ملیکا محمدی فخار
۹۹۵۲۲۰۸۶

1.

defining and visualizing linguistic variables for the fuzzy logic controller:

- **Angle Linguistic Variable:**

The universe of discourse for 'angle' spans from -π to π with a granularity of 0.1.
Membership Functions:
'Negative': Triangular membership function with support on the interval [-π, 0].
'Zero': Trapezoidal membership function covering the range from -π to π/2 and π/2 to π.
'Positive': Triangular membership function with support on the interval [0, π].

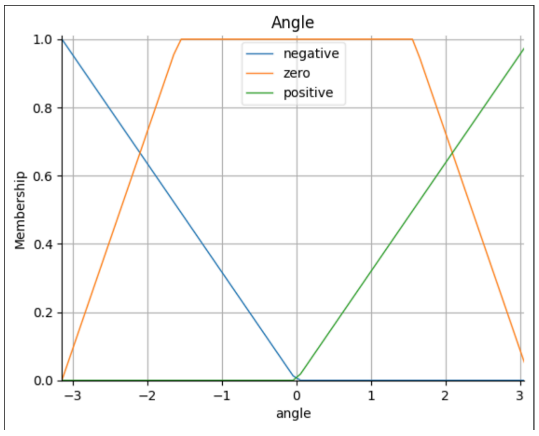
- **Angular Velocity Linguistic Variable:**

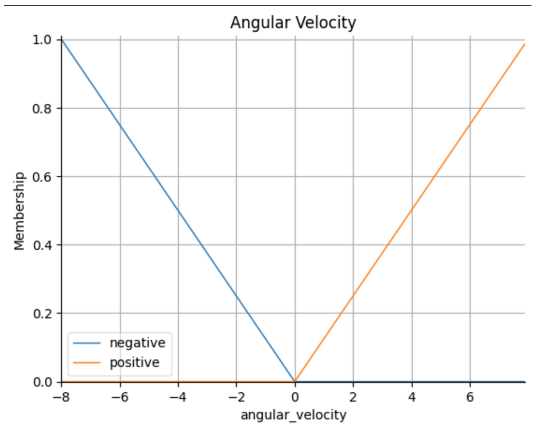
The universe of discourse for 'angular_velocity' ranges from -8 to 8 with a step size of 0.1.
Membership Functions:
'Negative': Triangular membership function with support on the interval [-8, 0].
'Positive': Triangular membership function with support on the interval [0, 8].

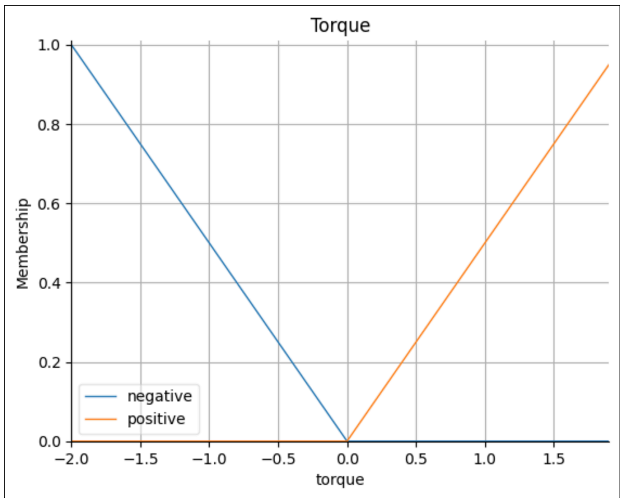
- **Torque Linguistic Variable:**

The universe of discourse for 'torque' spans from -2 to 2 with a step size of 0.1.
Membership Functions:
'Negative': Triangular membership function with support on the interval [-2, 0].
'Positive': Triangular membership function with support on the interval [0, 2].

Visualizations:





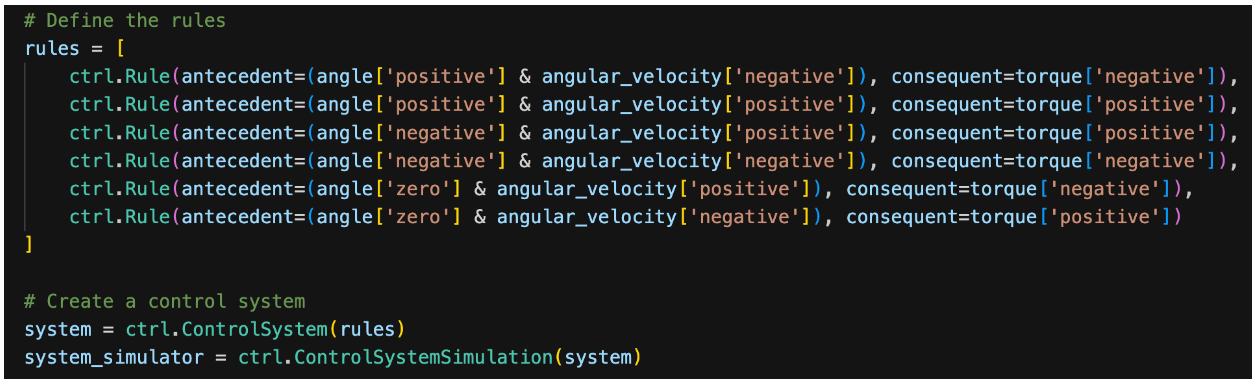


Fuzzy Control Rules:

Rule 1: Positive Angle & Negative Angular Velocity → Negative Torque
Rule 2: Positive Angle & Positive Angular Velocity → Positive Torque
Rule 3: Negative Angle & Positive Angular Velocity → Positive Torque
Rule 4: Negative Angle & Negative Angular Velocity → Negative Torque
Rule 5: Zero Angle & Positive Angular Velocity → Negative Torque
Rule 6: Zero Angle & Negative Angular Velocity → Positive Torque

Control System Creation and Simulation:

- The defined rules are organized into a control system using the 'ctrl.ControlSystem' class.
- A control system simulation environment ('system_simulator') is established to execute the fuzzy logic rules and evaluate the system's response under various input conditions.



```python
# Define the rules
rules = [
    ctrl.Rule(antecedent=(angle['positive'] & angular_velocity['negative']), consequent=torque['negative']),
    ctrl.Rule(antecedent=(angle['positive'] & angular_velocity['positive']), consequent=torque['positive']),
    ctrl.Rule(antecedent=(angle['negative'] & angular_velocity['positive']), consequent=torque['positive']),
    ctrl.Rule(antecedent=(angle['negative'] & angular_velocity['negative']), consequent=torque['negative']),
    ctrl.Rule(antecedent=(angle['zero'] & angular_velocity['positive']), consequent=torque['negative']),
    ctrl.Rule(antecedent=(angle['zero'] & angular_velocity['negative']), consequent=torque['positive'])
]

# Create a control system
system = ctrl.ControlSystem(rules)
system_simulator = ctrl.ControlSystemSimulation(system)
```

Calculate angle function:

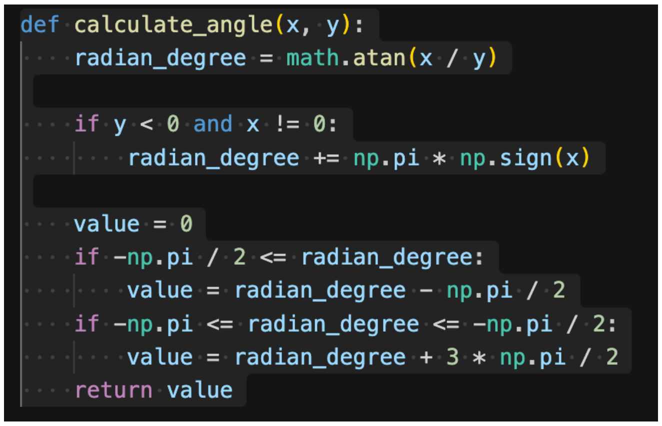
The function computes the arctangent of the ratio x / y to obtain the angle in radians. This is a standard approach to determining the angle formed by the coordinates (x, y).

- Adjustment for Quadrant and Negative y:

If y is negative and x is not zero, the function adjusts the angle by adding π multiplied by the sign of x. This adjustment accounts for the fact that the arctangent function alone may not correctly determine the quadrant when y is negative.

- Range Adjustment:

The function further adjusts the angle to ensure it falls within a specific range.
If the angle is between -π / 2 and 0, it is shifted by subtracting π / 2.
If the angle is between -π and -π / 2, it is shifted by adding 3 * π / 2.



```python
def calculate_angle(x, y):
    radian_degree = math.atan(x / y)

    if y < 0 and x != 0:
        radian_degree += np.pi * np.sign(x)

    value = 0
    if -np.pi / 2 <= radian_degree:
        value = radian_degree - np.pi / 2
    if -np.pi <= radian_degree <= -np.pi / 2:
        value = radian_degree + 3 * np.pi / 2
    return value
```

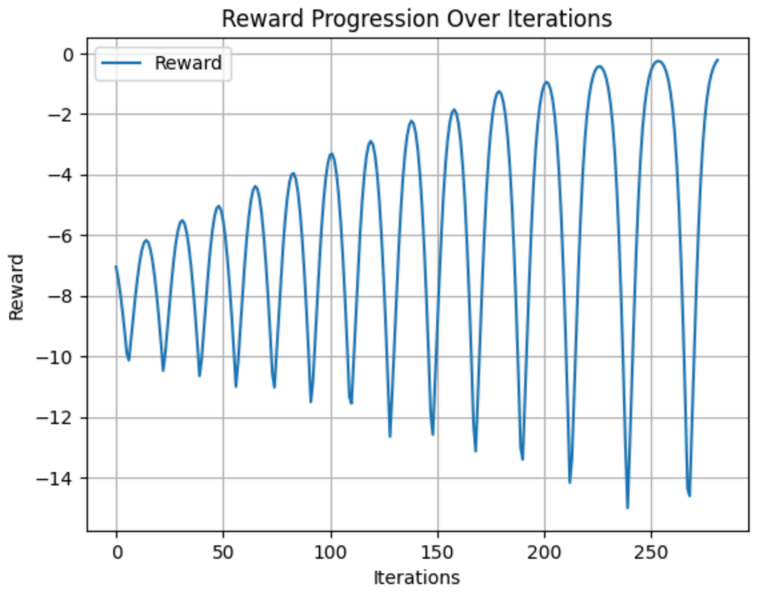
Reward List:
The reward_list is a container that accumulates rewards obtained at each time step during the simulation.

Break Statement:
Exits the loop, terminating the simulation. Once this condition (X >= 0.99 and the speed =< 1.5) is met, the loop will not continue to execute.

Interpretation of Rewards:

- Distance from the Highest Point:
  When the pendulum is away from the highest point of its swing (oscillation), it experiences a torque applied in the direction of its current motion.
  The purpose of this torque is to enhance the pendulum's speed, encouraging it to move more dynamically.
- Approaching the Highest Point:
  As the pendulum approaches the highest point of its swing, a torque in the opposite direction is introduced.
  This torque is designed to decelerate the pendulum, reducing its speed as it gets closer to the highest point.

Reward Progression Plot:
As the model applies force to oscillate the pendulum, it experiences varying levels of success. The periodicity in rewards arises because the model alternately moves away from the peak point and then approaches



2.

a. how does the C-Mean works?

- Initialization:

Randomly assign initial cluster centroids.
Initialize a membership matrix, where each entry represents the degree of membership of a data point to a cluster.

- Update Membership Matrix:

Calculate the membership of each data point to each cluster using a fuzzy membership function.
The membership values are real numbers between 0 and 1, indicating the degree of belongingness to each cluster.

- Update Cluster Centers:

Update the cluster centroids using the updated membership values.
The new centroid of a cluster is computed as a weighted average of all data points, with weights given by the membership values.

- Repeat:

Iteratively update the membership matrix and cluster centroids until convergence or a specified stopping criterion is met.
The fuzzy membership function in FCM allows for a soft assignment of data points to clusters, capturing the uncertainty in the assignment. This is in contrast to the hard assignment in K-Means, where each data point belongs to only one cluster.

The main differences between Fuzzy C-Means and K-Means are:

- Membership Values:

FCM assigns membership values to each data point for every cluster, indicating the degree of belongingness. K-Means, on the other hand, assigns each data point to a single cluster with a binary membership.

- Hard vs. Soft Clustering:

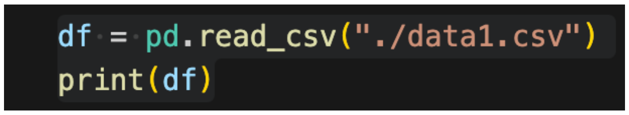
K-Means performs hard clustering, meaning each data point is exclusively assigned to one cluster. FCM introduces soft clustering by allowing data points to have partial membership in multiple clusters.

• Sensitivity to Initial Conditions:
FCM is less sensitive to initial cluster centroids than K-Means. This is because FCM involves a continuous membership assignment, whereas K-Means relies on assigning each point to the nearest centroid.

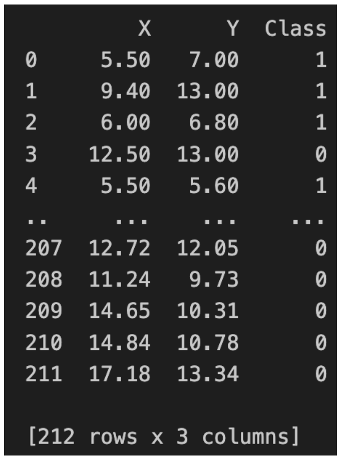
• Objective Function:
The objective function in FCM considers the sum of the squared deviations of each data point from all cluster centroids, weighted by their membership values. K-Means minimizes the sum of squared distances of each data point to its assigned cluster centroid.

b. code

I. The pd.read_csv() function is used to read data from a CSV file into a Pandas DataFrame. and then I printed the data frame:



```
df = pd.read_csv("./data1.csv")
print(df)
```



```
         X      Y  Class
0     5.50   7.00      1
1     9.40  13.00      1
2     6.00   6.80      1
3    12.50  13.00      0
4     5.50   5.60      1
..     ...    ...    ...
207  12.72  12.05      0
208  11.24   9.73      0
209  14.65  10.31      0
210  14.84  10.78      0
211  17.18  13.34      0

[212 rows x 3 columns]
```

II.

Feature Selection:
The variable feats is defined as a list containing the names of the features that need to be normalized. In this case, the features are labeled as 'X' and ‘Y'.

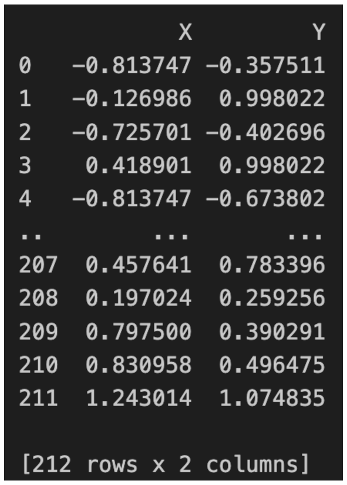
Data Normalization using StandardScaler:
The StandardScaler from scikit-learn is utilized to standardize the selected features in the DataFrame. The apply method is employed to normalize

each feature individually, and the lambda function within applies the standardization to each feature.

Creating a New DataFrame with Normalized Data:
The normalized data is stored in a new DataFrame named df_normalized_standard. The normalized values are applied to the 'X' and 'Y' features, preserving the original structure of the DataFrame.

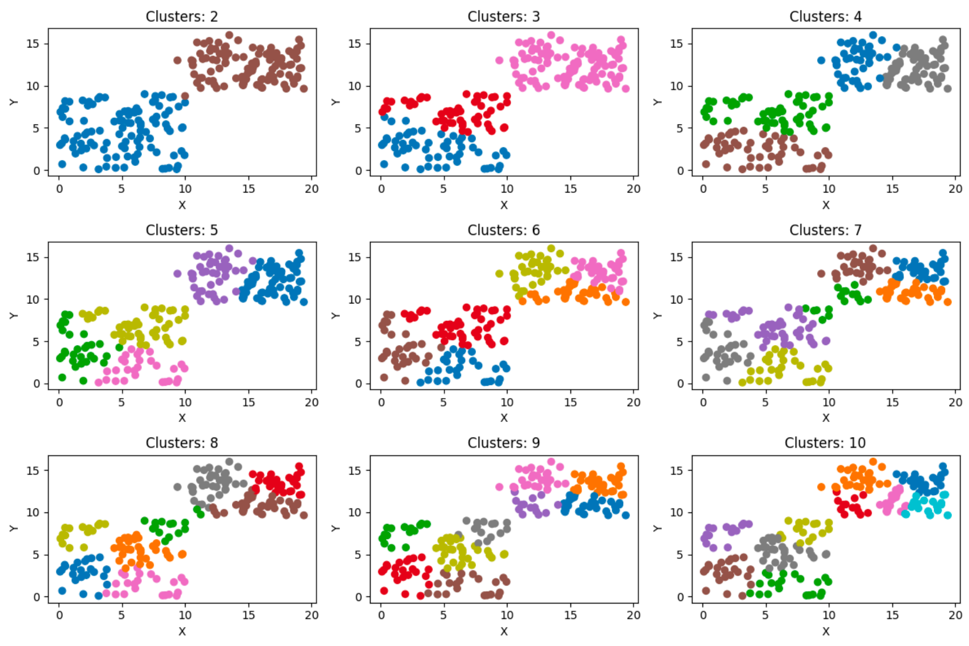
Column Renaming:
The columns in the new DataFrame are renamed to 'X' and 'Y' using df_normalized_standard.columns = feats. This step ensures that the columns retain their original names after normalization.

Printing the Normalized Data:

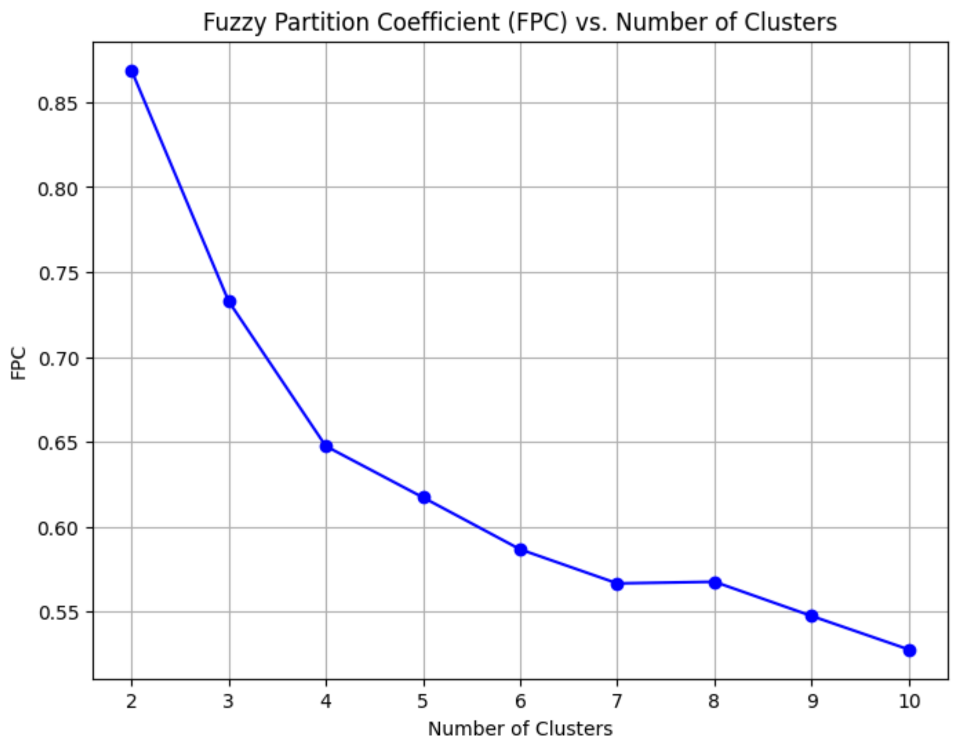


```
            X         Y
0   -0.813747 -0.357511
1   -0.126986  0.998022
2   -0.725701 -0.402696
3    0.418901  0.998022
4   -0.813747 -0.673802
..        ...       ...
207  0.457641  0.783396
208  0.197024  0.259256
209  0.797500  0.390291
210  0.830958  0.496475
211  1.243014  1.074835

[212 rows x 2 columns]
```

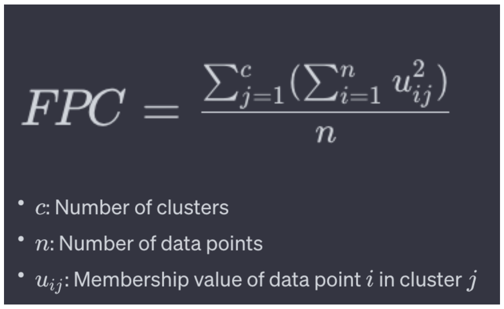
III. I generated a set of subplots, each representing a different number of clusters ranging from 2 to 10. For each cluster count, fuzzy c-means clustering is applied, and the resulting cluster memberships are visualized using scatter plots, where data points are colored based on their cluster memberships. The Fuzzy Partition Coefficient (FPC) values for each clustering iteration are calculated and stored:



IV. and a separate plot is generated to illustrate the relationship between the number of clusters and the corresponding FPC values:
It’s clear that with 2 cluster we have the best fpc.



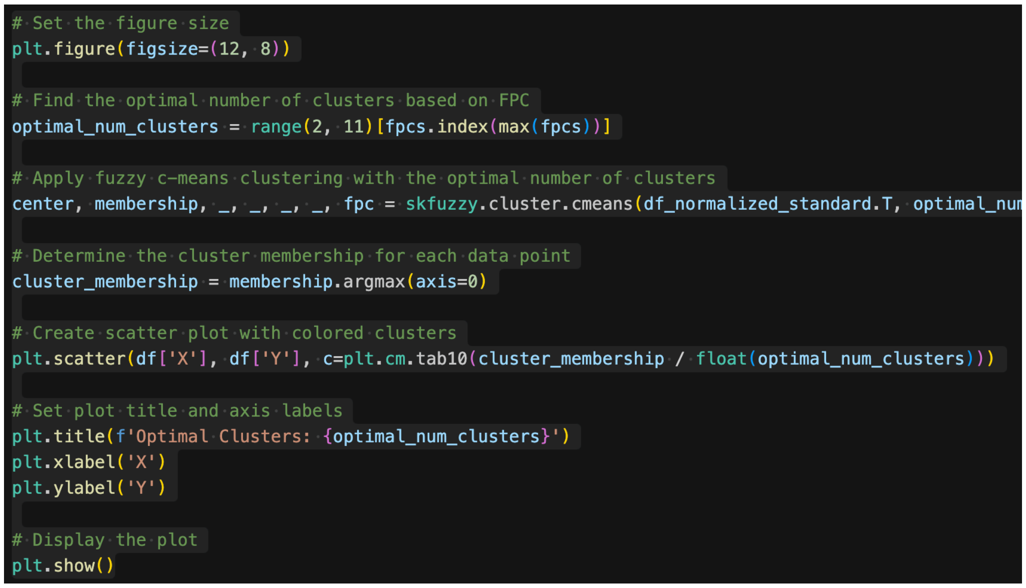
V. The Fuzzy Partition Coefficient (FPC) is a metric used to evaluate the quality of a fuzzy clustering solution. It provides a measure of how well-defined and separated the clusters are in a fuzzy clustering result. FPC values range from 0 to 1, where a higher FPC indicates better-defined clusters. The formula for FPC is based on the ratio of the trace of the covariance matrix of the membership function to the sum of the squared membership values:



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

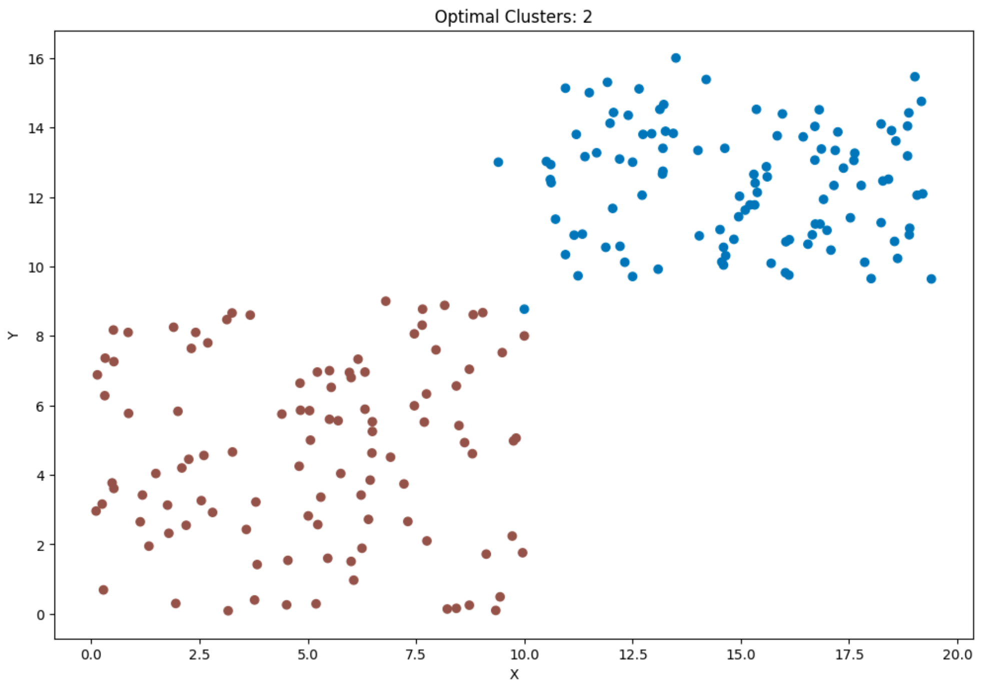
- c: Number of clusters
- n: Number of data points
- u_{ij}: Membership value of data point i in cluster j

code explanation:
I identified the optimal number of clusters, denoted as optimal_num_clusters, by selecting the cluster count associated with the maximum Fuzzy Partition Coefficient (FPC) value from the previously calculated list of FPC values. I then applied fuzzy c-means clustering to the standardized data, utilizing this optimal number of clusters (optimal_num_clusters). In this process, I determined the cluster memberships and calculated the FPC value for the optimal clustering solution.

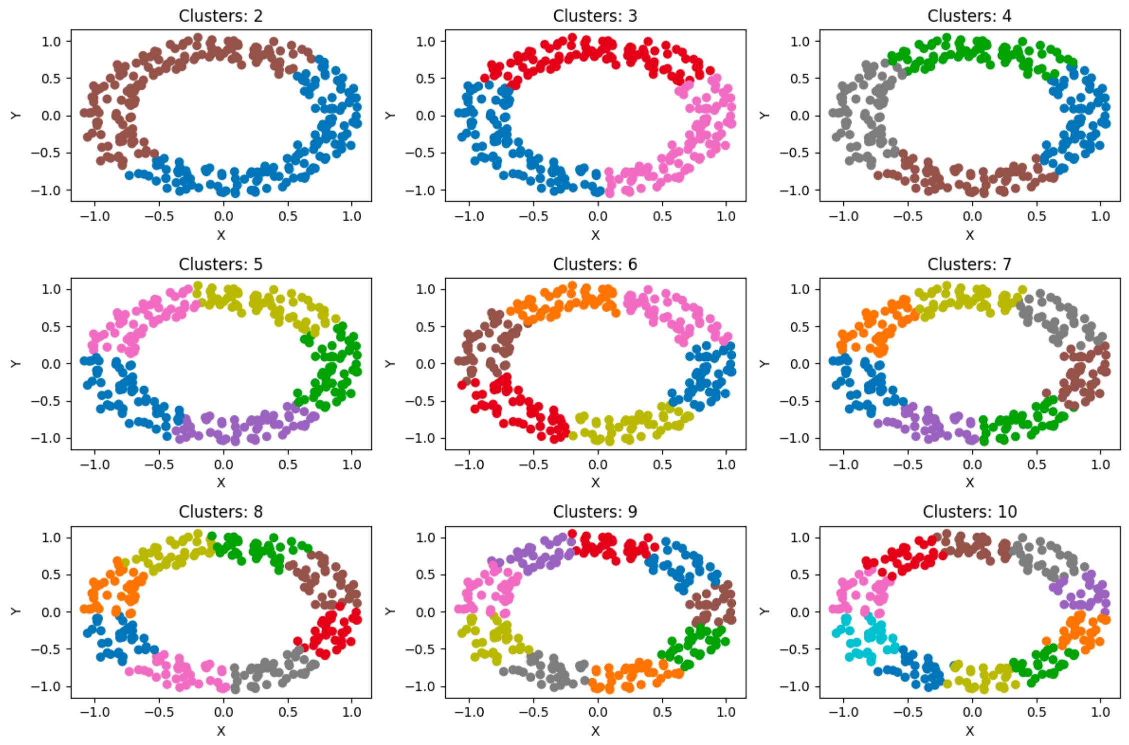


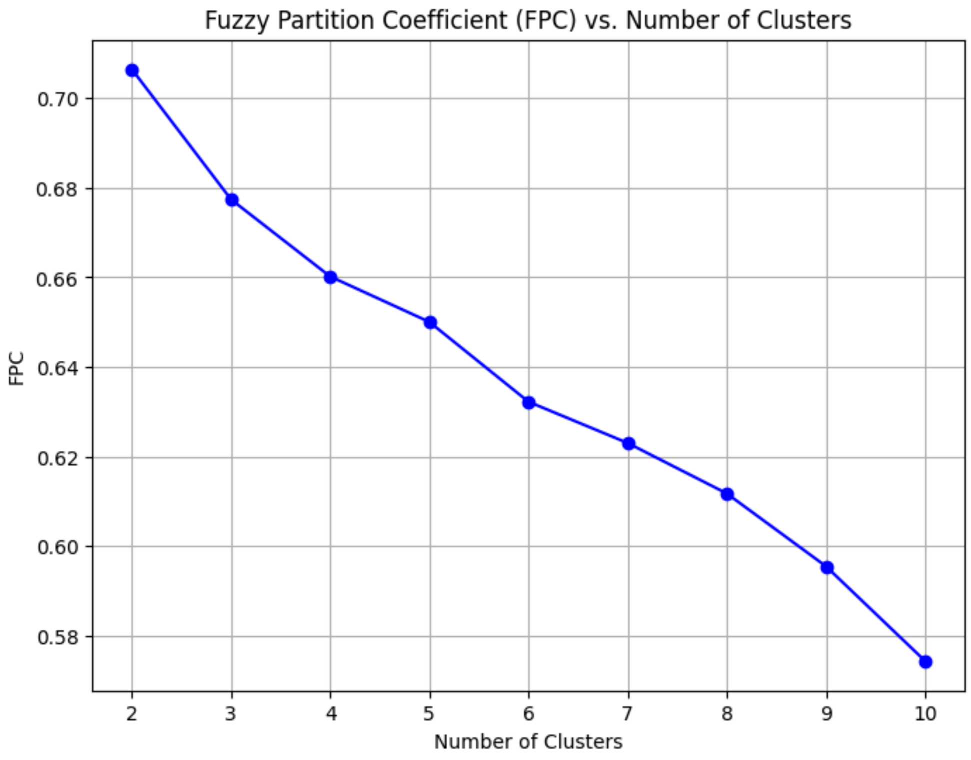
```
# Set the figure size
plt.figure(figsize=(12, 8))

# Find the optimal number of clusters based on FPC
optimal_num_clusters = range(2, 11)[fpcs.index(max(fpcs))]

# Apply fuzzy c-means clustering with the optimal number of clusters
center, membership, _, _, _, _, fpc = skfuzzy.cluster.cmeans(df_normalized_standard.T, optimal_nu

# Determine the cluster membership for each data point
cluster_membership = membership.argmax(axis=0)

# Create scatter plot with colored clusters
plt.scatter(df['X'], df['Y'], c=plt.cm.tab10(cluster_membership / float(optimal_num_clusters)))

# Set plot title and axis labels
plt.title(f'Optimal Clusters: {optimal_num_clusters}')
plt.xlabel('X')
plt.ylabel('Y')

# Display the plot
plt.show()
```

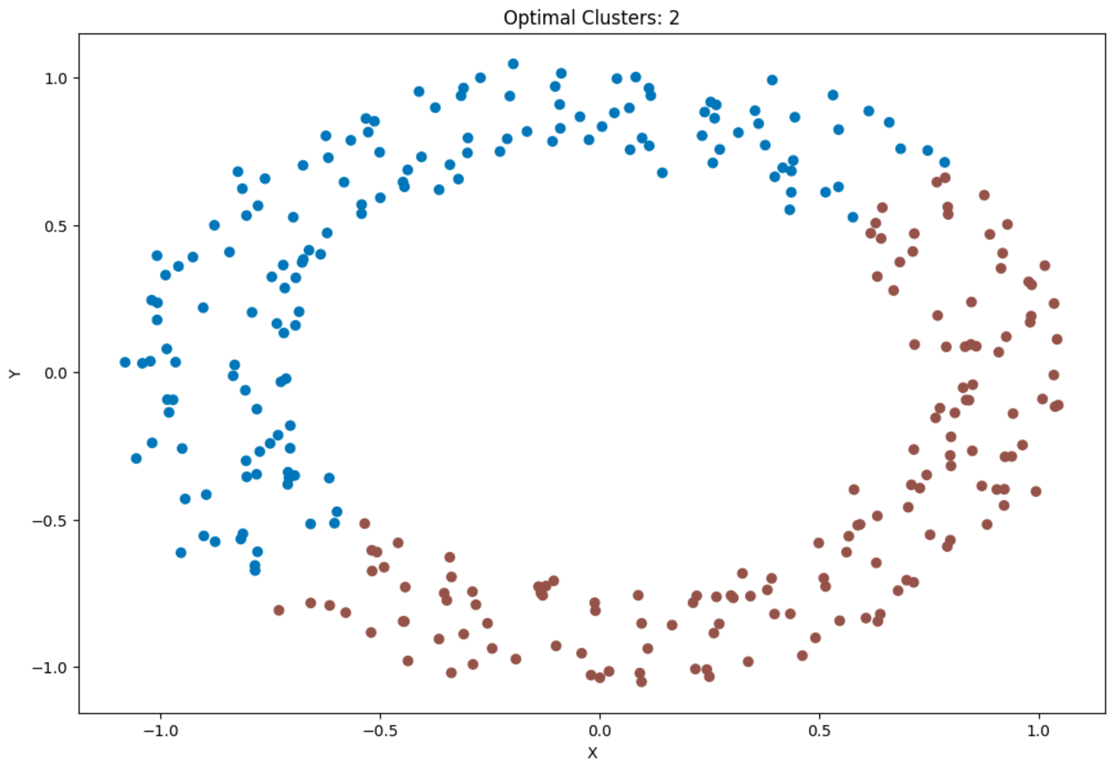
output:



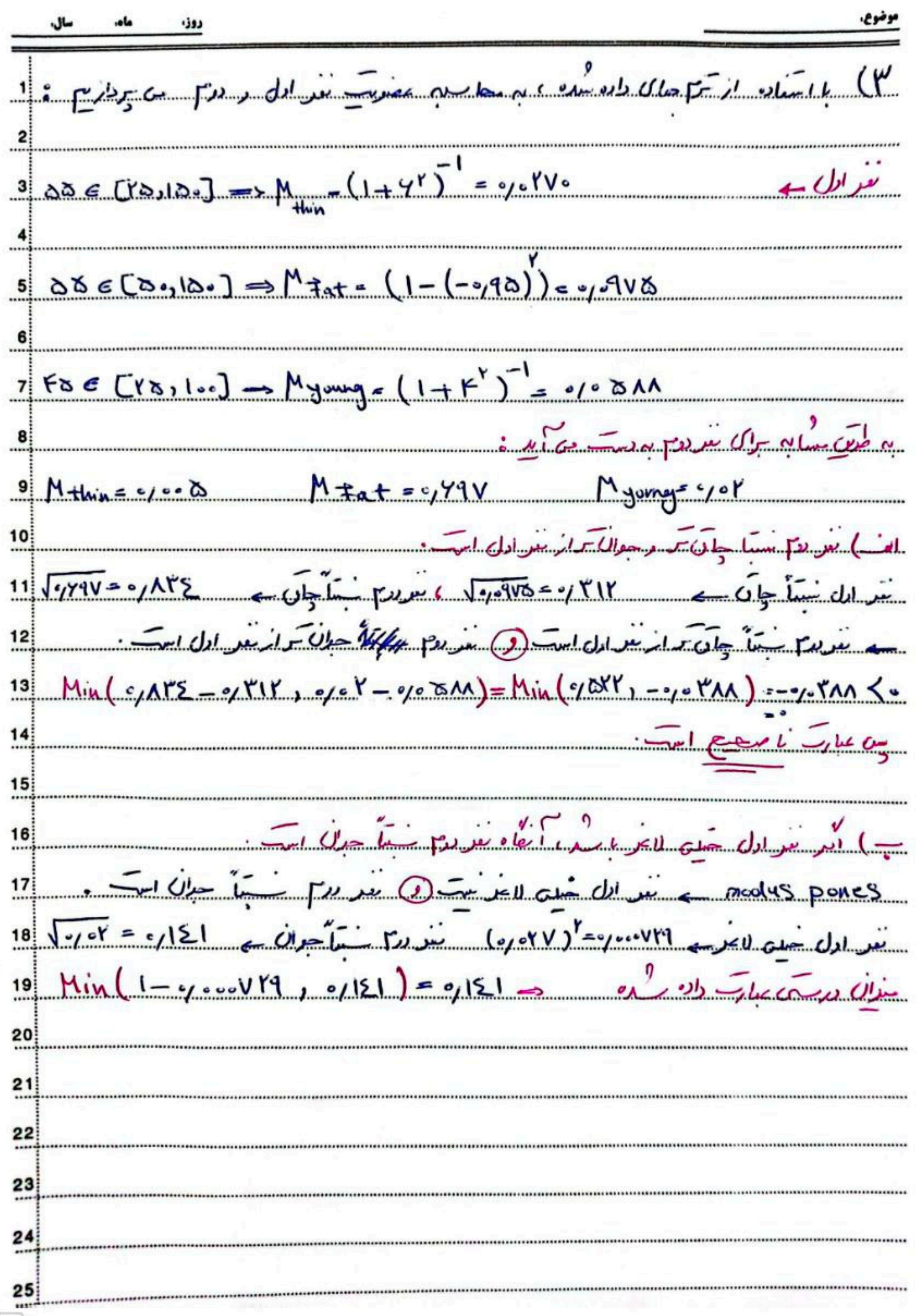
then i repeated same steps for data2 and here are the results:




Fuzzy Partition Coefficient (FPC) vs. Number of Clusters
FPC
0.70
0.68
0.66
0.64
0.62
0.60
0.58
2
3
4
5
6
7
8
9
10
Number of Clusters


Optimal Clusters: 2
Y
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
X

3.



موضوع:　　　　روز:　ماه:　سال:

۳) با استفاده از ترم های داده شده ، به محاسبه عضویت نفر اول و دوم می پردازیم :

نفر اول ←

$۵۵ \in [۲۵,۱۵۰] \Rightarrow M_{thin} = (1+۶^۲)^{-1} = ۰٫۰۲۷۰$

$۵۵ \in [۵۰,۱۵۰] \Rightarrow M_{fat} = (1-(-۰٫۹۵)^۲) = ۰٫۰۹۷۵$

$۴۵ \in [۲۵,۱۰۰] \rightarrow M_{young} = (1+۴^۲)^{-1} = ۰٫۰۵۸۸$

به طریق مشابه برای نفر دوم به دست می آید :

$M_{thin} = ۰٫۰۰۵ \qquad M_{fat} = ۰٫۶۹۷ \qquad M_{young} = ۰٫۰۲$

الف) نفر دوم نسبتاً چاق تر و جوان تر از نفر اول است.

نفر اول نسبتاً چاق ← $\sqrt{۰٫۰۹۷۵} = ۰٫۳۱۲$ ، نفر دوم نسبتاً چاق ← $\sqrt{۰٫۶۹۷} = ۰٫۸۳۴$

← نفر دوم نسبتاً چاق تر از نفر اول است (و) نفر دوم ~~[illegible]~~ جوان تر از نفر اول است.

$Min(۰٫۸۳۴ - ۰٫۳۱۲ \ , \ ۰٫۰۲ - ۰٫۰۵۸۸) = Min(۰٫۵۲۲ \ , \ -۰٫۰۳۸۸) = -۰٫۰۳۸۸ < ۰$

پس عبارت نا صحیح است.

ب) اگر نفر اول خیلی لاغر باشد، آنگاه نفر دوم نسبتاً جوان است.

modus pones ← نفر اول خیلی لاغر نیست (و) نفر دوم نسبتاً جوان است.

نفر اول خیلی لاغر ← $(۰٫۰۲۷)^۲ = ۰٫۰۰۰۷۲۹$ 　نفر دوم نسبتاً جوان ← $\sqrt{۰٫۰۲} = ۰٫۱۴۱$

میزان درستی عبارت داده شده ⇐ $Min(1 - ۰٫۰۰۰۷۲۹ \ , \ ۰٫۱۴۱) = ۰٫۱۴۱$