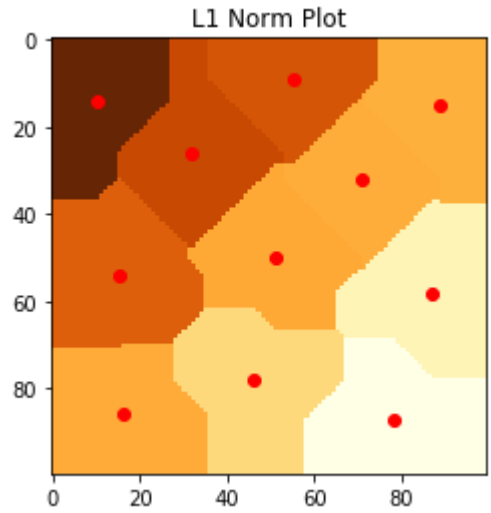


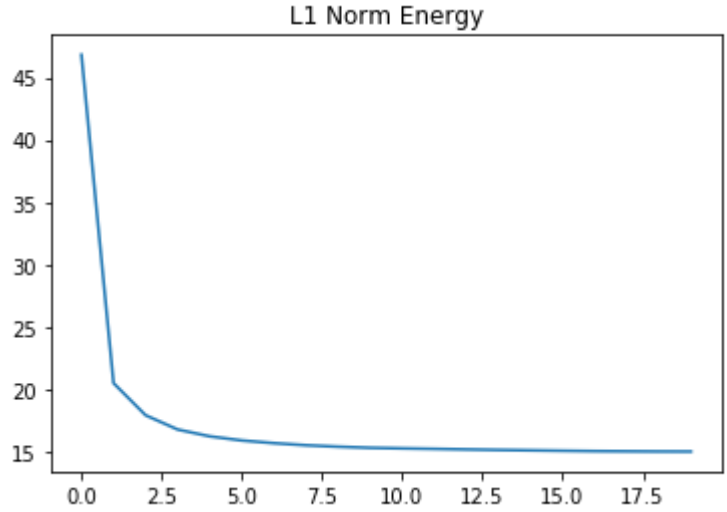

L2 Norm Energy
1600
1400
1200
1000
800
600
400
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

3) k is 11

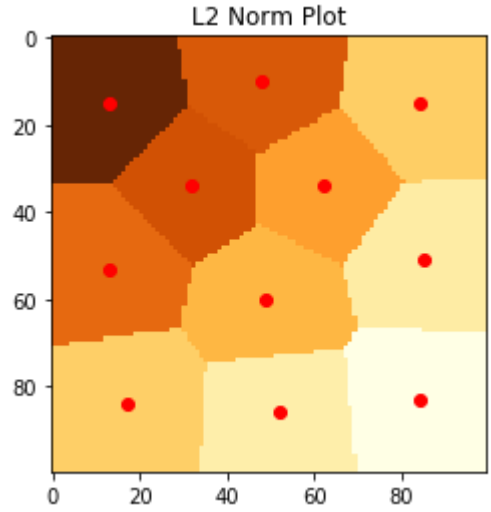
In [161]:

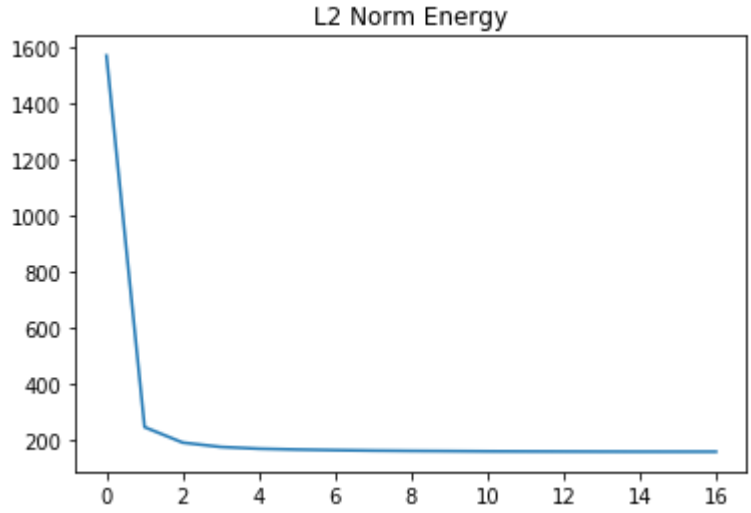
```
clustering(100, 100, 11, 0)
clustering(100, 100, 11, 1)
```






```
C:₩ProgramData₩Anaconda3₩lib₩site-packages₩ipykernel_launcher.py:124: RuntimeWarni
ng: invalid value encountered in double_scalars
C:₩ProgramData₩Anaconda3₩lib₩site-packages₩ipykernel_launcher.py:125: RuntimeWarni
ng: invalid value encountered in double_scalars
```

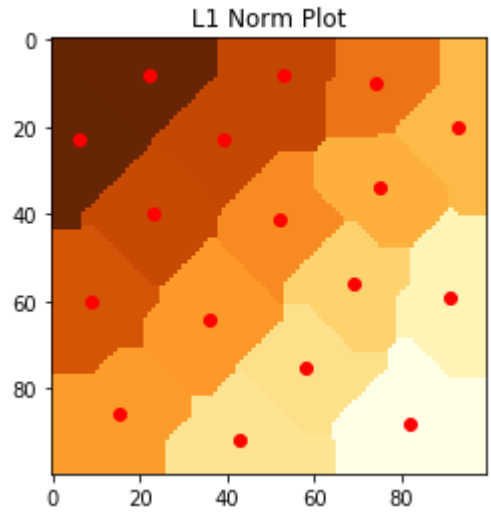


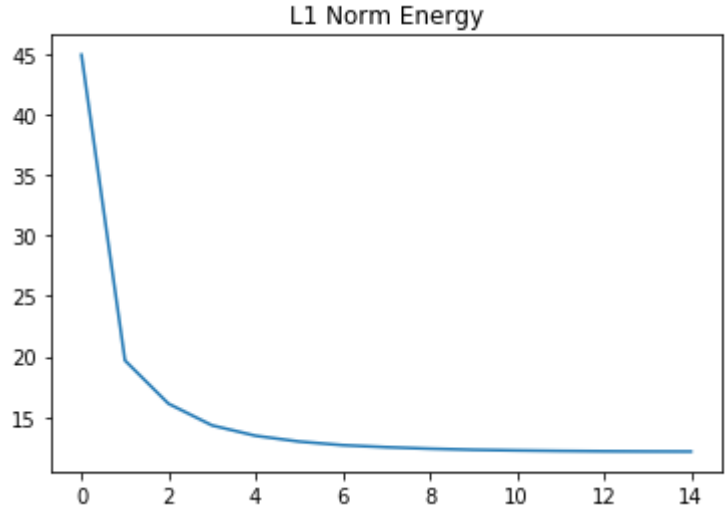

L2 Norm Energy
1600
1400
1200
1000
800
600
400
200
0
2
4
6
8
10
12
14
16

4) k is 17

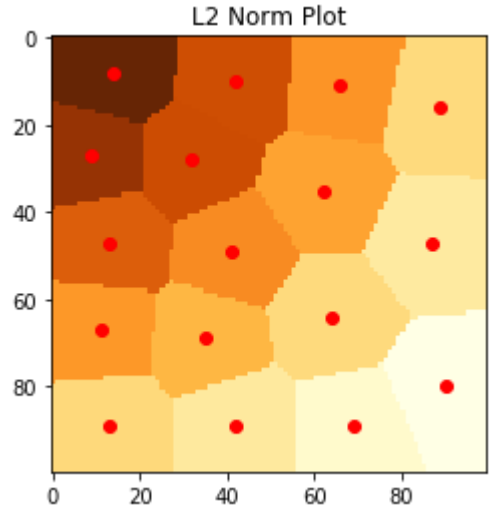
In [163]:

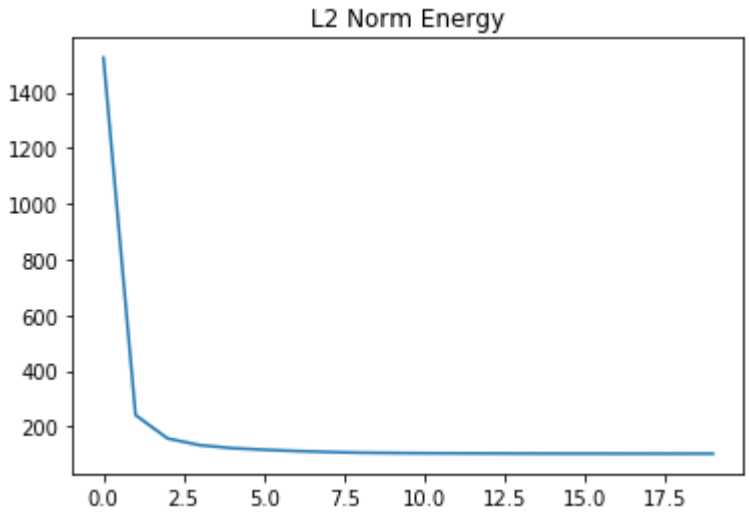
```
clustering(100, 100, 17, 0)
clustering(100, 100, 17, 1)
```






```
C:₩ProgramData₩Anaconda3₩lib₩site-packages₩ipykernel_launcher.py:124: RuntimeWarni
ng: invalid value encountered in double_scalars
C:₩ProgramData₩Anaconda3₩lib₩site-packages₩ipykernel_launcher.py:125: RuntimeWarni
ng: invalid value encountered in double_scalars
```




L2 Norm Energy
1400
1200
1000
800
600
400
200
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5