The Matlab code for all experiments is in the **Appendix** section.

6.1. In this task, we are performing fuzzy c-means clustering with the fuzzifier parameter $q = 2$ and distance measure $d(x_i, \theta_j) = (x_i - \theta_j)^T A(x_i - \theta_j)$ with $A = \mathbf{I}$ on GMD dataset from the homework 1 and by considering that there are four significant clusters $m = 4$ represented by centroid or mean center.

We randomly initialize the four clusters centroid using uniformly distribution random generator which gives the value between 0 and 1 then we got:

$$\Theta^{(0)} = [\theta_1^{(0)}\ \theta_2^{(0)}\ \theta_3^{(0)}\ \theta_4^{(0)}]; \tag{1}$$

$$= \begin{bmatrix} 0.7802 & 0.6079 & 0.1048 & 0.5495 \\ 0.3376 & 0.7413 & 0.1279 & 0.4852 \end{bmatrix} \tag{2}$$

In the fuzzy c-means algorithm, we need to first compute $U = [u_{ij}]$ matrix where

$$u_{ij} = u_j(x_i) \tag{3}$$

$$= \frac{1}{\sum_{k=1}^{m} \left(\frac{d(x_i,\theta_j)}{d(x_i,\theta_k)}\right)^{\frac{1}{q-1}}} \tag{4}$$

then updating the parameter θ_j by solving $\sum_{i=1}^{N} u_{ij}^q \frac{\partial d(x_i, \theta_j)}{\partial \theta_j} = 0$ and we obtain:

$$\theta_j = \frac{\sum_{i=1}^{N} u_{ij}^q x_i}{\sum_{i=1}^{N} u_{ij}^q} \tag{5}$$

We repeat this process until the termination criterion $||\Theta(t) - \Theta(t-1)|| < \epsilon$ where $\epsilon = 0.001$ is met and the final values of Θ (cluster centroids) is:

$$\Theta = \begin{bmatrix} 13.2092 & 0.8970 & 8.5777 & 4.5329 \\ 2.8618 & 1.7325 & 6.4067 & 6.7135 \end{bmatrix} \tag{6}$$

6.2. With the estimated cluster centroids we can perform cluster assignment by assigning each samples to the closest cluster.

$$k_n^* = \underset{k}{\operatorname{argmin}}\, ||x_n - \theta_k||^2 \tag{7}$$

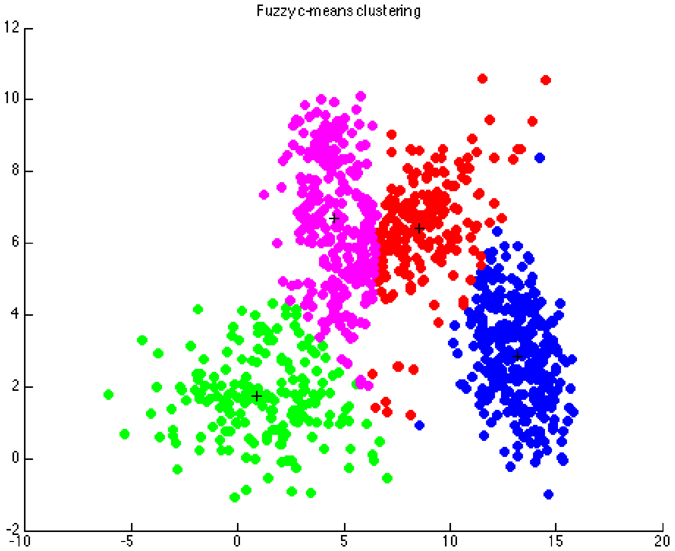

Figure 1: Plot of samples for different clusters

6.3. Finally, we computed the total distortion $\sum_{n=1}^{N} ||x_n - \theta_{k_n}||^2$ for each iteration = [27360, 24016, 17841, 11821, 9014, 7400, 5820, 4925, 4709, 4683, 4689, 4696, 4700, 4704, 4706, 4708, 4709, 4710, 4711, 4712, 4712, 4713, 4713, 4713, 4714, 4714, 4714, 4714, 4714, 4714].

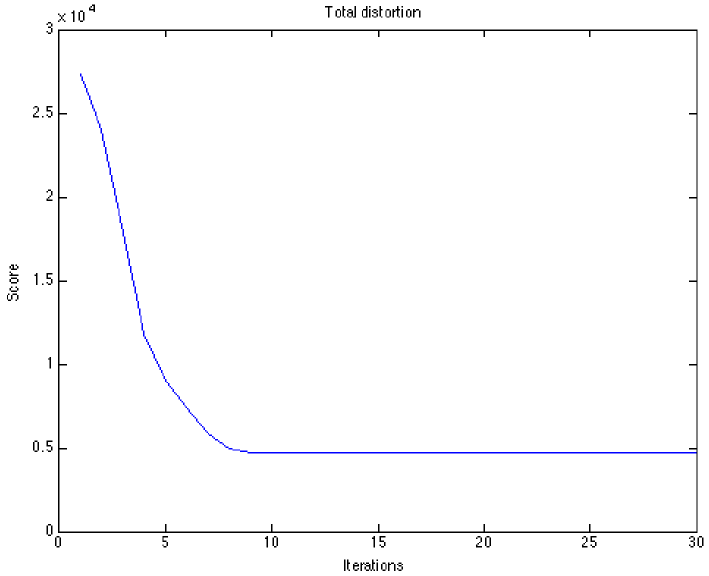

Figure 2: Total distortion

Appendix:

assignment_3.m

```
clc;
clear all;
close all;

problem_3
problem_4
```

problem_3.m

```
% dissimilarity matrix
prox_mat = [0 2 4.2 6.6 9.2 12 15 300 340 420
            2 0 2.2 4.6 7.2 10 13 280 320 400
            4.2 2.2 0 2.4 5 7.8 10.8 270 310 390
            6.6 4.6 2.4 0 2.6 5.4 8.4 260 300 380
            9.2 7.2 5.0 2.6 0 2.8 5.8 262 296 388
            12 10 7.8 5.4 2.8 0 3 316 280 414
            15 13 10.8 8.4 5.8 3 0 380 326 470
            300 280 270 260 262 316 380 0 4 4.4
            340 320 310 300 296 280 326 4 0 9
            420 400 390 380 388 414 470 4.4 9 0];

% single linkage algorithm
[ Zs, ls ] = linkage(prox_mat, 'single');
display('Single␣linkage:');
N = length(ls);
for i=1:N
    print_cluster(Zs{i}, ls(i));
end

% complete linkage algorithm
[ Zc, lc ] = linkage(prox_mat, 'complete');
N = length(lc);
display('Complete␣linkage:');
for i=1:N
    print_cluster(Zc{i}, lc(i));
end
```

problem_4.m

```
clc; clear all; close all;
% load data
X = load('../1/GMD.dat');

% number of clusters
m = 4;
% fuzzifier
q = 2;

[~,d] = size(X);
% initialization
% Theta = rand(d,m);
Theta = [0.7802    0.6079    0.1048    0.5495
                        0.3376    0.7413    0.1279    0.4852];
display('Random␣initialization:');
display(Theta);

[I, Theta, distortion] = fuzzy_c_mean(X, Theta, q);

% result
display('Result:');
display(Theta);
display(distortion);
% plot data
K = unique(I)';
color = 'bgrm';
figure; hold on;
for k=K
    Ck = X(I==k,:);
    scatter(Ck(:,1), Ck(:,2), 'filled', color(k));
```

```
end
plot(Theta(1,:), Theta(2,:), 'k+');
hold off;
title('Fuzzy␣c-means␣clustering');
% plot distortion
figure;
plot(distortion);
title('Total␣distortion');
xlabel('Iterations');
ylabel('Score');
```

linkage.m

```
function [ R, l ] = linkage( D, algorithm )
% linkage - Agglomerative linkage algorithm
%
%       D           : dissimilarity matrix
%       algorithm   : 'single' or 'complete'

[N,~] = size(D);
R = cell(N,1);
l = zeros(N, 1);

% start with every point is a cluster by itself
R{1} = vec2cell(1:N);
for t=2:N-1
    % upper triangular
    U = triu(D);
    l(t) = non_zero_min(U);
    [r, s] = find(U == l(t));
    % merge r and s
    R{t} = merge(R{t-1}, r, s);
    D = update(D, r, s, algorithm);
end
R{N} = {1:N};
l(N) = non_zero_min(D);

function [ D ] = update(D, r, s, algorithm)
% update - Update distance matrix

dr = D(r,:);
dr([r s]) = [];
ds = D(s,:);
ds([r s]) = [];

% remove merge rows and columns
D([r s],:) = [];
D(:,[r s]) = [];

if strcmpi(algorithm, 'single')
    dq = min([dr;ds]);
elseif strcmpi(algorithm, 'complete')
    dq = max([dr;ds]);
end

D = [dq' D];
D = [0 dq; D];
```

vec2cell.m

```
function [ C ] = vec2cell( v )
% vec2cell - Convert each element of the vector to cell

l = length(v);
C = cell(1, l);

for i=1:l
    C{i} = v(i);
end
```

non_zero_min.m

```
function [ v ] = non_zero_min( X )
```

```
% NON_ZERO_MIN - Get a single non zero min in X

% vectorize
v = X(:);
% remove all zero
v(v==0) = [];
v = min(v);
```

merge.m

```
function [ R ] = merge( R, r, s )
% merge - Merge two clusters r and s

Cq = [R{r} R{s}];
% remove cluster r and s
R([r s]) = [];
% prepend Cq to R
R =  [Cq R];
```

print_cluster.m

```
function print_cluster(C, level)
% print_cluster - print cluster values
%   C     - set of clusters (cell array)
%   level - level that form the clusters

disp(['level(' num2str(level) ')␣=␣']);
disp('␣');
m = length(C);
fprintf('␣␣{');
for i=1:m
    fprintf(['␣{' sprintf('␣x%d␣', C{i}) '}␣']);
end
fprintf('}');
disp('␣');
disp('␣');
```

fuzzy_c_mean.m

```
function [ I, Theta, distortion ] = fuzzy_c_mean( X, Theta, q )
% fuzzy_c_mean - run fuzzy c-mean clustering algorithm on X
%
%        q : fuzzifier

[N,~] = size(X);
[d,m] = size(Theta);
p = 1/(q-1);

t = 0;
distortion = [];
Theta_t = zeros(d,m);
epsilon = 1e-3;

while true
    U = zeros(N, m);
    for i=1:N
        D = distance(X(i,:)', Theta);
        for j=1:m
            U(i,j) = 1/(sum( (D(j)./D).^p ));
        end
    end
    t = t + 1;
    % parameter update
    denominator = sum(U.^q);
    for j=1:m
         Theta_t(:,j) = sum((U(:,j).^q*ones(1,2)) .* X) / denominator(j);
    end

    % total distortion
    [distortion(t), I] = total_distortion(X, Theta_t);
    % check for termination
    % : if the total distortion can only be decreased
%      if t > 1 && distortion(t) > distortion(t-1)
%           break;
%       end
    % : if change in Theta is smaller than epsilon
    c = Theta(:) - Theta_t(:);
    if sqrt(c'*c) < epsilon
        break;
    end
    Theta = Theta_t;
end



function [ D ] = distance( x, Theta )
% distance - compute distance (x-theta)'A(x-theta)

A = eye(2);
[~,m] = size(Theta);

x_tilde = x*ones(1,m) - Theta;
D = diag(x_tilde'*A*x_tilde)';
```

total_distortion.m

```
function [ distortion, I ] = total_distortion( X, Theta )
% total_distortion - compute total distortion

[N,d] = size(X);
[~,m] = size(Theta);
S = zeros(N,d);
I = cluster_assignment(X, Theta);

for j=1:m
    S(I==j,:) = ones(nnz(I==j),1) * Theta(:,j)';
end
distortion = sum( sum( (X-S).^2 ) );


function [ I ] = cluster_assignment( X, Theta )
% cluster_assignment - assign X to clusters

[N,~] = size(X);
[~,m] = size(Theta);
D = zeros(N, m);

for j=1:m
    theta = ones(N,1) * Theta(:,j)';
    D(:, j) = sum((X - theta).^2, 2);
end
[~,I] = min(D, [], 2);
```