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

4.1. 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 $\displaystyle\sum_{i=1}^{N} u_{ij}^q \frac{\partial d(x_i, \theta_j)}{\partial \theta_j} = 0$ 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:

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

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

$$j_n^* = \operatorname*{argmin}_j d(x_n, \theta_j) \tag{7}$$

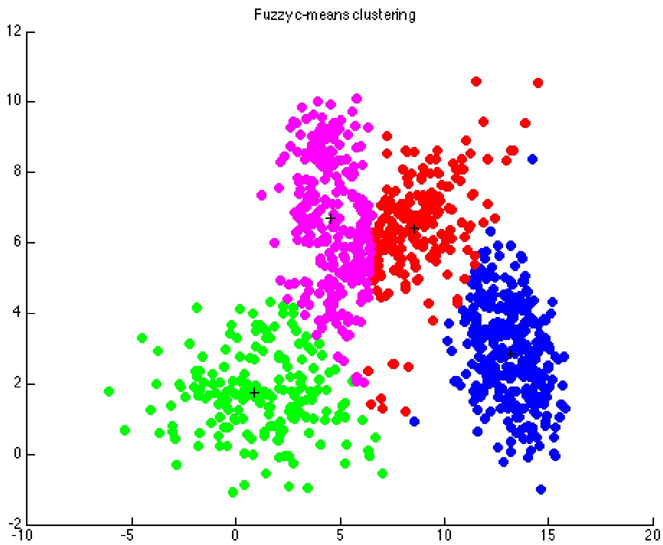

Figure 1: Plot of samples for different clusters

4.3. Finally, we computed the total distortion for each iteration.

$$D(i) = \sum_{n=1}^{N} \min_j d(x_n, \theta_j(i)) \tag{8}$$

i	1	2	3	4	5	6	7	8	9	10
$D(i)$	27360	24016	17841	11821	9014	7400	5820	4925	4709	4683

i	11	12	13	14	15	16	17	18	19	20
$D(i)$	4689	4696	4700	4704	4706	4708	4709	4710	4711	4712

i	21	22	23	24	25	26	27	28	29
$D(i)$	4712	4713	4713	4713	4714	4714	4714	4714	4714

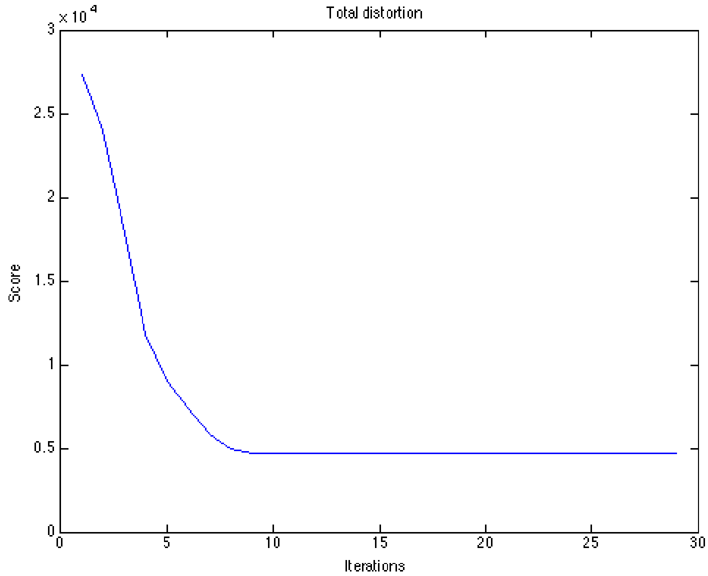

Figure 2: Total distortion

In this case we observed that there are slightly increase in total distortion scores from the 11$^{\text{th}}$ iteration which is not the case in k-means clustering since fuzzy c-means clustering embraces the fuzzy nature to consider a vector belongs simultaneously to more than one clusters and the objective function $J_q(\theta, U)$ contains the weight u_{ij} which balances the minimization.

$$J_q(\theta, U) = \sum_{i=1}^{N} \sum_{j=1}^{m} u_{ij}^q d(x_i, \theta_j) \tag{9}$$

5. In this problem, we are working **GKlines.dat** dataset to perform Gustafson-Kessel clustering and we will let it run for five iterations. The merit that Gustafson-Kessel clustering brings is the incorporation of covariance matrix to better capture the shape of the planar cluster that can overcome collinear distinct cluster clustering issue in fuzzy c-varieties (FCV) algorithm. The distance measure for Gustafson-Kessel algorithm is defined as:

$$d_{GK}^2(x, \theta_j) = |\Sigma_j|^{1/l} (x - c_j)^T \Sigma_j^{-1} (x - c_j) \tag{10}$$

By minimizing $J_{GK}(\theta, U) = \sum_{i=1}^{N} \sum_{j=1}^{m} u_{ij}^q d_{GK}^2(x_i, \theta_j)$ with respect to c_j and θ_j we obtain:

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^q x_i}{\sum_{i=1}^{N} u_{ij}^q x} \tag{11}$$

and

$$\Sigma_j = \frac{\sum_{i=1}^{N} u_{ij}^q (x - c_j)(x - c_j)^T}{\sum_{i=1}^{N} u_{ij}^q x} \tag{12}$$

5.2.b The estimated cluster representatives $\theta(5)$:

$$c^{(5)} = [c_1^{(5)} \quad c_2^{(5)}]; \tag{13}$$

$$= \begin{bmatrix} -0.9964 & 0.2988 \\ 0.6515 & 0.7042 \end{bmatrix} \tag{14}$$

$$\Sigma^{(5)} = [\Sigma_1^{(5)} \Sigma_2^{(5)}]; \tag{15}$$

$$= \begin{bmatrix} 0.0010 & 2.0179 \\ 0.0060 & -2.0293 \\ 2.0895 & 2.0426 \end{bmatrix} \tag{16}$$

Where, Σ_j is the upper triangular values for covariance matrix of the j^{th} cluster.

5.2.c By using the parameters estimated from the first iteration we assigned each sample to clusters using minimum distance rule.

$$j^* = \underset{j}{\operatorname{argmin}}\, d_{GK}^2(x_n, \theta_j(1)) \tag{17}$$

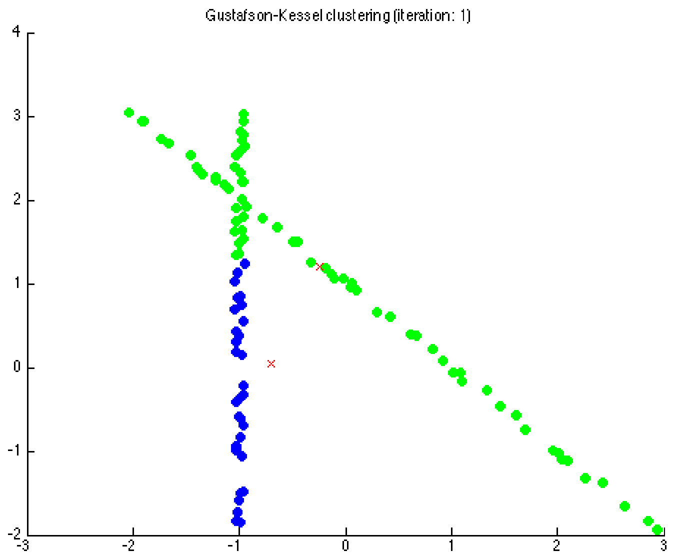

Figure 3: Plot of samples in first iteration

5.2.d By using the parameters estimated from the fifth iteration.

$$j^* = \underset{j}{\operatorname{argmin}}\, d_{GK}^2(x_n, \theta_j(5)) \tag{18}$$

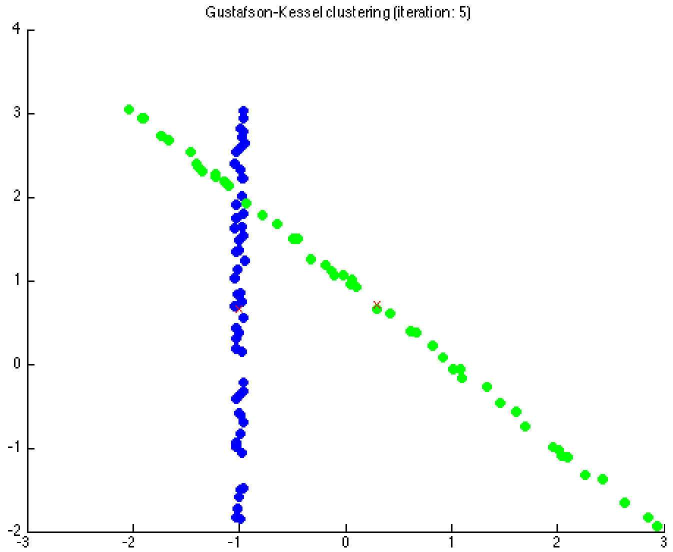

Figure 4: Plot of samples in first iteration

5.2.e Finally, we computed the total distance in G-K clustering for each iteration.

$$D(i) = \sum_{n=1}^{N} \min_j d^2_{GK}(x_n, \theta_j(i)) \tag{19}$$

i	1	2	3	4	5
$D(i)$	107.2321	69.4622	38.1877	15.0506	9.3709

Appendix:

assignment_3.m

```
clc;
clear all;
close all;

problem_3
problem_4
problem_5
```

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

```
% load data
X = load('../1/GMD.dat');

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

[~,l] = size(X);
% initialization
Theta = rand(l,m);
display('Random initialization:');
fprintf('Theta(0) = \n\n');
disp(Theta);

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

% result
display('Result:');
fprintf('Theta(%d) = \n\n', length(distortion));
disp(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');
```

problem_5.m

```
% load data
X = load('GKlines.dat');

% design parameters
% number of cluster
m = 2;
% number of iterations
T = 5;
% fuzzifier
q = 2;

% initialization
c = [[-1 0]' [0 1]'];
Sigma = [sigma2vec(eye(2)) sigma2vec(eye(2))];

% Gustafson-Kessel clustering
[C, SIGMA, total_distance] = gustafson_kessel(X, c, Sigma, q, T);

% result
fprintf('c(5) = \n\n');
disp(C{T});
fprintf('Sigma(5) = \n\n');
disp(SIGMA{T});
display(total_distance);

% plot data samples
color = 'bg';
for t=[1 5]
    c = C{t};
    I = gk_cluster_assignment(X, C{t}, SIGMA{t});
    figure; hold on;
    for k=unique(I)'
        Ck = X(I==k,:);
        scatter(Ck(:,1), Ck(:,2), 'filled', color(k));
    end
    plot(c(1,:), c(2,:), 'rx');
    hold off;
    title(['Gustafson-Kessel clustering (iteration: ' num2str(t) ')']);
end
```

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

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

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

while true
    U = zeros(N, m);
    for i=1:N
        D = point_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,l)) .* X) / denominator(j);
    end

    % check for termination
    % : if change in Theta is smaller than epsilon
    c = Theta(:) - Theta_t(:);
    if sqrt(c'*c) < epsilon
        break;
    end
    Theta = Theta_t;
    % total distortion
    distortion(t) = total_distortion(X, Theta);
end
```

point_distance.m

```
function [ D ] = point_distance( x, c )
% distance - Compute distance from x to set of points c
%            d = (x-c)'A(x-c) by assuming that A = I

[l,m] = size(c);
A = eye(l);
D = zeros(m,1);

for j=1:m
    x_tilde = x - c(:,j);
    D(j) = x_tilde'*A*x_tilde;
end
```

total_distortion.m

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

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

```
distance = zeros(N,m);

for i=1:N
    x = X(i,:)';
    distance(i,:) = point_distance(x, Theta);
end
distortion = sum( min(distance, [], 2) );
```

fcm_cluster_assignment.m

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

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

for i=1:N
    x = X(i,:)';
    distance(i,:) = point_distance(x, Theta);
end
[~,I] = min(distance, [], 2);
```

sigma2vec.m

```
function [ v ] = sigma2vec( S )
% sigma2vec - Vectorized form of covariance matrix

l = length(S);
v = zeros(l*(l+1)/2, 1);
ind = 1;

for i=1:l
    for j=i:l
        v(ind) = S(i, j);
        ind = ind + 1;
    end
end
```

vec2sigma.m

```
function [ S ] = vec2sigma( v, l )
% vec2sigma - Convert vector to l*l covariance matrix

if l*(l+1)/2 ~= length(v)
    error('The required elements mismatch with the dimensionalty.');
end

S = zeros(l, l);
ind = 1;
for i=1:l
    for j=i:l
        S(i, j) = v(ind);
        ind = ind + 1;
    end
end
% clone upper-triangle off-diagonal to the lower
S = S + triu(S, 1)';
```

gustafson_kessel.m

```
function [ C, SIGMA, total_distance ] = gustafson_kessel( X, c, Sigma, q, T )
% gustafson_kessel - Run Gustafson-Kessel (G-K) clustering algorithm
%
%         X       - Dataset
%         C       - Cluster centers
%         Sigma   - Cluster covariances
%         q       - Fuzzifier
%         T       - Number of iterations

[N,l] = size(X);
[~,m] = size(c);
p = 1/(q-1);
```

```
C = cell(1,T);
SIGMA = cell(1,T);
total_distance = zeros(1,T);

for t=1:T
    U = zeros(N, m);
    for i=1:N
        D = gk_distance(X(i,:)', c, Sigma);
        for j=1:m
            U(i,j) = 1/(sum( (D(j)./D).^p ));
        end
    end
    % parameter update
    denominator = sum(U.^q);
    for j=1:m
         c(:,j) = sum((U(:,j).^q*ones(1,l)) .* X) / denominator(j);
         X_tilde = X' - c(:,j)*ones(1,N);
         S = (ones(l,1)*U(:,j)'.^q) .* X_tilde * X_tilde' ./ denominator(j);
         Sigma(:,j) = sigma2vec(S);
    end
    total_distance(t) = gk_total_distance(X, c, Sigma);
    C{t} = c;
    SIGMA{t} = Sigma;
end
```

gk_distance.m

```
function [ D ] = gk_distance( x, c, Sigma )
% distance - Compute gustafson-kessel distance d^2_GK(x, \Theta_j)

[l,m] = size(c);
D = zeros(m,1);

for j=1:m
    x_tilde = x - c(:,j);
    S = vec2sigma(Sigma(:,j), l);
    D(j) = det(S)^(1/l) * x_tilde'/S*x_tilde;
end
```

gk_total_distance.m

```
function [ total_distance ] = gk_total_distance( X, c, Sigma )
% gk_total_distance - Compute gustafson-kessel total distance

[N,~] = size(X);
[~,m] = size(c);
distance = zeros(N,m);

for i=1:N
    x = X(i,:)';
    distance(i,:) = gk_distance(x, c, Sigma);
end
total_distance = sum( min(distance, [], 2) );
```

gk_cluster_assignment.m

```
function [ I ] = gk_cluster_assignment( X, c, Sigma )
% gk_cluster_assignment - Gustafson-Kessel cluster assignment

[N,~] = size(X);
[~,m] = size(c);
distance = zeros(N,m);

for i=1:N
    x = X(i,:)';
    distance(i,:) = gk_distance(x, c, Sigma);
end
[~, I] = min(distance, [], 2);
```