Sheet 3_Clustering

March 15, 2019

1 Sheet#3 Normalized Cut and Kmeans Clustering

1.1 Question One on Kmeans Clustering

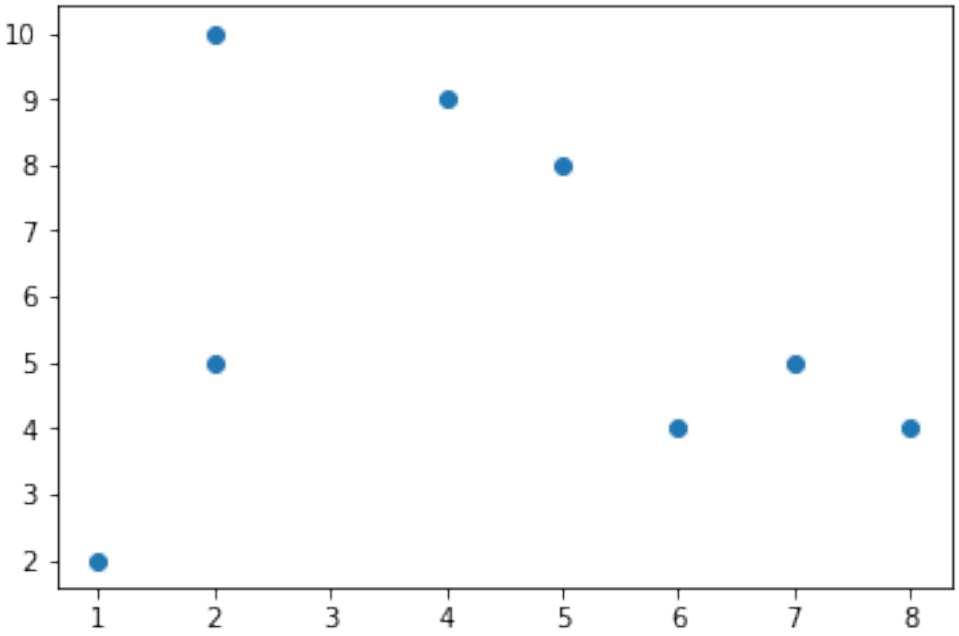
Use the k-means algorithm and Euclidean distance to cluster the following 8 examples into 3 clusters.

Data Set Input:

Point	x	y
A1	2	10
A2	2	5
A3	8	4
A4	5	8
A5	7	5
A6	6	4
A7	1	2
A8	4	9

In [1]:
```
#imports cell
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from copy import deepcopy
from sklearn.metrics.pairwise import rbf_kernel as rbf
from sklearn.neighbors import NearestNeighbors as nn
from scipy.sparse.linalg import eigs as sciEigs
```

In [2]:
```
#Loading and visulaizing the dataset.
dataSet = np.array([[2,10],[2,5],[8,4],[5,8],[7,5],[6,4],[1,2],[4,9]])
plt.scatter(dataSet[:,0],dataSet[:,1]);
```




```
In [3]: #to compute distance between the 2 points.
        def Distance(x,y):
            dist = np.sum(np.abs(x-y)**2,axis=-1)**(1./2)
            return dist
```

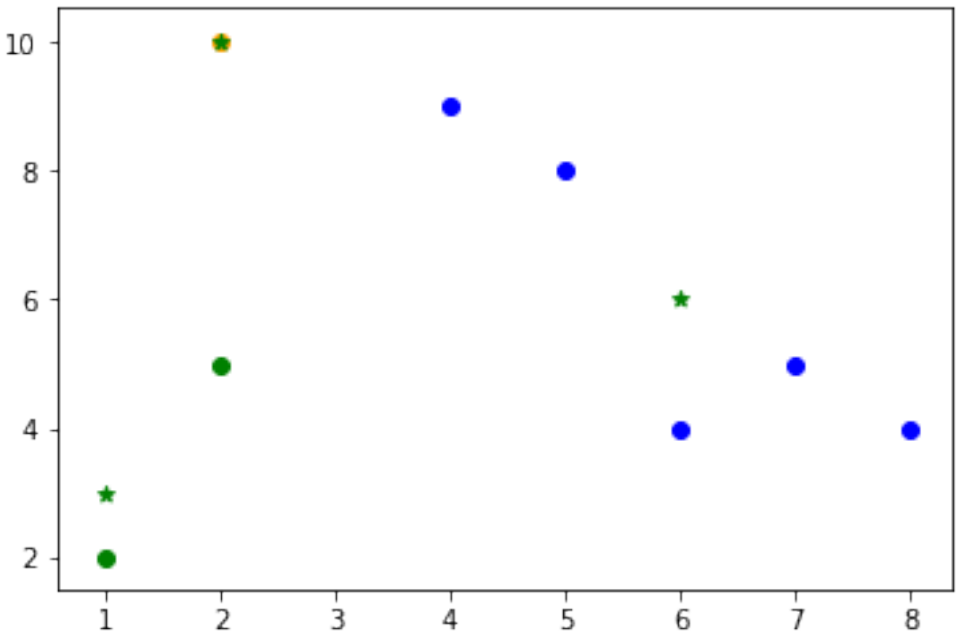
Suppose that the initial seeds (centers of each cluster) are A1, A4 and A7. Run the k-means algorithm for 1 epoch only.

```
In [4]: def kmeans(data, num_clusters = 2, tolerance=0.0001, max_iter = 300, init_seed = None)
            iter_num = 0
            # Number of training data
            n = data.shape[0]
            # Number of features in the data
            c = data.shape[1]
            # Generate random centers, here i use standard devation
            #and mean to ensure it represents the whole data
            if(init_seed is None):
                mean = np.mean(data, axis = 0)
                std = np.std(data, axis = 0)
                centroids = np.random.randn(num_clusters,c)*std + mean
            else:
                centroids = init_seed

            # to store old centers
            old_centroids = np.zeros(centroids.shape)
            # Store new centers
```

```
            new_centroids = deepcopy(centroids)
            #generate error vector
            error = np.linalg.norm(new_centroids - old_centroids)
            #create clusters array
            clusters = np.zeros(n)
            #create distaces array
            distances = np.zeros((n,num_clusters))
            # When, after an update, the estimate of that center stays the same, exit loop
            while error > tolerance and iter_num < max_iter:
                iter_num +=1
                # Measure the distance to every center
                for i in range(num_clusters):
                    distances[:,i] = np.linalg.norm(data - new_centroids[i], axis=1)
                # Assign all training data to closest center
                clusters = np.argmin(distances, axis = 1)
                old_centroids = deepcopy(new_centroids)
                # Calculate mean for every cluster and update the center
                for i in range(num_clusters):
                    new_centroids[i] = np.mean(data[clusters == i], axis=0)
                error = np.linalg.norm(new_centroids - old_centroids)

            return new_centroids,clusters,error,iter_num
```

Epoch 1

```
In [5]: #running the algorithm for one epoch only and for initial seed given.
        centroids, clusters, error,numofiter = kmeans(dataSet,num_clusters=3,max_iter=1,
                                                      init_seed = np.array([dataSet[0],dataSet
        print("Centroids are : ",centroids)
        print("Points assigned to Cluster : ",clusters)
        print("Error : ",error)

Centroids are :  [[ 2 10]
 [ 6  6]
 [ 1  3]]
Points assigned to Cluster :  [0 2 1 1 1 1 2 1]
Error :  2.449489742783178


In [6]: # Plot the data and the centers generated as random
        colors=['orange', 'blue', 'green']
        for i in range(dataSet.shape[0]):
            plt.scatter(dataSet[i, 0], dataSet[i,1],color = colors[clusters[i]])
        plt.scatter(centroids[:,0], centroids[:,1], marker='*', c='g')

Out[6]: <matplotlib.collections.PathCollection at 0x7fd2d3285978>
```

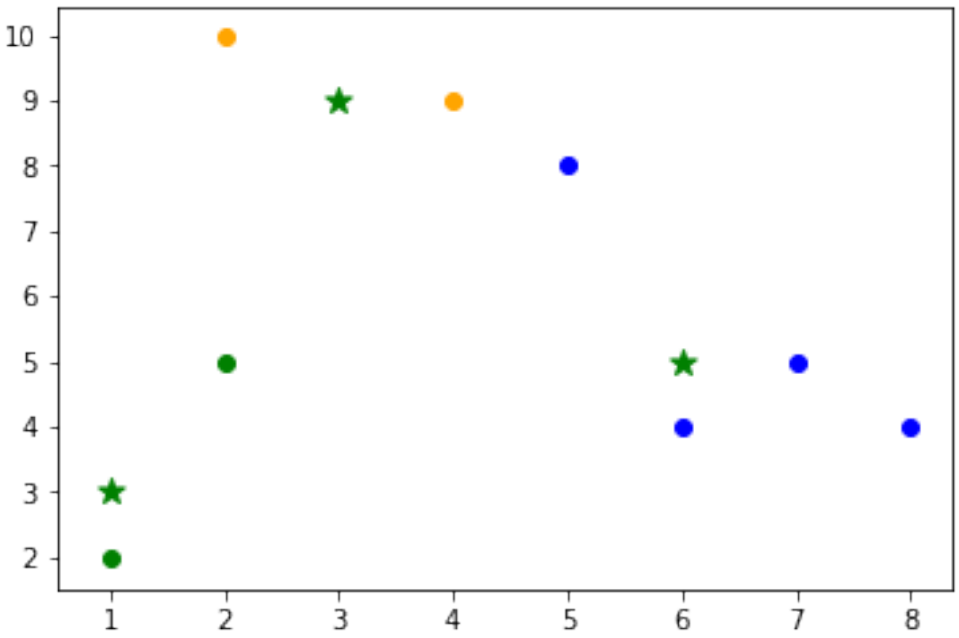


After 1 epoch : *New Clusters:* Cluster 1, Orange [2,10] Cluster 2, Blue [8,4],[5,8],[7,5],[6,4],[4,9] Cluster 3, Green [1,2],[2,5] *New Centers:* Center 1 : [2,10] Center 2 : [6,6] Center 3 : [1,3]

d. How many more iterations are needed to converge? Draw the result for each epoch.

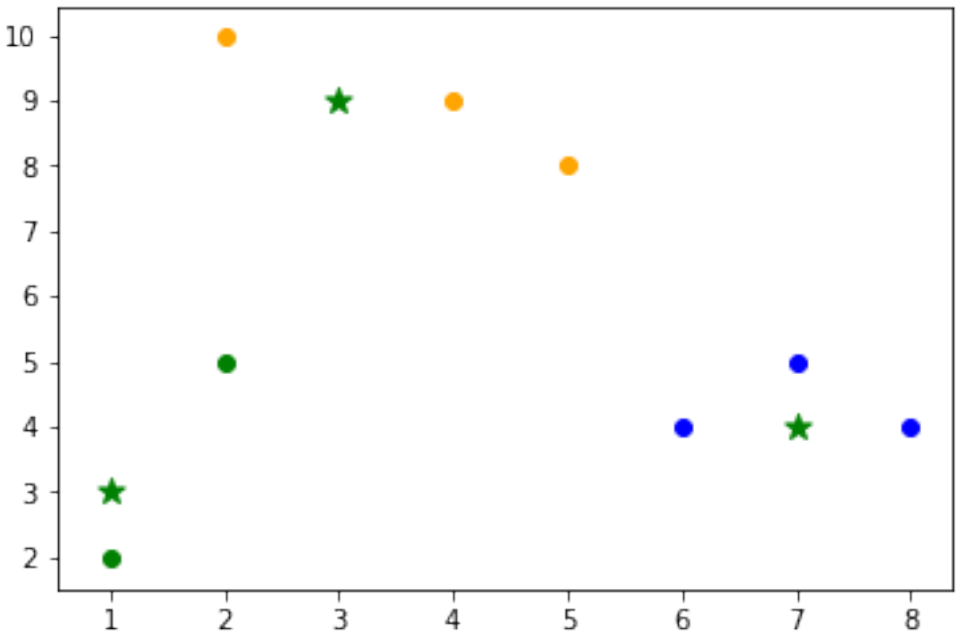
Epoch 2

```
In [7]: #running the algorithm for two epoch only and for initial seed given.
        centroids, clusters, error,numofiter = kmeans(dataSet,num_clusters=3,max_iter=2,
                                              init_seed = np.array([dataSet[0],dataSet[3],dataSet[6]]
        print("Centroids are : ",centroids)
        print("Points assigned to Cluster : ",clusters)
        print("Error : ",error)

Centroids are :  [[3 9]
 [6 5]
 [1 3]]
Points assigned to Cluster :  [0 2 1 1 1 1 2 0]
Error :  1.7320508075688772


In [8]: # Plot the data and the centers generated as random
        colors=['orange', 'blue', 'green']
        for i in range(dataSet.shape[0]):
            plt.scatter(dataSet[i, 0], dataSet[i,1],color = colors[clusters[i]])
        plt.scatter(centroids[:,0], centroids[:,1], marker='*', c='g',s=100)

Out[8]: <matplotlib.collections.PathCollection at 0x7fd2d31ff898>
```



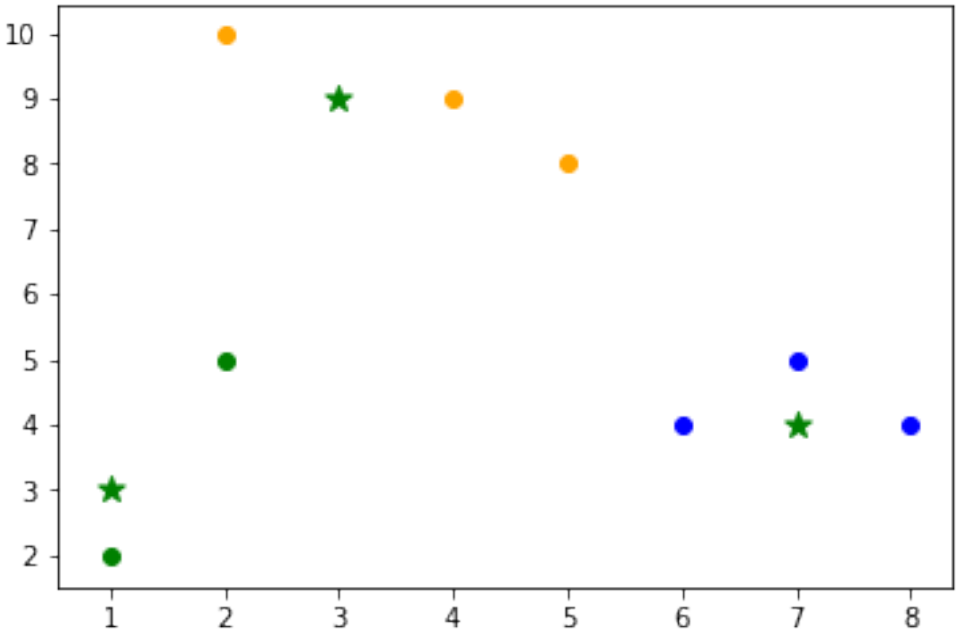
Epoch 3

```
In [9]: #running the algorithm for three epoch only and for initial seed given.
        centroids, clusters, error,numofiter = kmeans(dataSet,num_clusters=3,max_iter=3,
                                                      init_seed = np.array([dataSet[0],dataSet
        print("Centroids are : ",centroids)
        print("Points assigned to Cluster : ",clusters)
        print("Error : ",error)

Centroids are :  [[3 9]
 [7 4]
 [1 3]]
Points assigned to Cluster :  [0 2 1 0 1 1 2 0]
Error :  1.4142135623730951


In [10]: # Plot the data and the centers generated as random
         colors=['orange', 'blue', 'green']
         for i in range(dataSet.shape[0]):
             plt.scatter(dataSet[i, 0], dataSet[i,1],color = colors[clusters[i]])
         plt.scatter(centroids[:,0], centroids[:,1], marker='*', c='g',s=100)

Out[10]: <matplotlib.collections.PathCollection at 0x7fd2d3636b70>
```



Epoch 4

```
In [11]: #running the algorithm for four epoch only and for initial seed given.
         centroids, clusters, error,numofiter = kmeans(dataSet,num_clusters=3,max_iter=4,
                                                       init_seed = np.array([dataSet[0],dataSe
         print("Centroids are : ",centroids)
         print("Points assigned to Cluster : ",clusters)
         print("Error : ",error)

Centroids are :  [[3 9]
 [7 4]
 [1 3]]
Points assigned to Cluster :  [0 2 1 0 1 1 2 0]
Error :  0.0


In [12]: # Plot the data and the centers generated as random
         colors=['orange', 'blue', 'green']
         for i in range(dataSet.shape[0]):
             plt.scatter(dataSet[i, 0], dataSet[i,1],color = colors[clusters[i]])
         plt.scatter(centroids[:,0], centroids[:,1], marker='*', c='g',s=100)

Out[12]: <matplotlib.collections.PathCollection at 0x7fd2d318e390>
```



The algorithm took 3 iterations or epochs to converge since the fourth iteration doesn't add any information to the clusters and the centroids and clusters stay the same for the rest of any number of itertaions.

1.2 Question 2 on Normalized Cut

Given the graph below. The weight on each edge is the affinity between two nodes. Consider the two cuts C1 and C2 and in the graph. For each cut, compute the values of the graph cut and the normalized cut.

For Cut C1 :

1- Graph Cut(A,B) = $\sum Wij$ = [10+10+20] = 40

2- Normalized Cut(A,B) = cut(A,B)($\frac{1}{vol(A)} + \frac{1}{vol(B)}$) = 40(1/70 + 1/185) = 0.7876

For Cut C1 :

1- Graph Cut(A,B) = $\sum Wij$ = [10+10+10+20] = 50

2- Normalized Cut(A,B) = cut(A,B)($\frac{1}{vol(A)} + \frac{1}{vol(B)}$) = 50(1/140 + 1/125) = 0.7571

Which cut will be favored by each algorithm?What is your explanation?

Ans: As explained above Graph cut or Min cut will prefer the C1 cut while Normalized Cut will prefer the cut C2. It is intuitively clear that C2 is the better cut.The minimum cut criteria occasionally supports cutting isolated nodes in the graph due to the small values achieved by partitioning such nodes. Advantage of Normalized Cut is being an unbiased measure, the Ncut value with respect to the isolated nodes will be of a large percentage compared to the total connection from small set to all other nodes.

1.3 Question 3 on Normalized Cut

Write your python code to implement K ways normalized cut k=3

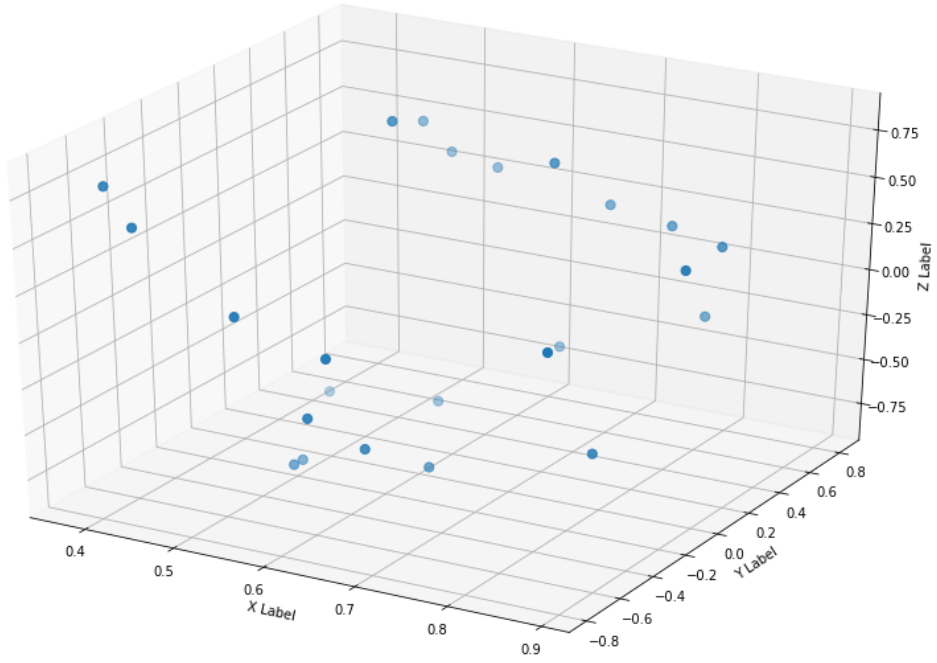
```
In [13]: def SpectralClustering(data, num_clusters=2, affinity='rbf', gamma=1.0, num_neighbors=
             if(affinity == 'rbf'):
                 sim_matrix = rbf(data,data,gamma)
             elif(affinity == 'knn'):
                 nearest_neigbhor = nn(n_neighbors=num_neighbors)
                 nearest_neigbhor.fit(data)
                 sim_matrix = nearest_neigbhor.kneighbors_graph(data, mode='connectivity').toa

             deg_matrix = np.diag(np.sum(sim_matrix, axis=1))
             laplace_matrix = deg_matrix - sim_matrix
             asym_laplace_matrix = np.dot(np.linalg.inv(deg_matrix),laplace_matrix)
             eig_values,eig_vectors = np.linalg.eig(asym_laplace_matrix)
             idx = np.real(eig_values).argsort()[:num_clusters]
             eig_vectors = np.real(eig_vectors[:,idx])
             rows_norm = np.linalg.norm(eig_vectors, axis=1)
             normalized_eig_vectors = (eig_vectors.T / rows_norm).T
             return normalized_eig_vectors,np.real(eig_values)

In [14]: #load data on 2D graph onto 2D array for usage.
         clustering_dataSet = np.array([[2,4],[3,3],[3,4],[5,4],[5,6],[5,8],[6,4],[6,5],[6,7],

In [15]: Ymatrices = []
         for gammaVal in [0.01,0.1,1.0,10.0]:
             y_matrix,values = SpectralClustering(clustering_dataSet, num_clusters=3, affinity=
             Ymatrices.append(y_matrix)
             fig = plt.figure();
             plt.gcf().set_size_inches(15,10);
             ax = fig.add_subplot(111, projection='3d');
             ax.scatter(y_matrix[:,0],y_matrix[:,1],y_matrix[:,2], s=50);
             ax.set_xlabel('X Label');
             ax.set_ylabel('Y Label');
             ax.set_zlabel('Z Label');
             values = values[values.argsort()[:3]]
             print("\nFor Gamma = ", gammaVal)
             print("\nEigen Vectors = \n",y_matrix,"\nEigen Values = \n",values)
```

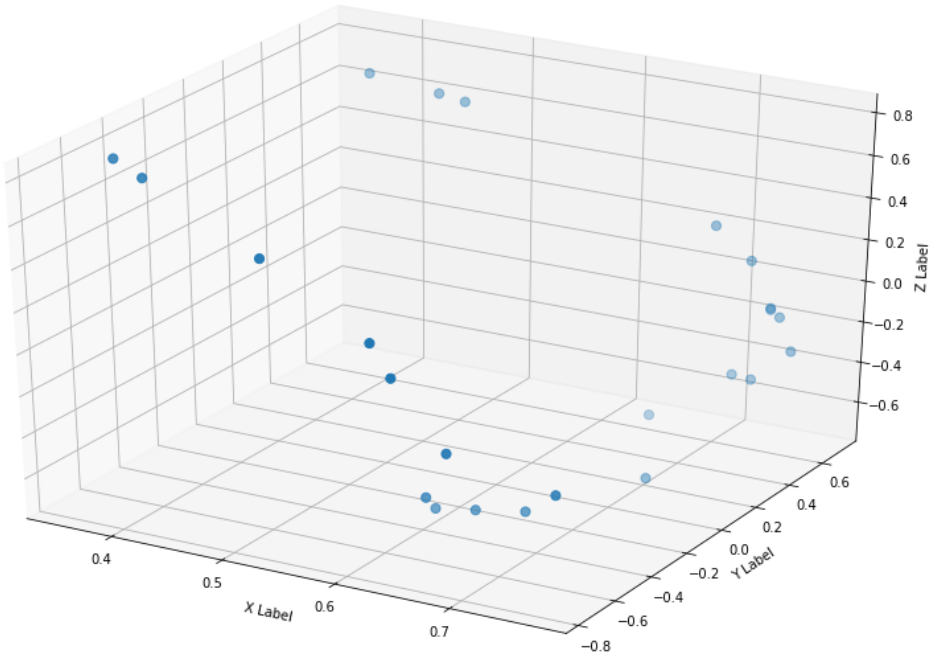
```
For Gamma =  0.01

Eigen Vectors =
 [[ 0.48469361  0.81568545  0.31579956]
 [ 0.4717949   0.71134742  0.52095529]
 [ 0.54172725  0.78895113  0.28997879]
 [ 0.68814602  0.67457621  0.2672115 ]
 [ 0.65288784  0.57097837 -0.49771595]
```

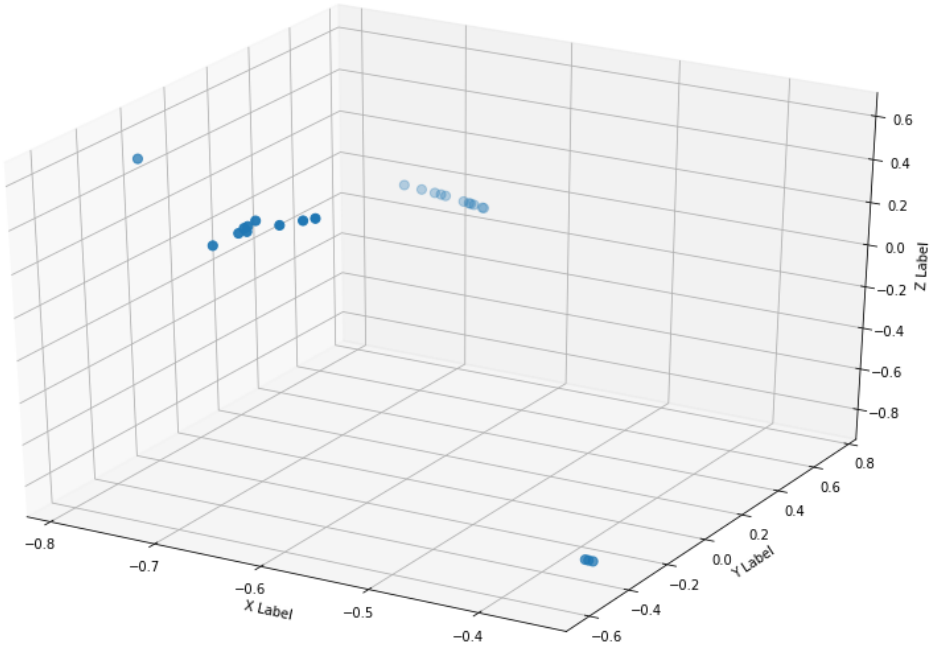
```
 [ 0.43340636  0.33331067 -0.837295  ]
 [ 0.77433842  0.56698455  0.28094224]
 [ 0.81537465  0.55389973 -0.16840212]
 [ 0.56117167  0.32210643 -0.76245249]
 [ 0.68145218  0.36221159  0.63594473]
 [ 0.85497805  0.40930615  0.31856085]
 [ 0.53904442  0.17563858  0.82376101]
 [ 0.89159882 -0.03556095  0.45142769]
 [ 0.85143498 -0.34106092 -0.39841677]
 [ 0.66744578 -0.29927091 -0.68187466]
 [ 0.51775544 -0.25641679 -0.81619834]
 [ 0.83253914 -0.51028536  0.21560945]
 [ 0.53690009 -0.40265479 -0.74135512]
 [ 0.66158878 -0.63519584 -0.39853047]
 [ 0.64383592 -0.75386197  0.13102457]
 [ 0.62305035 -0.75475832 -0.20530013]
 [ 0.55227955 -0.78449543  0.28205357]
 [ 0.3745975  -0.59630214  0.71000033]
 [ 0.4245056  -0.69143167  0.58456586]]
Eigen Values =
 [-1.11022302e-16  7.39964832e-01  9.47184487e-01]

For Gamma =  0.1

Eigen Vectors =
 [[ 0.40031141  0.5173857   0.75634834]
 [ 0.45533177  0.56413678  0.68878347]
 [ 0.47591177  0.58126518  0.66002938]
 [ 0.68246202  0.70169848  0.204609  ]
 [ 0.71562681  0.69499301  0.06973511]
 [ 0.73900598  0.66135196 -0.12838903]
 [ 0.73873002  0.66285507 -0.12207014]
 [ 0.74934365  0.64547547 -0.1478023 ]
 [ 0.76941218  0.58243265 -0.26225387]
 [ 0.72239789  0.5562932  -0.41070568]
 [ 0.74239186  0.53437838 -0.40409662]
 [ 0.66854009  0.4325236  -0.60496072]
 [ 0.71688269  0.10274542 -0.68958145]
 [ 0.68910383 -0.39301606 -0.60883026]
 [ 0.65425163 -0.44698669 -0.6100473 ]
 [ 0.62362834 -0.47389011 -0.62170399]
 [ 0.74028064 -0.55344323 -0.38168725]
 [ 0.63642035 -0.60195561 -0.48230548]
 [ 0.67359246 -0.72033054 -0.16552075]
 [ 0.6112519  -0.73561597  0.29195935]
 [ 0.63259202 -0.75718754  0.16277086]
 [ 0.50658686 -0.67065745  0.54183794]
 [ 0.35275167 -0.51747996  0.77960294]
```

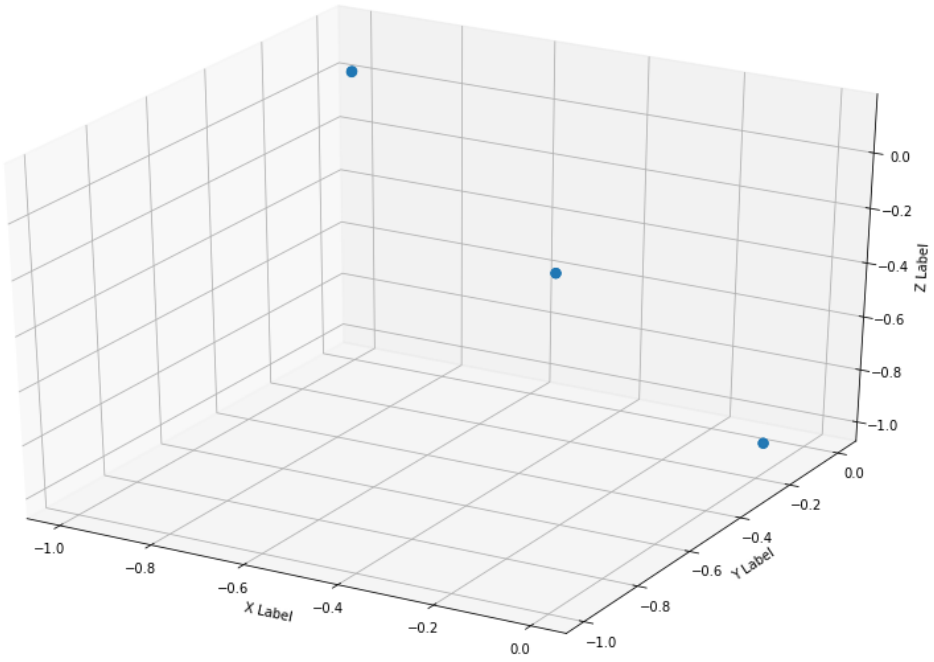
```
 [ 0.38476853 -0.55477203  0.73768636]]
Eigen Values =
 [1.11022302e-16 1.21461656e-01 4.48874537e-01]

For Gamma =  1.0

Eigen Vectors =
 [[-0.35344517 -0.40369681 -0.84386338]
 [-0.35694113 -0.40694822 -0.8408247 ]
 [-0.35945294 -0.4092811  -0.83861944]
 [-0.65504372 -0.61599396  0.4375719 ]
 [-0.6025986  -0.57363681  0.55481145]
 [-0.5739635  -0.55547233  0.60167797]
 [-0.63655366 -0.59020567  0.49644407]
 [-0.62943585 -0.58744421  0.50864508]
 [-0.5838231  -0.56174516  0.58616804]
 [-0.63160623 -0.57228487  0.52303308]
 [-0.63410345 -0.5770356   0.51472588]
 [-0.62745018 -0.55719363  0.54391317]
 [-0.79081458 -0.28402759  0.54216292]
 [-0.71570694  0.6981785  -0.01761682]
 [-0.70489923  0.70868727 -0.02965535]
 [-0.69984874  0.71341252 -0.03541638]
 [-0.729799    0.68366002 -0.00154661]
 [-0.69600196  0.71690183 -0.04041073]
 [-0.68095357  0.72981295 -0.06062424]
 [-0.67491361  0.73469436 -0.0686718 ]
 [-0.67643217  0.73349088 -0.06656315]
 [-0.67221375  0.73681224 -0.07236297]
 [-0.66388181  0.7430864  -0.0841638 ]
 [-0.66512917  0.74216837 -0.0823972 ]]
Eigen Values =
 [1.99376585e-16 9.40049236e-04 6.89518032e-03]

For Gamma =  10.0

Eigen Vectors =
 [[-3.50941661e-04 -9.91861208e-01  1.27323297e-01]
 [-3.50941661e-04 -9.91861208e-01  1.27323297e-01]
 [-3.50941661e-04 -9.91861208e-01  1.27323297e-01]
 [-2.33445645e-02 -1.27206039e-01 -9.91601561e-01]
 [-2.33445625e-02 -1.27206038e-01 -9.91601561e-01]
 [-2.33445604e-02 -1.27206036e-01 -9.91601561e-01]
 [-2.33445645e-02 -1.27206039e-01 -9.91601561e-01]
 [-2.33445645e-02 -1.27206039e-01 -9.91601561e-01]
 [-2.33445611e-02 -1.27206037e-01 -9.91601561e-01]
 [-2.33445645e-02 -1.27206039e-01 -9.91601561e-01]
 [-2.33445645e-02 -1.27206039e-01 -9.91601561e-01]
```

```
 [-2.33445638e-02 -1.27206038e-01 -9.91601561e-01]
 [-9.99727079e-01  3.32041920e-03  2.31244936e-02]
 [-9.99722400e-01  1.66103974e-03  2.35024157e-02]
 [-9.99726662e-01  1.89680297e-03  2.33024323e-02]
 [-9.99676692e-01  4.94558588e-03  2.49409693e-02]
 [-9.99727080e-01  3.32047585e-03  2.31244664e-02]
 [-9.99716286e-01  4.93693554e-03  2.33018053e-02]
 [-9.99726290e-01  4.02439702e-03  2.30466918e-02]
 [-9.99666995e-01  3.75286064e-03  2.55306526e-02]
 [-9.99769048e-01  2.96372294e-03  2.12853498e-02]
 [-9.99713352e-01  3.30937585e-03  2.37120566e-02]
 [-9.99727079e-01  3.32048016e-03  2.31244764e-02]
 [-9.99727079e-01  3.32042317e-03  2.31245060e-02]]
Eigen Values =
 [-4.36245829e-20  5.42457998e-17  6.07190939e-17]
```



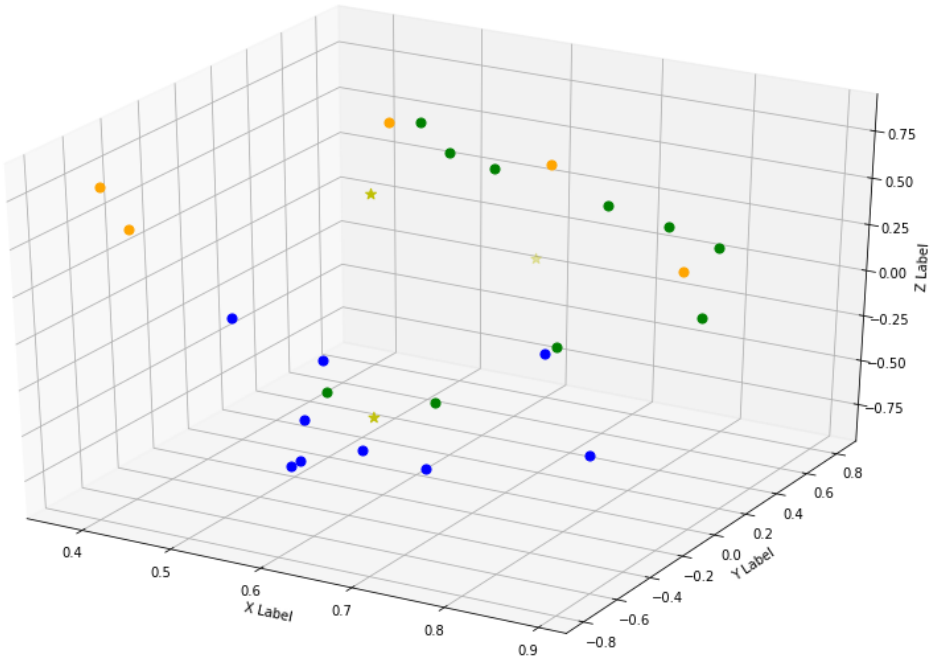

X Label
Y Label
Z Label


X Label
Y Label
Z Label



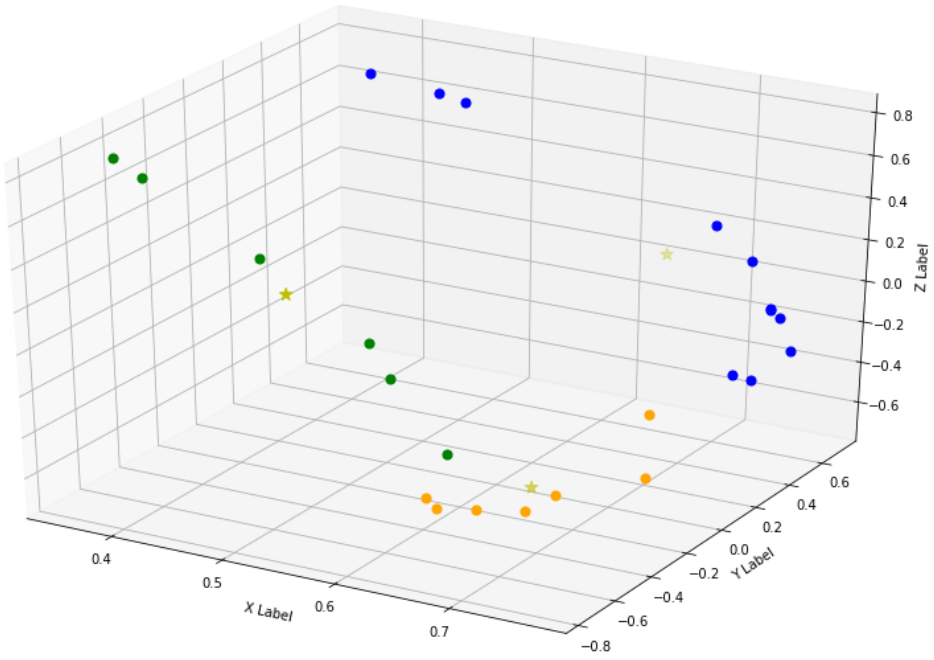
1.3.1 For $\gamma = 0.01$

```
In [16]: y_matrix = Ymatrices[0]
         centroids, clusters, error,numofiter = kmeans(y_matrix, num_clusters=3)
         # Plot the data and the centers generated as random
         colors=['orange', 'blue', 'green']
         fig = plt.figure();
         plt.gcf().set_size_inches(15,10);
         ax = fig.add_subplot(111, projection='3d');
         for i in range(y_matrix.shape[0]):
             ax.scatter(y_matrix[i,0],y_matrix[i,1],y_matrix[i,2], s=50, color = colors[cluste
         ax.scatter(centroids[:,0], centroids[:,1],centroids[:,2], marker='*', c='y',s=70)
         ax.set_xlabel('X Label');
         ax.set_ylabel('Y Label');
         ax.set_zlabel('Z Label');
```



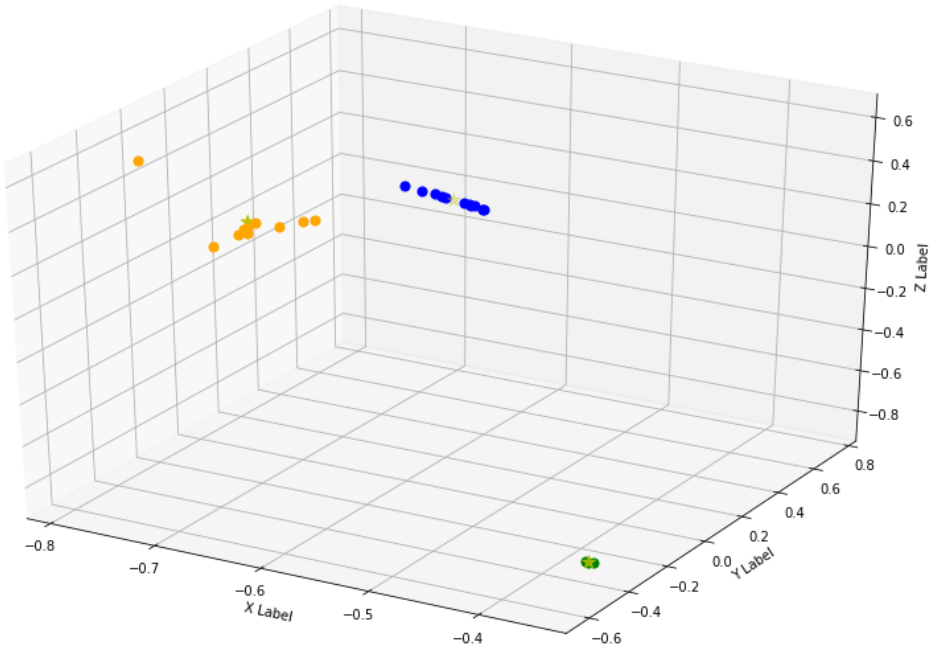
1.3.2 For $\gamma = 0.1$

```
In [17]: y_matrix = Ymatrices[1]
         centroids, clusters, error,numofiter = kmeans(y_matrix, num_clusters=3)
         # Plot the data and the centers generated as random
         colors=['orange', 'blue', 'green']
         fig = plt.figure();
         plt.gcf().set_size_inches(15,10);
         ax = fig.add_subplot(111, projection='3d');
         for i in range(y_matrix.shape[0]):
             ax.scatter(y_matrix[i,0],y_matrix[i,1],y_matrix[i,2], s=50, color = colors[cluste

         ax.scatter(centroids[:,0], centroids[:,1],centroids[:,2], marker='*', c='y',s=100)
         ax.set_xlabel('X Label');
         ax.set_ylabel('Y Label');
         ax.set_zlabel('Z Label');
```



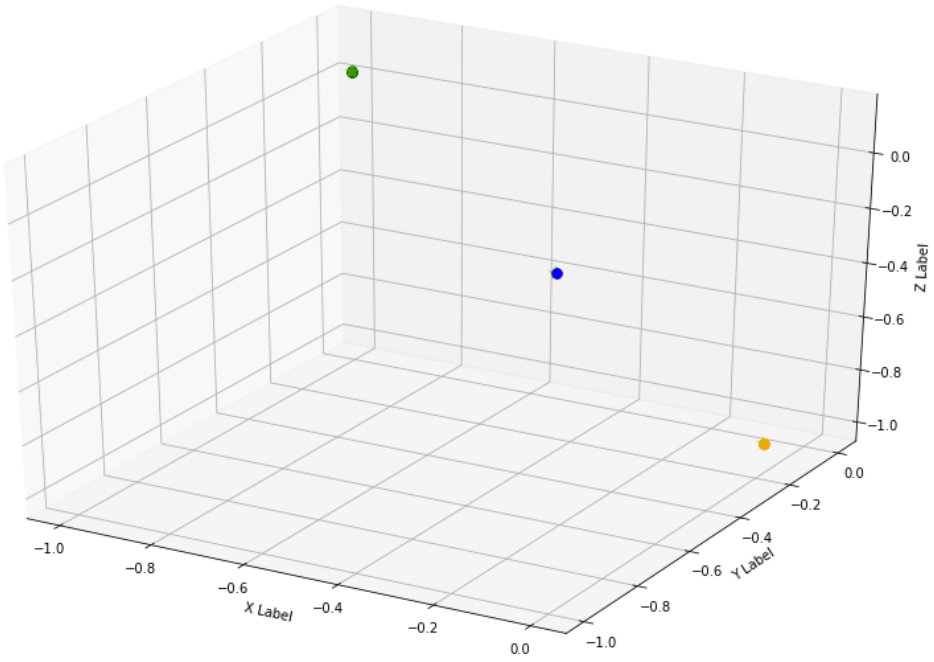
1.3.3 For $\gamma = 1.0$

```
In [18]: y_matrix = Ymatrices[2]
         centroids, clusters, error,numofiter = kmeans(y_matrix, num_clusters=3)
         # Plot the data and the centers generated as random
         colors=['orange', 'blue', 'green']
         fig = plt.figure();
         plt.gcf().set_size_inches(15,10);
         ax = fig.add_subplot(111, projection='3d');
         for i in range(y_matrix.shape[0]):
             ax.scatter(y_matrix[i,0],y_matrix[i,1],y_matrix[i,2], s=50, color = colors[cluste

         ax.scatter(centroids[:,0], centroids[:,1],centroids[:,2], marker='*', c='y',s=100)
         ax.set_xlabel('X Label');
         ax.set_ylabel('Y Label');
         ax.set_zlabel('Z Label');
```



1.3.4 For $\gamma = 10.0$

```
In [19]: y_matrix = Ymatrices[3]
         centroids, clusters, error,numofiter = kmeans(y_matrix, num_clusters=3)
         # Plot the data and the centers generated as random
         colors=['orange', 'blue', 'green']
         fig = plt.figure();
         plt.gcf().set_size_inches(15,10);
         ax = fig.add_subplot(111, projection='3d');
         for i in range(y_matrix.shape[0]):
             ax.scatter(y_matrix[i,0],y_matrix[i,1],y_matrix[i,2], s=50, color = colors[cluste

         ax.scatter(centroids[:,0], centroids[:,1],centroids[:,2], marker='*', c='y',s=100)
         ax.set_xlabel('X Label');
         ax.set_ylabel('Y Label');
         ax.set_zlabel('Z Label');
```



1.3.5 Which of these gamma values produces a connected graph?

Ans: By Looking into the eigenvalues, we can see that they all have approximatly zero in the smallest eigen value. Since only for $\gamma = 10$ the 3 smallest values are almost zero so we can say that is not a connected graph and consists of three connected componenets in the graph and for rest of γ values we can say they are conencted graphs.

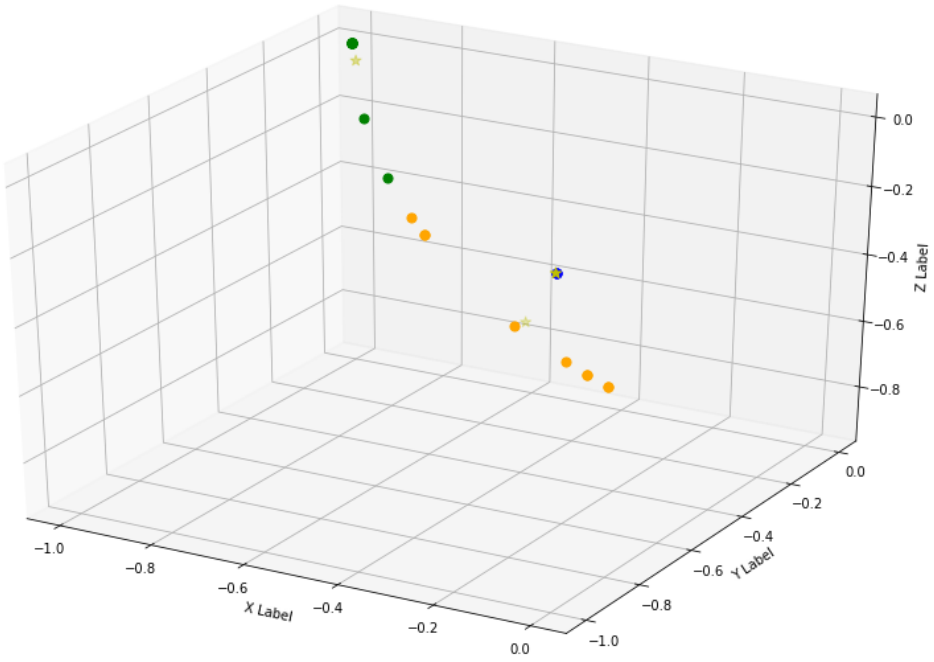
1.3.6 Using KNN to produce simialrity graph instead of RBF Kernel.

```
In [20]: y_matrix,values = SpectralClustering(clustering_dataSet, num_clusters=3, affinity='kn
         values = values[values.argsort()[:3]]
         print("Eigen Vectors = \n",y_matrix,"\nEigen Values = \n",values)
```

```
Eigen Vectors =
 [[ 1.01857327e-19 -1.00000000e+00 -5.52540858e-32]
 [ 1.01857327e-19 -1.00000000e+00 -1.10752529e-47]
 [ 1.01857327e-19 -1.00000000e+00  6.13934287e-32]
 [-1.00000000e+00  0.00000000e+00  6.29113688e-15]
 [-1.00000000e+00  0.00000000e+00  7.82734472e-15]
 [-1.00000000e+00  0.00000000e+00  1.03876911e-14]
 [-1.00000000e+00  0.00000000e+00  6.14483137e-15]
 [-1.00000000e+00  0.00000000e+00  6.36428963e-15]
 [-1.00000000e+00  0.00000000e+00  9.94877460e-15]
 [-1.00000000e+00  0.00000000e+00  7.09581718e-15]
```

```
 [-1.00000000e+00  0.00000000e+00  6.76662978e-15]
 [-1.00000000e+00  0.00000000e+00  7.43323779e-15]
 [-1.00000000e+00  0.00000000e+00  9.61958720e-15]
 [-9.22752146e-01  0.00000000e+00 -3.85393924e-01]
 [-8.71498144e-01  0.00000000e+00 -4.90398802e-01]
 [-8.42314346e-01  0.00000000e+00 -5.38986588e-01]
 [-9.76185315e-01  0.00000000e+00 -2.16938312e-01]
 [-8.42314346e-01  0.00000000e+00 -5.38986588e-01]
 [-6.39367902e-01  0.00000000e+00 -7.68900960e-01]
 [-4.77016271e-01  0.00000000e+00 -8.78894463e-01]
 [-5.25209920e-01  0.00000000e+00 -8.50972702e-01]
 [-4.77016271e-01  0.00000000e+00 -8.78894463e-01]
 [-4.30736872e-01  0.00000000e+00 -9.02477561e-01]
 [-4.30736872e-01  0.00000000e+00 -9.02477561e-01]]
Eigen Values =
 [-2.51302485e-16  0.00000000e+00  4.02049195e-02]
```

In [26]:
```
centroids, clusters, error,numofiter = kmeans(y_matrix, num_clusters=3)
# Plot the data and the centers generated as random
colors=['orange', 'blue', 'green']
fig = plt.figure();
plt.gcf().set_size_inches(15,10);
ax = fig.add_subplot(111, projection='3d');
for i in range(y_matrix.shape[0]):
    ax.scatter(y_matrix[i,0],y_matrix[i,1],y_matrix[i,2], s=50, color = colors[cluste

ax.scatter(centroids[:,0], centroids[:,1],centroids[:,2], marker='*', c='y',s=70)
ax.set_xlabel('X Label');
ax.set_ylabel('Y Label');
ax.set_zlabel('Z Label');
```


X Label
Y Label
Z Label
−1.0
−0.8
−0.6
−0.4
−0.2
0.0