Reyhane Shahrokhian 99521361

HomeWork5 of Computational Intelligence Course

Dr. Mozayeni

Q1:

1-1:

To show the window of the pendulum, It just needed to set the environment and render it(the code was mentioned in the gym library site).

```
import gym

env = gym.make("Pendulum-v1", render_mode="human")

state = env.reset()
observation, info = env.reset(seed=42)
for _ in range(500):
   action = env.action_space.sample()
   observation, reward, terminated, truncated, info = env.step(action)

   if terminated or truncated:
      observation, info = env.reset()

env.close()
```

1-2:

First of all I installed the needed libraries and imported them as follows. Then according to the inputs and the output of the problem, the linguistic variables are defined with steps equal to 0.1 and 0.2:

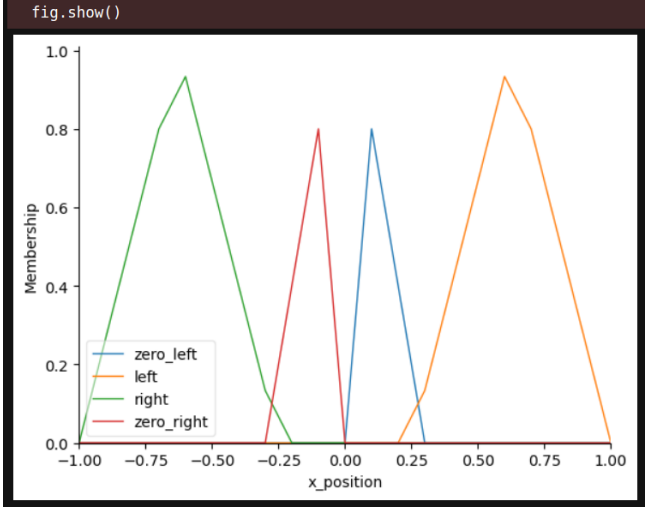
```
# Define the input variables
x_position = ctrl.Antecedent(np.arange(-1, 1.1, 0.1), 'x_position')
y_position = ctrl.Antecedent(np.arange(-1, 1.1, 0.1), 'y_position')
angular_velocity = ctrl.Antecedent(np.arange(-8, 8.1, 0.1), 'angular_velocity')

# Define the output variable
torque = ctrl.Consequent(np.arange(-2, 2.2, 0.2), 'torque')
```

I choose the terms for x and y based on the trigonometric circle in which if y is more than 0 means that the pendulum is up and if it's less than 0 means that it's down. And if x is more or less than zero it shows it in the left or right part(mentioning that in that case it was vice versa). I also define 2 other terms zero_right and zero_left to know if the pendulum is near the vertical axis or not.

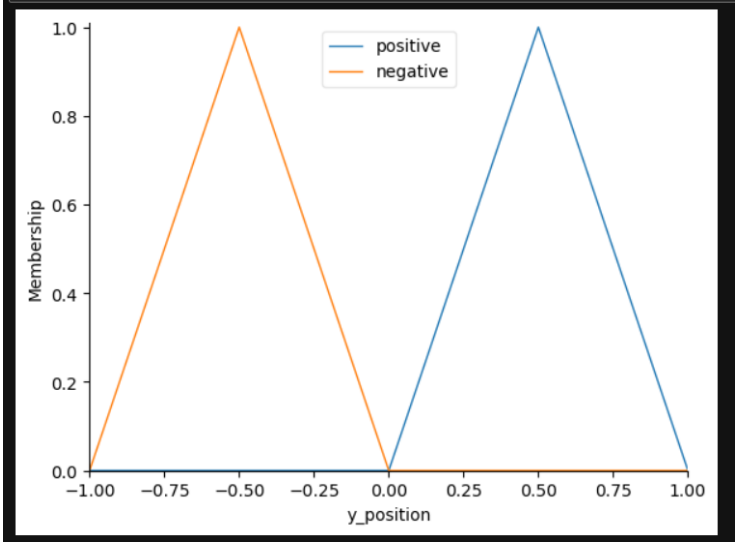
Also about the angular velocity and torque as it was mentioned the positive values mean clockwise and the negative ones are counter clockwise. The angular velocity and torque have 3 speeds: fast, medium and slow in both directions. Beside that the angular velocity has 2 other terms showing just the direction defined as positive and negative. And the torque also has a term named same where changing speeds are not needed.

```
# Define the fuzzy sets for input variables
y_position['positive'] = fuzz.trimf(y_position.universe, [0, 0.5, 1])
y_position['negative'] = fuzz.trimf(y_position.universe, [-1, -0.5, 0])

x_position['zero_left'] = fuzz.trimf(x_position.universe, [0, 0.125, 0.25])
x_position['left'] = fuzz.trimf(x_position.universe, [0.25, 0.625, 1])
x_position['right'] = fuzz.trimf(x_position.universe, [-1, -0.625, -0.25])
x_position['zero_right'] = fuzz.trimf(x_position.universe, [-0.25, -0.125, 0])

angular_velocity['slow_cc'] = fuzz.trimf(angular_velocity.universe, [0, 1, 2])
angular_velocity['medium_cc'] = fuzz.trimf(angular_velocity.universe, [2, 3.5, 5])
angular_velocity['fast_cc'] = fuzz.trimf(angular_velocity.universe, [5, 6.5, 8])

angular_velocity['fast_c'] = fuzz.trimf(angular_velocity.universe, [-8, -6.5, -5])
angular_velocity['medium_c'] = fuzz.trimf(angular_velocity.universe, [-5, -3.5, -2])
angular_velocity['slow_c'] = fuzz.trimf(angular_velocity.universe, [-2, -1, 0])

angular_velocity['negative'] = fuzz.trimf(angular_velocity.universe, [-8, -4, 0])
angular_velocity['positive'] = fuzz.trimf(angular_velocity.universe, [0, 4, 8])

# Define the fuzzy sets for the output variable
torque['slow_cc'] = fuzz.trimf(torque.universe, [0, 0.375, 0.75])
torque['medium_cc'] = fuzz.trimf(torque.universe, [0.75, 1.125, 1.5])
torque['fast_cc'] = fuzz.trimf(torque.universe, [1.5, 1.75, 2])

torque['fast_c'] = fuzz.trimf(torque.universe, [-2, -1.75, -1.5])
torque['medium_c'] = fuzz.trimf(torque.universe, [-1.5, -1.125, -0.75])
torque['slow_c'] = fuzz.trimf(torque.universe, [-0.75, -0.375, 0])

torque['same'] = fuzz.trimf(torque.universe, [-0.1, 0, 0.1])
```

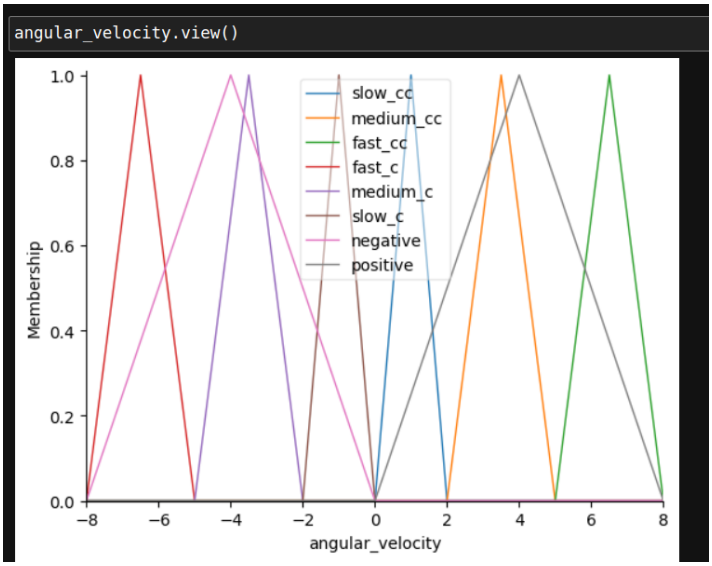
```
x_position.view()
/home/reyhan/anaconda3/lib/python3.9/site-packages/skfuzzy/control/fuzzyvariabl
s currently using module://matplotlib_inline.backend_inline, which is a non-GUI
e.
  fig.show()
```



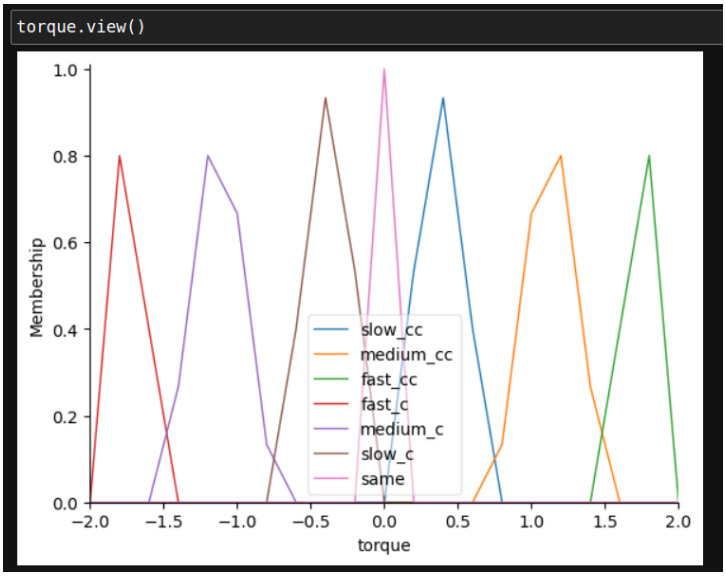

```
y_position.view()
```




```
angular_velocity.view()
```



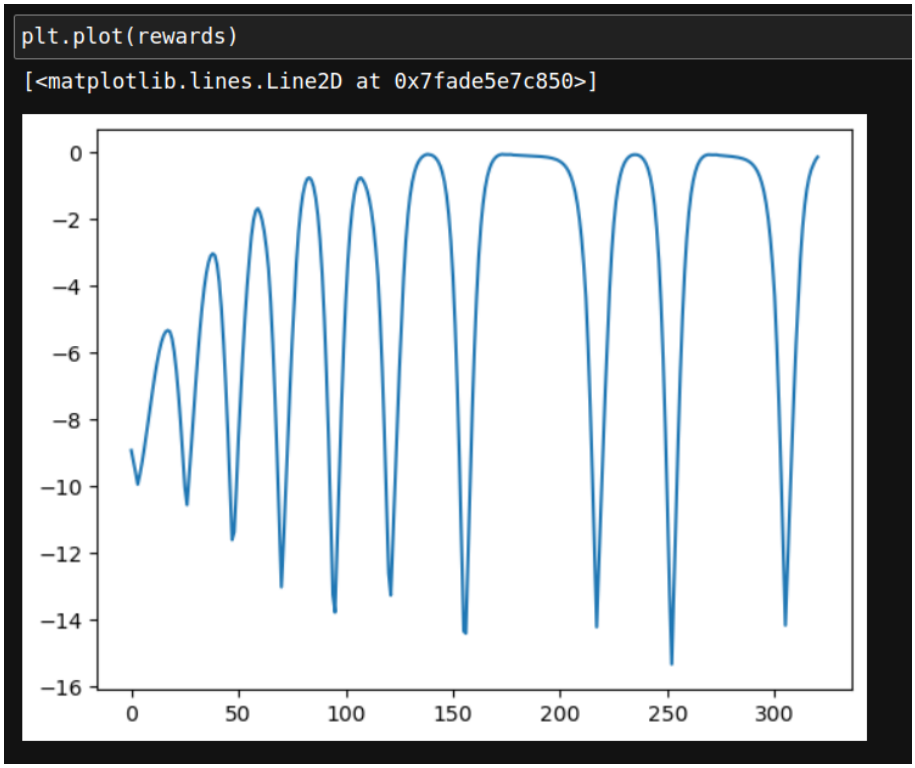

```
torque.view()
```



I then defined the rules based on 4 quarters of the trigonometric circle. To tell the truth, first I defined them with other logics but I have some errors regarding the not completed rules so I changed them to what it is in the notebook file.

At the end, I rendered the environment and iterated it until 500 rounds(or if the y_position is more than 0.99 and abs(angular velocity) is less than 1.5).

As the start points are different in each time of running code it may have different outputs. Here is a reward plot of once which the conditions were met in 320 iterations.



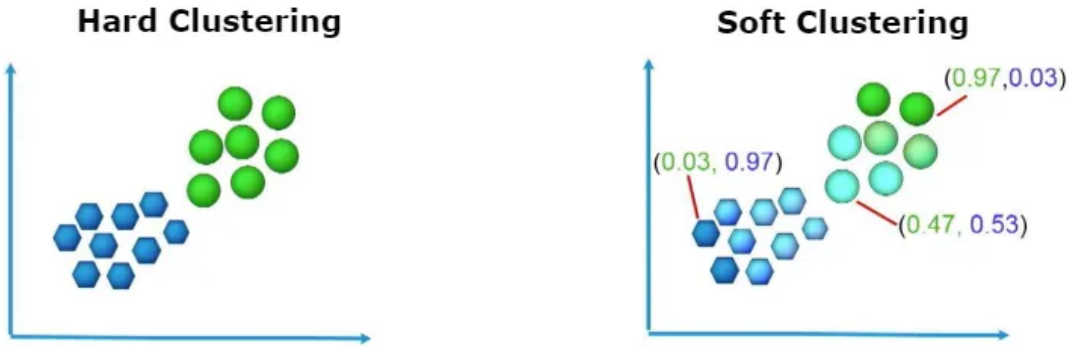
This plot shows that in some points the pendulum was near the place we wanted but as we know the ways that a pendulum moves, it gets far from it and then gets back to that place again. Finally, it could reach that point and if we continued the iterations, the reward would stay near 0 and the pendulum would be approximately motionless.

Q2:

2-1:

Fuzzy C-means (FCM) is a clustering algorithm assigning each point a membership in each cluster center from 0 to 100 percent. It is an extension of the traditional K-means algorithm. Both K-means and FCM are unsupervised learning algorithms used for partitioning a dataset into

clusters based on similarity. However, there are significant differences between the two. I think this image can show the difference clearly:



Fuzzy C-means (FCM) Algorithm:

- Objective Function:

  FCM introduces fuzziness into the clustering process. Instead of assigning each data point to a specific cluster (as in K-means), FCM assigns a degree of membership to each point for every cluster. The objective function of FCM minimizes the weighted sum of squared deviations from the cluster centroids, where the weight is the degree of membership.

- Membership Degrees:

  In FCM, each data point belongs to every cluster with a certain degree of membership between 0 and 1. These membership degrees indicate the likelihood or probability of a data point belonging to a particular cluster.

- Centroid Update:

  The cluster centroids are updated iteratively based on the weighted average of data points, considering their membership degrees. The update rule is based on minimizing the objective function, similar to K-means.

- Fuzziness Parameter (m):

  FCM introduces a parameter "m" (fuzziness parameter) that controls the degree of fuzziness in the clustering. A higher value of "m" results in softer clustering, allowing data points to have memberships spread across multiple clusters.

Differences between Fuzzy C-means and K-means:

- Fuzziness:

  K-means produces hard assignments, meaning each data point belongs exclusively to one cluster. FCM introduces fuzziness by assigning membership degrees, allowing a data point to belong to multiple clusters with varying degrees.

- Membership Degrees vs. Hard Assignments:

  FCM assigns membership degrees to each data point, reflecting the degree of belonging to each cluster. K-means assigns data points to the cluster with the nearest centroid, resulting in hard assignments.

- Sensitivity to Initializations:

  K-means is sensitive to initial cluster centroids, and different initializations can lead to different final clusters. FCM is less sensitive to initializations due to its iterative process of updating membership degrees and centroids.

- Cluster Shape and Size:

  FCM tends to be more flexible in handling clusters with different shapes and sizes compared to K-means, which assumes spherical clusters of roughly equal sizes.

- Fuzziness Parameter:

FCM has an additional parameter (fuzziness parameter "m") that allows users to control the degree of fuzziness in the clustering process. Adjusting this parameter can impact the shape and overlap of clusters.

2-2:

First the dataset is read by using pandas:

```
data1 = pd.read_csv('/content/data1.csv')
```

Then data is normalized with StandardScaler tool:

```
# Normalizing
data1[['X', 'Y']] = StandardScaler().fit_transform(data1[['X', 'Y']])
```

Then the FCM algorithm is applyed for 2 to 10 clustering and it checks with FPC metrics that wich clustering is the best.

FPC:

FPC (Fuzzy Partition Coefficient) is a metric used in fuzzy clustering to evaluate the quality of a fuzzy partition. Fuzzy clustering algorithms, such as Fuzzy C-Means (FCM), assign each data point to multiple clusters with varying degrees of membership. FPC provides a measure of how well the data points are partitioned among the clusters in a fuzzy clustering solution.

The FPC metric takes into account both the degree of membership of data points in their assigned clusters and the overall fuzziness of the clustering. It is defined as the ratio of the sum of the squared membership values to the squared sum of all membership values. Mathematically, it is expressed as:

$$FPC = \frac{\sum_{i=1}^{n} \sum_{j=1}^{c} u_{ij}^2}{\left(\sum_{i=1}^{n} \sum_{j=1}^{c} u_{ij}\right)^2}$$

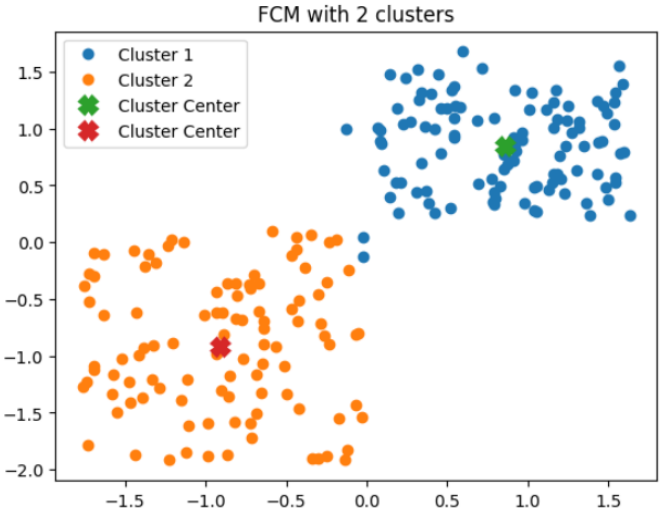
where:

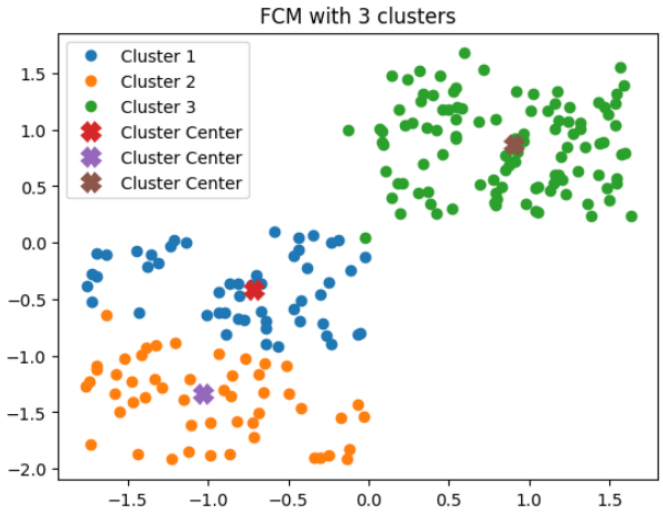
- n is the number of data points.
- c is the number of clusters.
- u_{ij} is the degree of membership of data point i in cluster j.

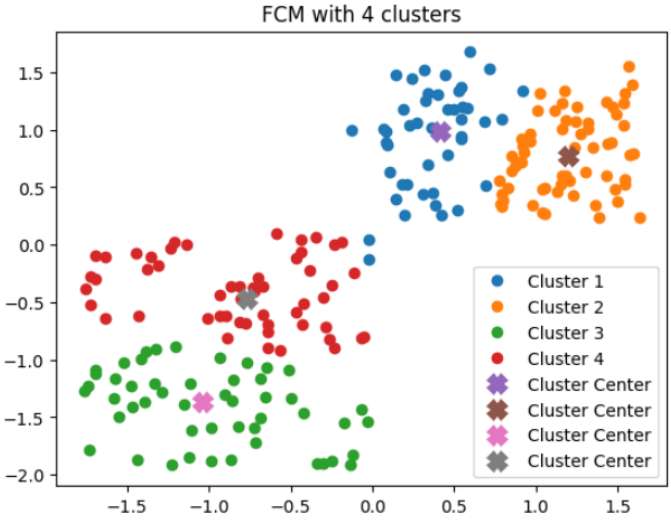
A higher FPC value indicates a better-defined partition, with data points having higher degrees of membership in their assigned clusters.

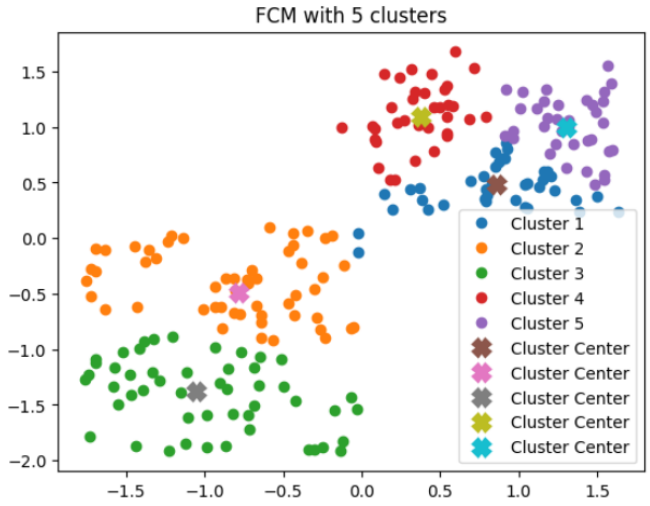
```
finalC=2
finalFPC=0

for c in range(2, 11):
    # FCM algorithm
    cntr, u, u0, d, jm, p, fpc = skfuzzy.cluster.cmeans(data1.T, c, 2, error=0.005, maxiter=1000, init=None)

    cluster_membership = np.argmax(u, axis=0)

    fig, ax = plt.subplots()

    for i in range(c):
        ax.plot(data1['X'][cluster_membership == i], data1['Y'][cluster_membership == i], 'o', label=f'Cluster {i + 1}')

    for cluster_center in cntr:
        ax.plot(cluster_center[0], cluster_center[1], 'X', markersize=10, markeredgewidth=2, label='Cluster Center')

    if fpc > finalFPC:
        finalFPC = fpc
        finalC = c


    # Plot for each cluster
    ax.set_title(f'FCM with {c} clusters')
    ax.legend()
    plt.show()
```

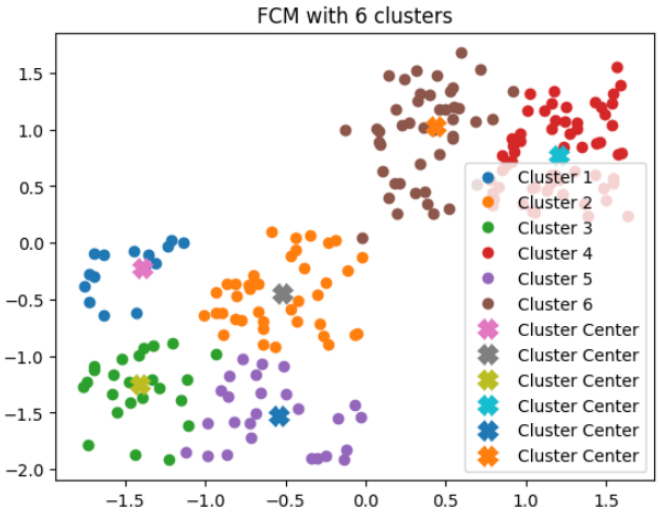
Plots of different clusters:

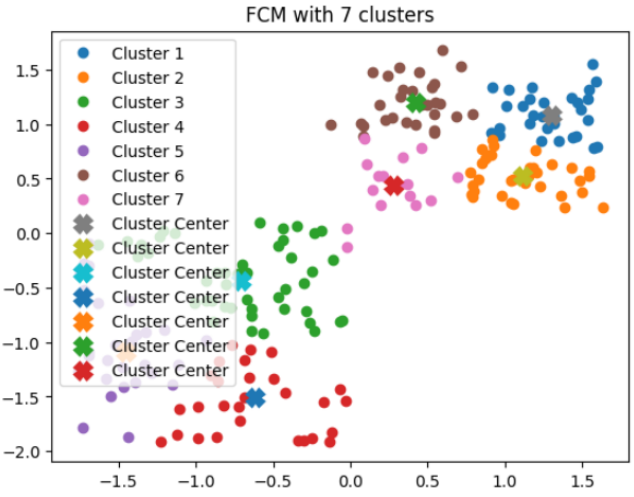


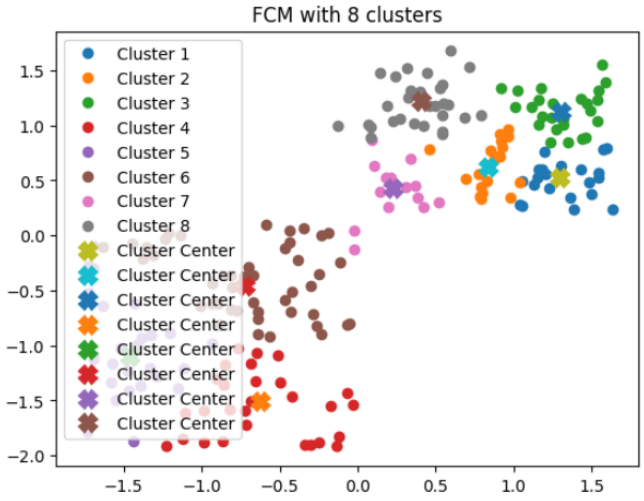


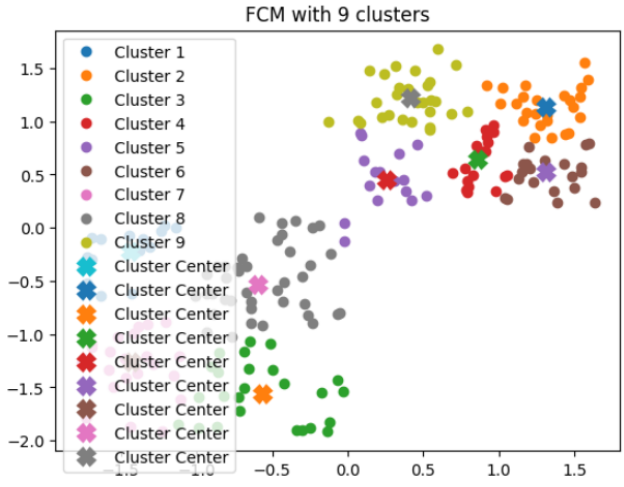

FCM with 4 clusters
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster Center
Cluster Center
Cluster Center
Cluster Center

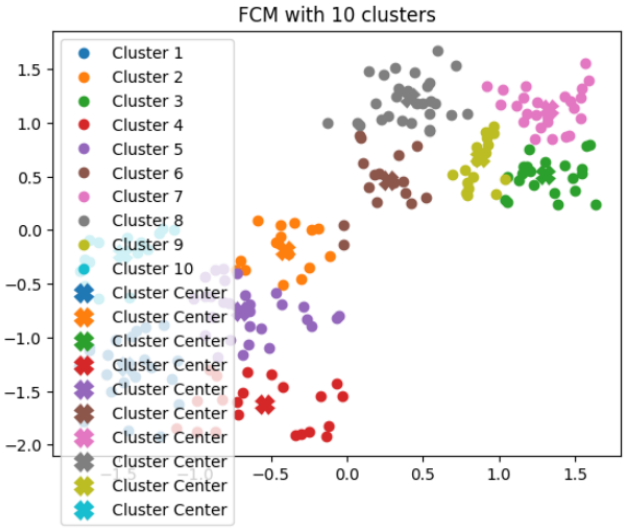

FCM with 5 clusters
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center


FCM with 6 clusters
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center


FCM with 7 clusters
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 7
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center


FCM with 8 clusters
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 7
Cluster 8
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center


FCM with 9 clusters
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 7
Cluster 8
Cluster 9
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center

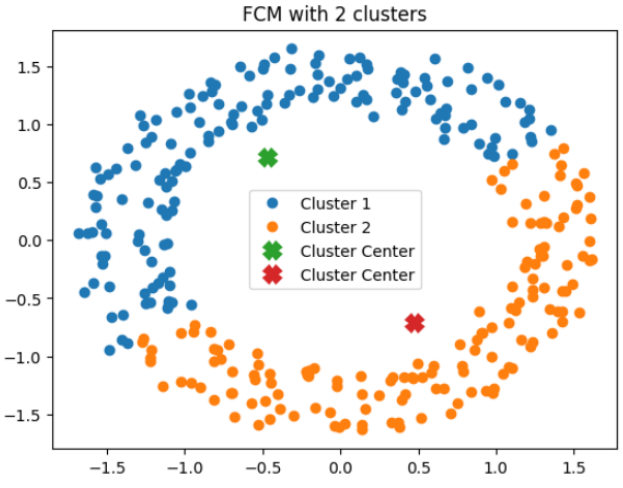


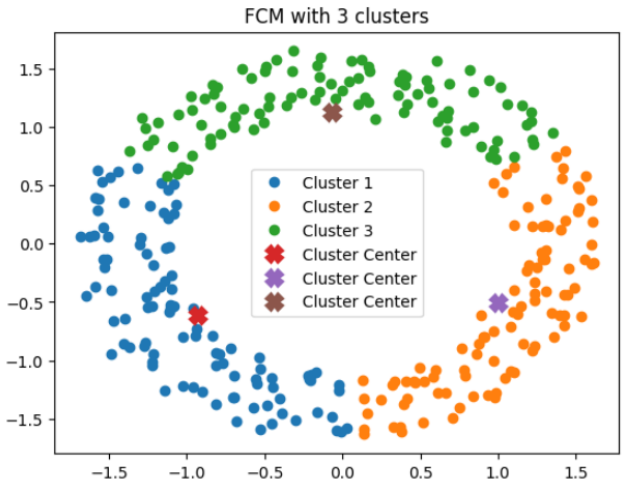
The best FPC is for 2 clustering:

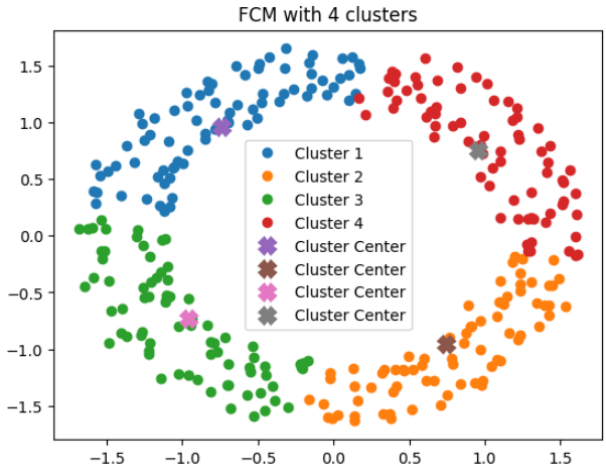
```
1 print(f"the best fpc is {finalFPC} which is with {finalC} clustring")
```

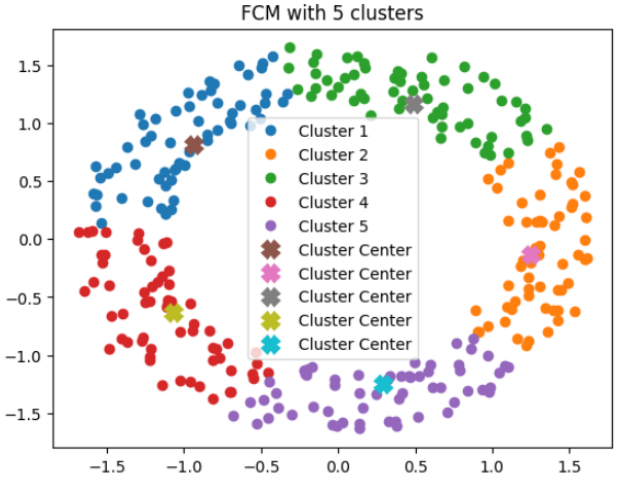
```
the best fpc is 0.8837666916772093 which is with 2 clustring
```

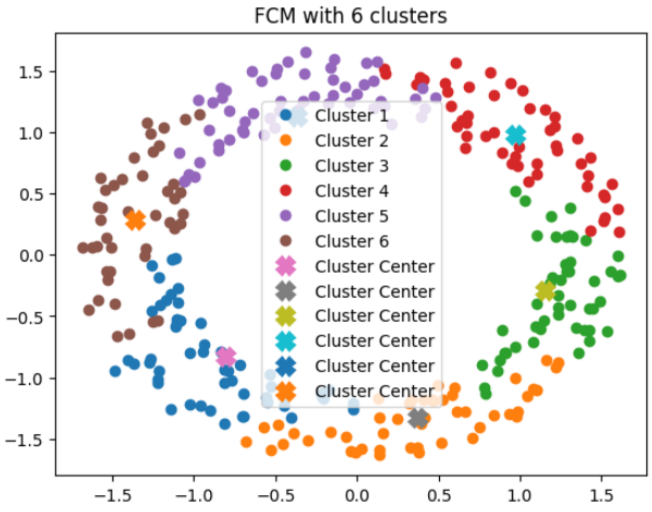
Now all the steps are repeated for dataset2 and here are plots:

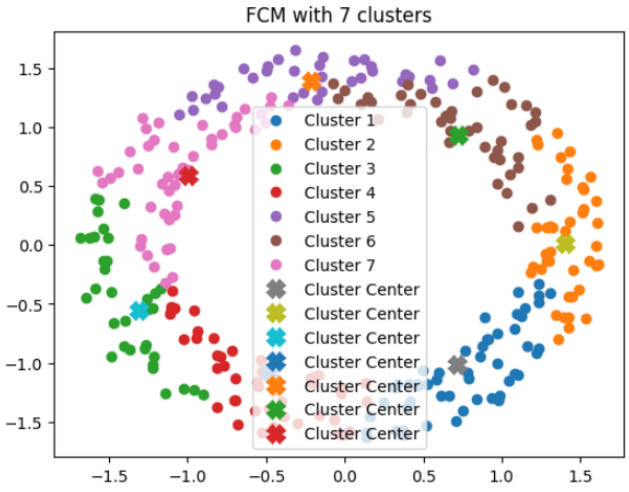


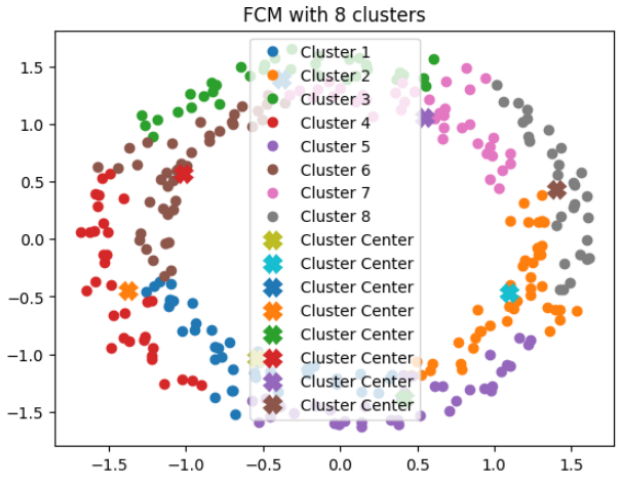


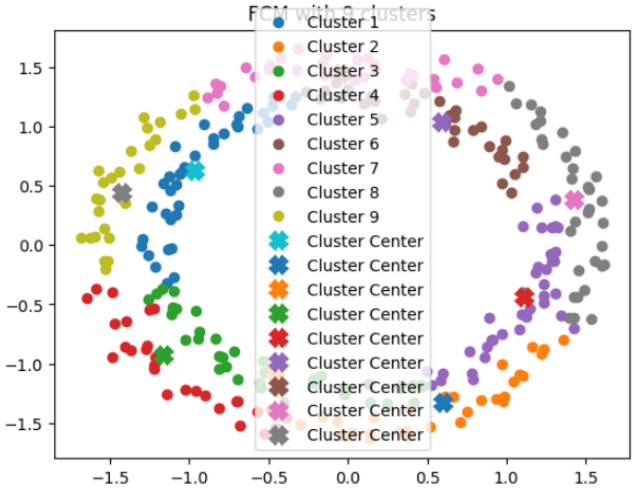

FCM with 4 clusters
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster Center
Cluster Center
Cluster Center
Cluster Center

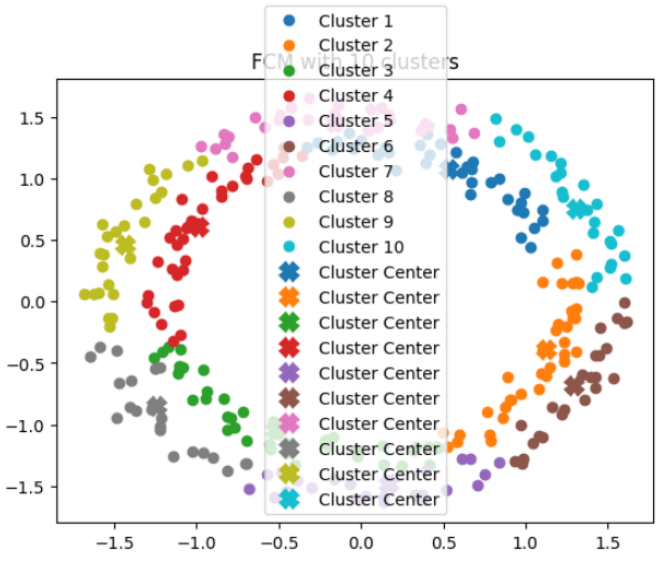

FCM with 5 clusters
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center


FCM with 6 clusters
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center


FCM with 7 clusters
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 7
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center


FCM with 8 clusters
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 7
Cluster 8
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center


Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 7
Cluster 8
Cluster 9
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center
Cluster Center



The best FPC is for 2 clustering:

```
1 print(f"the best fpc is {finalFPC} which is with {finalC} clustring")
```

```
the best fpc is 0.6648792326187354 which is with 2 clustring
```

Q3:

3-1:

First the thin, fat and young should be computed for both persons.

$$\mu_{thin} = \{(55,\ 0.027),\ (95,\ 0.005)\}$$

$$\mu_{fat} = \{(55,\ 0.0975),\ (95,\ 0.6975)\}$$

$$\mu_{young} = \{(45,\ 0.0588),\ (60,\ 0.02)\}$$

Now fairly fat should be computed:

$$\mu_{fairly\ fat} = \mu_{fat}^{\frac{1}{2}} = \{(55,\ 0.312),\ (95,\ 0.835)\}$$

Then the term fairly fatter should be computed in relation to each other($\frac{first - second + 1}{2}$):

$$\mu_{fairly\ fatter} = \{(55,\ 0.2385),\ (95,\ 0.7615)\}$$

Now we should calculate the younger:

$$\mu_{younger} = \{(45,\ 0.5194),\ (60,\ 0.4806)\}$$

Now we know both fairly fatter and younger for the second person. So we should compute the "and" between them(I use minimum for that):

$$\mu_{fairly\ fatter}(second)\ and\ \mu_{younger}(second) = min(0.7615, 0.4806) = 0.4806$$

3-2:

First very thin and fairly young should be calculated:

$$\mu_{very\ thin} = \mu_{thin}{}^{2} = \{(55,\ 0.000729),\ (95,\ 0.000025)\}$$

$$\mu_{fairly\ young} = \mu_{young}{}^{\frac{1}{2}} = \{(45,\ 0.242),\ (60,\ 0.14)\}$$

Now we know both very thin and fairly young for both persons. So we should compute the "implication" between them(I use min(1, 1 + second - first)):

$$If\ \mu_{very\ thin}(first) \rightarrow \mu_{fairly\ young}(second) = min(1,\ 1 + 0.14 - 0.000729) = 1$$