Aim: Study the PSO clustering approach studied in the class. Analyze the code of the algorithm given at: https://github.com/iralabdisco/pso-clustering/blob/master/pso_clustering.m

The Fisher's Iris dataset comes along with MATLAB installation and can be found at : C:\Program Files\MATLAB\R2023b\toolbox\stats\statsdata

Q1. Update the code to find optimized clusters for Iris dataset using PSO clustering. Hybrid PSO and K-means method is not to be used.

Q2. Utilize the PSO clustering to solve any other clustering problem. Any available and suitable dataset can be used.

Ans 1)

Code

```
clear;
close all;

%rng('default') % For reproducibility

% INIT PARTICLE SWARM
centroids = 2;          % == clusters here (aka centroids)
dimensions = 2;         % how many dimensions in each centroid
particles = 20;         % how many particles in the swarm, aka how many solutions
iterations = 50;        % iterations of the optimization alg.
simtime=0.01;           % simulation delay btw each iteration
dataset_subset = 2;     % for the IRIS dataset, change this value from 0 to 2
write_video = false;    % enable to grab the output picture and save a video

% VIDEO GRUB STUFF...
if write_video
    writerObj = VideoWriter('PSO.avi');
    writerObj.Quality=100;
%     writerObj.FrameRate=30;
    open(writerObj);
```

```
end

% LOAD DEFAULT CLUSTER (IRIS DATASET); USE WITH CARE!
load fisheriris.mat
meas = meas(:,1+dataset_subset:dimensions+dataset_subset); %RESIZE THE DATASET
WITH CURRENT DIMENSIONS; USE WITH CARE!
dataset_size = size (meas);

% GLOBAL PARAMETERS (the paper reports this values 0.72;1.49;1.49)
w  = 0.72; %INERTIA
c1 = 1.49; %COGNITIVE
c2 = 1.49; %SOCIAL

% PLOT STUFF... HANDLERS AND COLORS
pc = []; txt = [];
cluster_colors_vector = rand(particles, 3);

% PLOT DATASET
fh=figure(1);
hold on;
if dimensions == 3
   plot3(meas(:,1),meas(:,2),meas(:,3),'k*');
   view(3);
elseif dimensions == 2
   plot(meas(:,1),meas(:,2),'k*');
end

% PLOT STUFF .. SETTING UP AXIS IN THE FIGURE
axis equal;
axis(reshape([min(meas)-2; max(meas)+2],1,[]));
hold off;
```

```
% SETTING UP PSO DATA STRUCTURES
swarm_vel = rand(centroids,dimensions,particles)*0.1;
swarm_pos = rand(centroids,dimensions,particles);
swarm_best = zeros(centroids,dimensions);
c = zeros(dataset_size(1),particles);
ranges = max(meas)-min(meas); %%scale
swarm_pos = swarm_pos .* repmat(ranges,centroids,1,particles) +
repmat(min(meas),centroids,1,particles);
swarm_fitness(1:particles)=Inf;


for iteration=1:iterations

    %CALCULATE EUCLIDEAN DISTANCES TO ALL CENTROIDS
    distances=zeros(dataset_size(1),centroids,particles);
    for particle=1:particles
        for centroid=1:centroids
            distance=zeros(dataset_size(1),1);
            for data_vector=1:dataset_size(1)
                %meas(data_vector,:)
                distance(data_vector,1)=norm(swarm_pos(centroid,:,particle)-meas(data_vector,:));
            end
            distances(:,centroid,particle)=distance;
        end
    end

    %ASSIGN MEASURES with CLUSTERS
    for particle=1:particles
        [value, index] = min(distances(:,:,particle),[],2);
        c(:,particle) = index;
```

```
    end

    % PLOT STUFF... CLEAR HANDLERS
    delete(pc); delete(txt);
    pc = []; txt = [];

    % PLOT STUFF...
    hold on;
    for particle=1:particles
        for centroid=1:centroids
            if any(c(:,particle) == centroid)
                if dimensions == 3
                    pc = [pc
plot3(swarm_pos(centroid,1,particle),swarm_pos(centroid,2,particle),swarm_pos(centroid,3,p
article),'*','color',cluster_colors_vector(particle,:))];
                elseif dimensions == 2
                    pc = [pc
plot(swarm_pos(centroid,1,particle),swarm_pos(centroid,2,particle),'*','color',cluster_colors_
vector(particle,:))];
                end
            end
        end
    end
    set(pc,{'MarkerSize'},{12})
    hold off;

    %CALCULATE GLOBAL FITNESS and LOCAL FITNESS:=swarm_fitness
    average_fitness = zeros(particles,1);
    for particle=1:particles
        for centroid = 1 : centroids
            if any(c(:,particle) == centroid)
                local_fitness=mean(distances(c(:,particle)==centroid,centroid,particle));
```

```
            average_fitness(particle,1) = average_fitness(particle,1) + local_fitness;
        end
    end
    average_fitness(particle,1) = average_fitness(particle,1) / centroids;
    if (average_fitness(particle,1) < swarm_fitness(particle))
        swarm_fitness(particle) = average_fitness(particle,1);
        swarm_best(:,:,particle) = swarm_pos(:,:,particle);     %LOCAL BEST FITNESS
    end
end
[global_fitness, index] = min(swarm_fitness);       %GLOBAL BEST FITNESS
swarm_overall_pose = swarm_pos(:,:,index);          %GLOBAL BEST POSITION

% SOME INFO ON THE COMMAND WINDOW
fprintf('%3d. global fitness is %5.4f\n',iteration,global_fitness);
%uicontrol('Style','text','Position',[40 20 180 20],'String',sprintf('Actual fitness is: %5.4f',
global_fitness),'BackgroundColor',get(gcf,'Color'));
pause(simtime);

% VIDEO GRUB STUFF...
if write_video
    frame = getframe(fh);
    writeVideo(writerObj,frame);
end

% SAMPLE r1 AND r2 FROM UNIFORM DISTRIBUTION [0..1]
r1 = rand;
r2 = rand;

% UPDATE CLUSTER CENTROIDS
for particle=1:particles
    inertia = w * swarm_vel(:,:,particle);
```

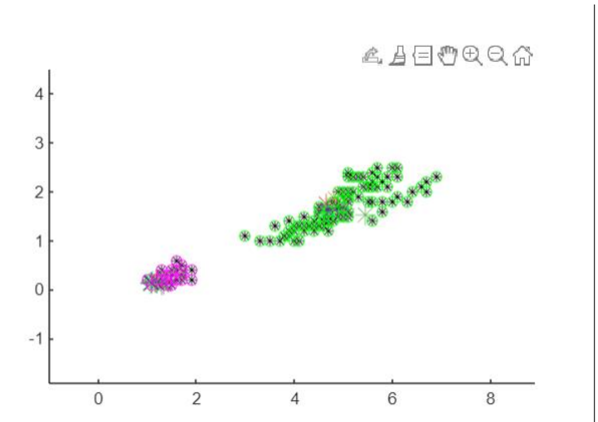
```
        cognitive = c1 * r1 * (swarm_best(:,:,particle)-swarm_pos(:,:,particle));
        social = c2 * r2 * (swarm_overall_pose-swarm_pos(:,:,particle));
        vel = inertia+cognitive+social;

        swarm_pos(:,:,particle) = swarm_pos(:,:,particle) + vel ;   % UPDATE PARTICLE
POSE
        swarm_vel(:,:,particle) = vel;                         % UPDATE PARTICLE VEL
    end

end



% PLOT THE ASSOCIATIONS WITH RESPECT TO THE CLUSTER
hold on;
particle=index; %select the best particle (with best fitness)
cluster_colors = ['m','g','y','b','r','c','g'];
for centroid=1:centroids
    if any(c(:,particle) == centroid)
        if dimensions == 3

plot3(meas(c(:,particle)==centroid,1),meas(c(:,particle)==centroid,2),meas(c(:,particle)==cen
troid,3),'o','color',cluster_colors(centroid));
        elseif dimensions == 2

plot(meas(c(:,particle)==centroid,1),meas(c(:,particle)==centroid,2),'o','color',cluster_colors(c
entroid));
        end
    end
end
hold off;

% VIDEO GRUB STUFF...
```

```
if write_video
    frame = getframe(fh);
    writeVideo(writerObj,frame);
    close(writerObj);
end

% SAY GOODBYE
fprintf('\nEnd, global fitness is %5.4f\n',global_fitness);
```

Output:

Command Window

```
 25. global fitness is 0.4970
 26. global fitness is 0.4970
 27. global fitness is 0.4970
 28. global fitness is 0.4970
 29. global fitness is 0.4958
 30. global fitness is 0.4933
 31. global fitness is 0.4933
 32. global fitness is 0.4933
 33. global fitness is 0.4894
 34. global fitness is 0.4894
 35. global fitness is 0.4894
 36. global fitness is 0.4894
 37. global fitness is 0.4894
 38. global fitness is 0.4887
 39. global fitness is 0.4873
 40. global fitness is 0.4863
 41. global fitness is 0.4863
 42. global fitness is 0.4863
 43. global fitness is 0.4862
 44. global fitness is 0.4862
 45. global fitness is 0.4862
 46. global fitness is 0.4861
 47. global fitness is 0.4861
 48. global fitness is 0.4860
 49. global fitness is 0.4860
 50. global fitness is 0.4860

End, global fitness is 0.4860
>>
```



Ans 2)

Dataset: Shopping Mall Customer Dataset

Code:

```
clc;

clear;

% Load the dataset (adjust the path as needed)

data = readtable('/MATLAB Drive/Dataset/Mall_Customers.csv', 'VariableNamingRule',
'preserve');


% Preprocess the dataset: Drop 'CustomerID', encode 'Gender', and extract relevant columns

data.Gender = grp2idx(data.Gender) - 1; % Convert Gender to 0 (Male) and 1 (Female)


% Extract the columns 'Gender', 'Age', 'Annual Income (k$)', 'Spending Score (1-100)'

processed_data = data{:, {'Gender', 'Age', 'Annual Income (k$)', 'Spending Score (1-100)'}};


% Standardize the data (zero mean and unit variance)

processed_data = (processed_data - mean(processed_data)) ./ std(processed_data);


% Perform PCA to reduce the data to 2 dimensions

[coeff, score, ~, ~, explained] = pca(processed_data);


% Print explained variance component-wise
```

```
fprintf('Explained Variance Component-wise:\n');
for i = 1:length(explained)
    fprintf('Component %d: %.2f%%\n', i, explained(i));
end

% Choose the number of principal components based on explained variance
pca_data = score(:, 1:2);  % Using only the first 2 components

% Initialize PSO parameters for clustering
centroids = 3;          % Number of clusters
dimensions = size(pca_data, 2); % Number of dimensions (after PCA)
particles = 20;         % Number of particles
iterations = 50;        % Number of iterations
w = 0.72;               % Inertia weight
c1 = 1.49;              % Cognitive component
c2 = 1.49;              % Social component

% Initialize particle positions and velocities
ranges = max(pca_data) - min(pca_data);
swarm_pos = rand(centroids, dimensions, particles) .* reshape(ranges, [1, dimensions, 1]) +
min(pca_data);
swarm_vel = rand(centroids, dimensions, particles) * 0.1;

swarm_fitness = inf(1, particles);
swarm_best = zeros(centroids, dimensions, particles);
dataset_size = size(pca_data, 1);
c = zeros(dataset_size, particles);

% PSO Clustering Loop
for iteration = 1:iterations
    % Calculate distances to centroids and assign clusters
```

```
    distances = zeros(dataset_size, centroids, particles);
    for particle = 1:particles
        for centroid = 1:centroids
            distances(:, centroid, particle) = sqrt(sum((pca_data - swarm_pos(centroid, :,
particle)).^2, 2));
        end
    end

    % Assign each point to the nearest centroid
    for particle = 1:particles
        [~, c(:, particle)] = min(distances(:, :, particle), [], 2);
    end

    % Calculate fitness (quantization error)
    average_fitness = zeros(1, particles);
    for particle = 1:particles
        for centroid = 1:centroids
            if any(c(:, particle) == centroid)
                local_fitness = mean(distances(c(:, particle) == centroid, centroid, particle));
                average_fitness(particle) = average_fitness(particle) + local_fitness;
            end
        end
        average_fitness(particle) = average_fitness(particle) / centroids;

        if average_fitness(particle) < swarm_fitness(particle)
            swarm_fitness(particle) = average_fitness(particle);
            swarm_best(:, :, particle) = swarm_pos(:, :, particle);
        end
    end

    % Get global best fitness and position
```

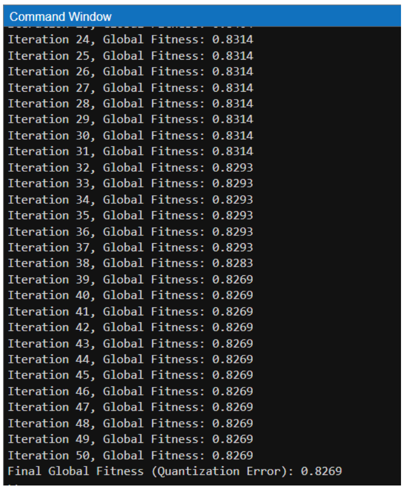
```
    [global_best_fitness, best_particle] = min(swarm_fitness);
    global_best_position = swarm_pos(:, :, best_particle);

    % Update particle positions and velocities
    r1 = rand;
    r2 = rand;
    for particle = 1:particles
        inertia = w * swarm_vel(:, :, particle);
        cognitive = c1 * r1 * (swarm_best(:, :, particle) - swarm_pos(:, :, particle));
        social = c2 * r2 * (global_best_position - swarm_pos(:, :, particle));
        swarm_vel(:, :, particle) = inertia + cognitive + social;
        swarm_pos(:, :, particle) = swarm_pos(:, :, particle) + swarm_vel(:, :, particle);
    end

    % Display iteration number and fitness
    fprintf('Iteration %d, Global Fitness: %.4f\n', iteration, global_best_fitness);
end

% Final quantization error
fprintf('Final Global Fitness (Quantization Error): %.4f\n', global_best_fitness);

% Plot final cluster assignment
figure;
hold on;
colors = ['r', 'g', 'b'];
for centroid = 1:centroids
    cluster_points = pca_data(c(:, best_particle) == centroid, :);
    scatter(cluster_points(:, 1), cluster_points(:, 2), [], colors(centroid), 'filled');
end
title('PSO Clustering Results After PCA');
```

xlabel('Principal Component 1');

ylabel('Principal Component 2');

hold off;

Output:



```
Command Window
Iteration 24, Global Fitness: 0.8314
Iteration 25, Global Fitness: 0.8314
Iteration 26, Global Fitness: 0.8314
Iteration 27, Global Fitness: 0.8314
Iteration 28, Global Fitness: 0.8314
Iteration 29, Global Fitness: 0.8314
Iteration 30, Global Fitness: 0.8314
Iteration 31, Global Fitness: 0.8314
Iteration 32, Global Fitness: 0.8293
Iteration 33, Global Fitness: 0.8293
Iteration 34, Global Fitness: 0.8293
Iteration 35, Global Fitness: 0.8293
Iteration 36, Global Fitness: 0.8293
Iteration 37, Global Fitness: 0.8293
Iteration 38, Global Fitness: 0.8283
Iteration 39, Global Fitness: 0.8269
Iteration 40, Global Fitness: 0.8269
Iteration 41, Global Fitness: 0.8269
Iteration 42, Global Fitness: 0.8269
Iteration 43, Global Fitness: 0.8269
Iteration 44, Global Fitness: 0.8269
Iteration 45, Global Fitness: 0.8269
Iteration 46, Global Fitness: 0.8269
Iteration 47, Global Fitness: 0.8269
Iteration 48, Global Fitness: 0.8269
Iteration 49, Global Fitness: 0.8269
Iteration 50, Global Fitness: 0.8269
Final Global Fitness (Quantization Error): 0.8269
```