ASSESSMENT 1: K-Means Clustering

COMP3003
Machine Learning
2022/2023

Abstract

Machine learning has many implementations, one of which is Unsupervised Learning, which can be used to find patterns within datasets. One such method is called Clustering, and this report demonstrates how the K-means clustering approach has been applied. In this experiment, limitations of this approach are discussed.

Table of Contents

Abstract 1
Introduction 2
Implementation 2
Discussions 12
References 13
Appendix 14
    main.m 14
    distance.m 16
    setcluster.m 17

Introduction

The concept of Machine Learning originated within the late 1950’s, with researchers becoming infatuated with the idea of recognition systems (Fradkov, 2020). Most models that were developed took a deterministic approach, wherein outputs are predictable; if repeatedly given a specific input, the model will always output the same result. However, in the 1960’s a probabilistic approach was suggested, and this concept of introducing randomness into models has resulted in modern day *Supervised* and *Unsupervised* machine learning algorithms.

Supervised learning algorithms have a component of human intervention – data being input into models is labelled. Another significant part is that the model is provided output data, called training data. The model trains input data against inferences made from the training data, resulting in ‘predicted’ outputs (IBM).

Unsupervised learning algorithms differ as the data is unlabelled and the model is not provided any targets or training data. Therefore, this paradigm has a lack of human intervention, which could be advantageous when trying to find specific solutions to a task – such as when trying to analyse the space landscapes (Cheng et al 2021). For unusual environments, human interaction can severely bias findings, such as: labels not being greatly understood. *Unsupervised learning* models are suited for analysing galaxies as there is not a lot of prior knowledge – it would be difficult to accurately label data, especially when its *Big Data* and consists of 10 years of telescopic data (Cheng et al 2021). Likewise, the level of expertise of the person(s) labelling the data needs to be considered. Overall, this paradigm is most often used when the common features within the dataset are unknown (IBM).

There are three different tasks that *Unsupervised learning* models can perform and each of these expose different Machine Learning concepts. These tasks are *clustering, association,* and *dimensionality reduction.* Each of these tasks takes a different approach and enables different analysis on the data. This report will focus on *clustering*.

Clustering is a “data mining” approach (IBM) wherein a dataset is partitioned into groups of similar data. *Clustering* is primarily used when there is a focus on the structure of the data – are there any patterns that can be identified? Common uses of *clustering* include (IBM):

- Market segmentation
- Image compression or segmentation
- Document clustering

One such implementation method is *K-Means clustering*. *K-means* is a popular method within *Unsupervised Machine Learning* because it’s very simple, so its quick and easy to implement. However, because of this simplicity it does suffer from some limitations, and these areexemplified within the **Implementation** and explored within the **Discussions** sections of this report.

Implementation

The K-means clustering algorithm is an iterative process wherein the algorithm repeatedly ‘trains’ itself until it reaches convergence, or its told to stop, whether this is by limiting the number of iterations or if it has been designed in a way wherein it can be manually stopped.

When working with small datasets or powerful supercomputers, stopping the algorithm may not be significant because a result will be produced quickly. However, when processing *big data* or if the host machine has limited computational power, then the question of when to end the algorithm becomes very significant as the algorithm could run for long amounts of time – which is generally not preferred.

The *K-means clustering* algorithm has been decomposed into steps below:

1. Define *K* .
2. Select *K* centroids randomly.
3. Calculate Euclidean distance between each data point and *K* centroid.
4. For each data point, cluster it to the centroid with the shortest distance.
5. Recalculate centroids.
6. Loop back to step 3 until convergence, then terminate.

Alternatively, Figure 1 offers a visual demonstration of the process, whilst better displaying how the algorithm is iterating.

The algorithm starts by defining *K* – which represents the number of centroids. Fundamentally, *K* is the number of clusters that the algorithm specifies and creates. Therefore, it's helpful to know prior how many clusters are truly within the dataset, and K can be set to the optimal value, otherwise the algorithm could possibly create more clusters than are represented, or perhaps make fewer but bigger clusters. This is because the K-means algorithm assigns every single data point a centroid. This limitation will be demonstrated further within the report.

It's worth noting that when the algorithm reaches *Step 6*, it checks for convergence, and if the centroids have not converged, the algorithm loops back to *Step 3*.

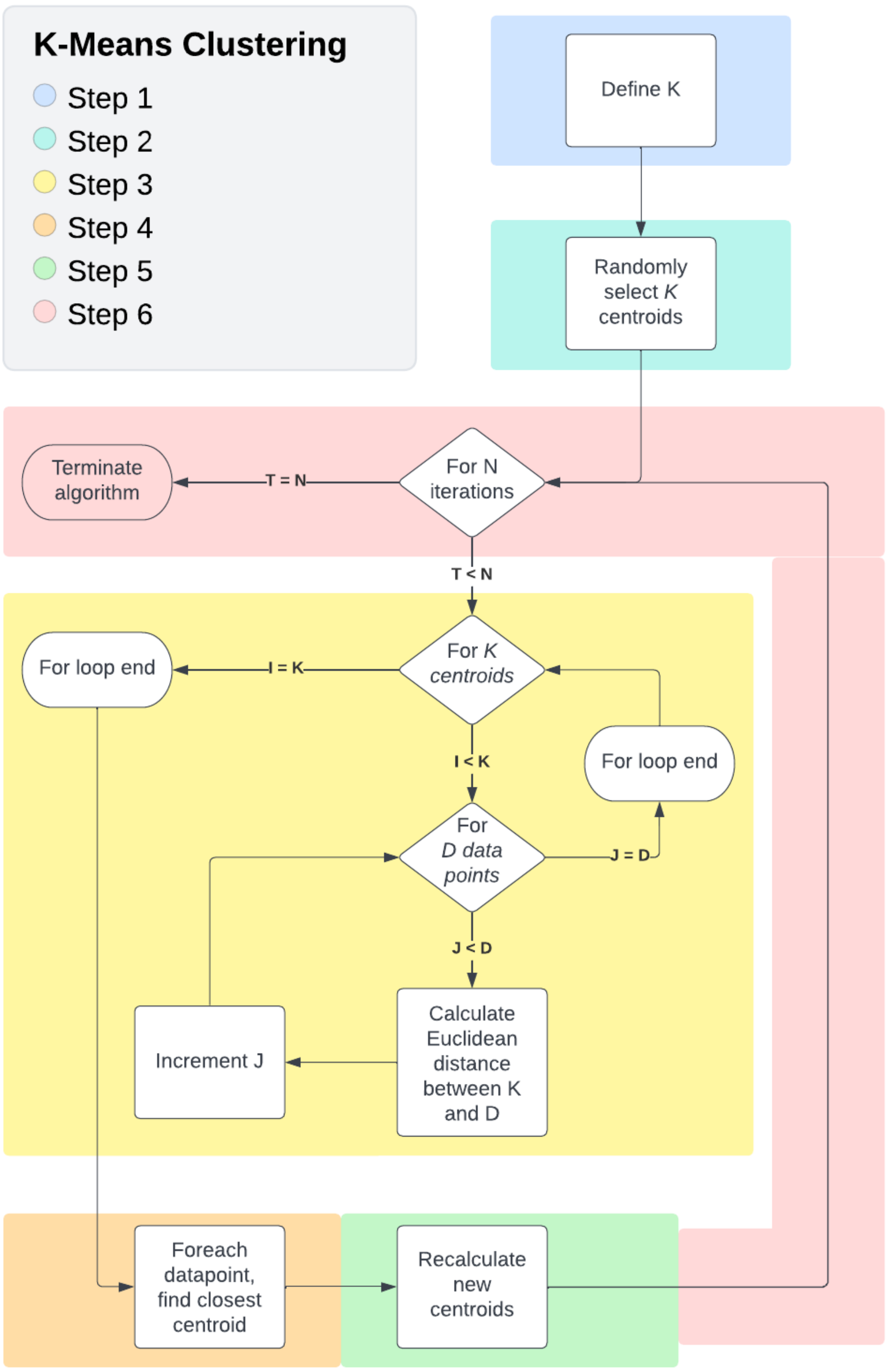


Figure 1. K-Means clustering algorithm design in flowchart.

Before starting to implement the *K-means* algorithm, a dataset had to be created. To fully demonstrate the limitations of *K-means clustering*, three clusters were created with random, uncorrelated 2-dimensional data – [X, Y].

```
 1
 2    % Mean and standard deviation for each cluster
 3    m1 = [-4;-2];   std1 = 0.75;
 4    m2 = [1;4];     std2 = 2.0;
 5    m3 = [-2;10];   std3 = 0.3;
 6
 7    N = 1000;   % Sample size (for each cluster)
 8
 9    d1 = std1 * randn(2, N) + repmat(m1, 1, N); % 2D Gausssian cluster 1
10    d2 = std2 * randn(2, N) + repmat(m2, 1, N); % 2D Gausssian cluster 2
11    d3 = std3 * randn(2, N) + repmat(m3, 1, N); % 2D Gausssian cluster 3
12
13    data = [d1 d2 d3]; % Merge the three cluster classes into one dataset
14    sz = size(data, 2); % Find how many coord pairs are within dataset
```

Lines 3 – 5 set the mean and standard deviation of each cluster. For this experiment, it was chosen that there would be a very sparse cluster (line 4) surrounded by a very dense cluster (line 5) and a slightly sparse cluster (line 3). This consideration has been made so that the limitation of K-means' hard clustering and the effect this has on outliers can be visualised.

Lines 9 – 11 create the three 2D Gaussian clusters using random numbers. These clusters are then concatenated to create one merged dataset (see Figure 2.1 and 2.2).

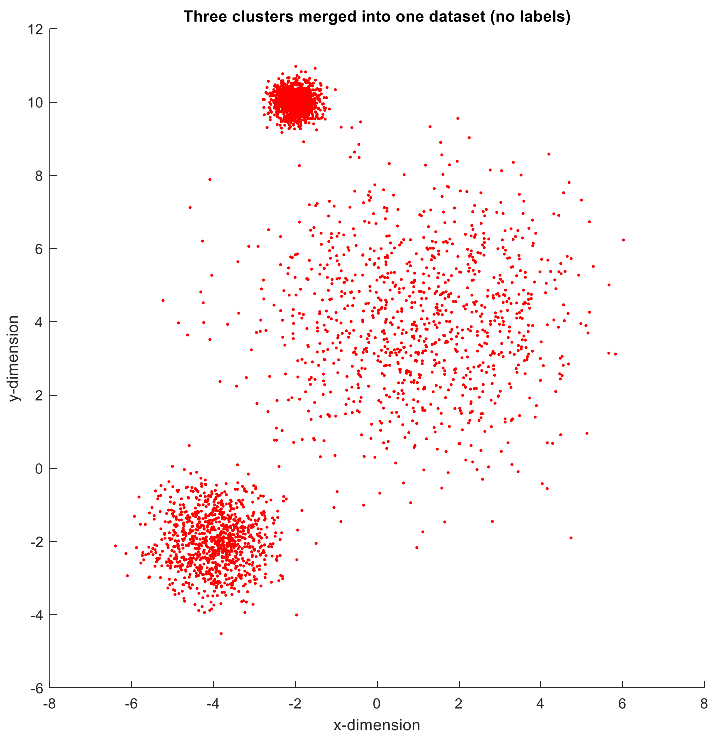


Figure 2.1, Dataset after merging

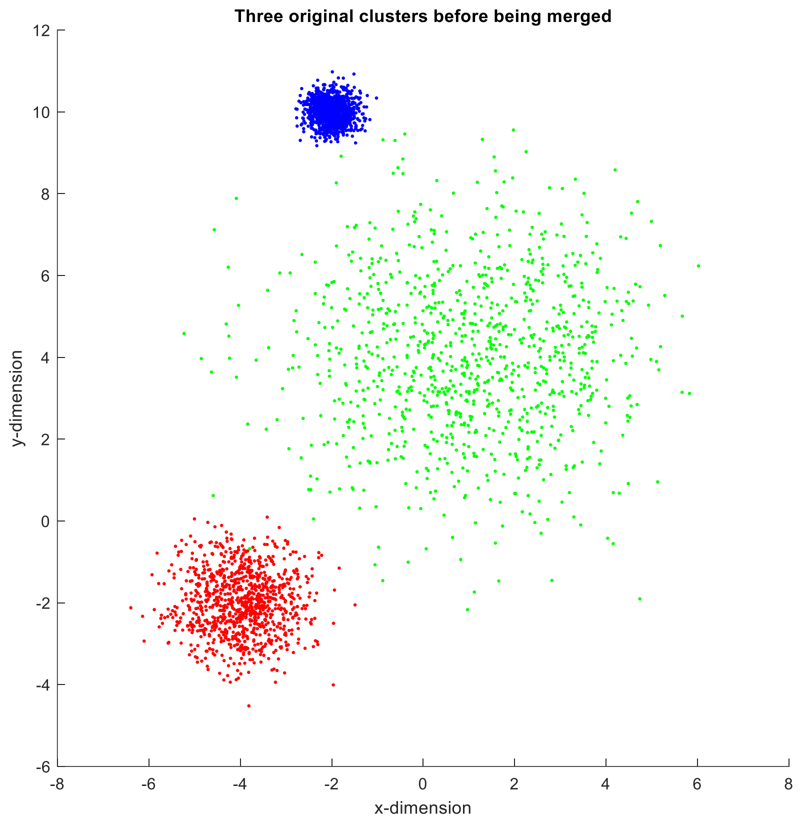
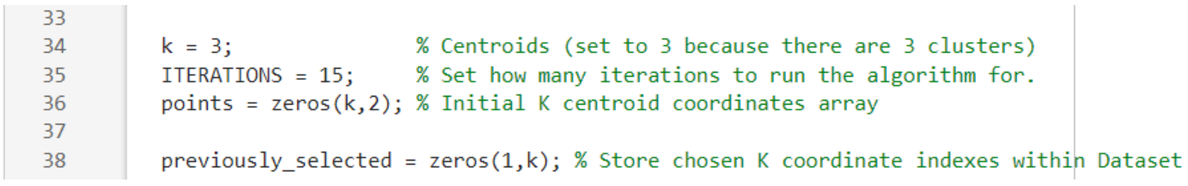


Figure 2.2, Dataset before merging

Now that the dataset is created, the next action the task performs is to define *k.* As explained beforehand, in this experiment, the number of clusters is already known so its optimal to set k to 3 on line 34.



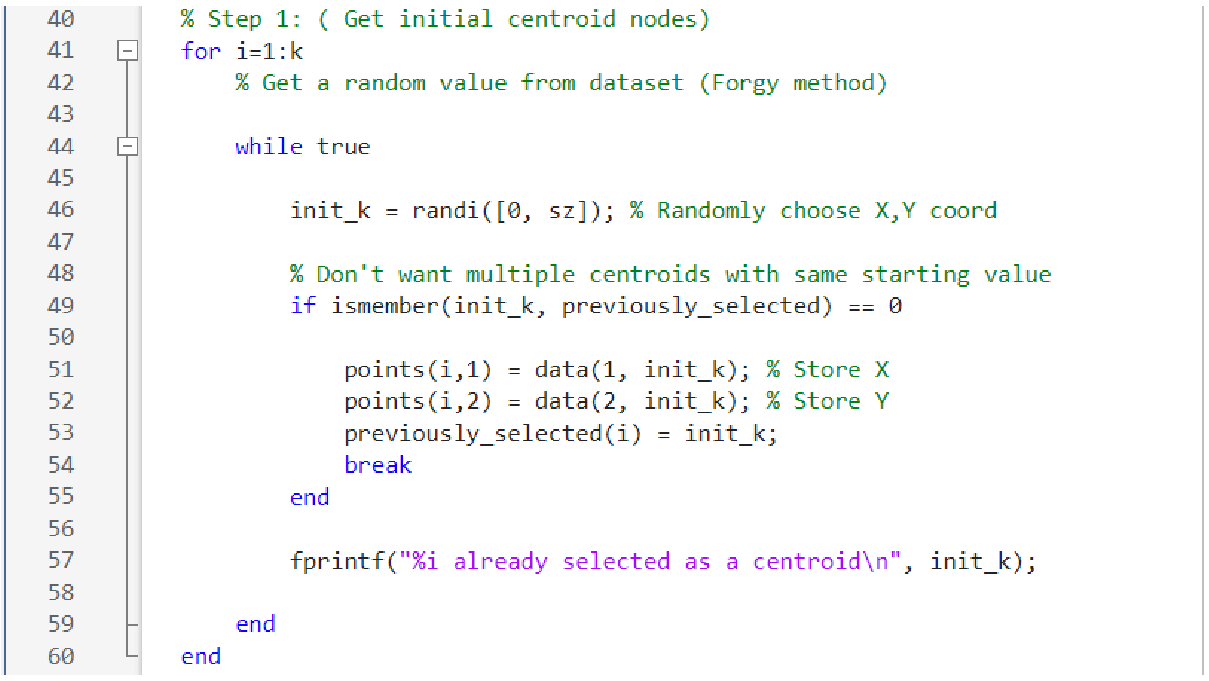
```
33
34    k = 3;              % Centroids (set to 3 because there are 3 clusters)
35    ITERATIONS = 15;    % Set how many iterations to run the algorithm for.
36    points = zeros(k,2); % Initial K centroid coordinates array
37
38    previously_selected = zeros(1,k); % Store chosen K coordinate indexes within Dataset
```

Line 35 sets how many iterations to perform until the algorithm terminates. Line 36 creates a 2D array which is used to store the centroid coordinates.

Line 38s initialises an array. As this implementation uses the Forgy method to find the first *K* centroid values, there is the risk that the same data value within the dataset could be chosen for multiple K clusters. This array is used to implement unique K values.

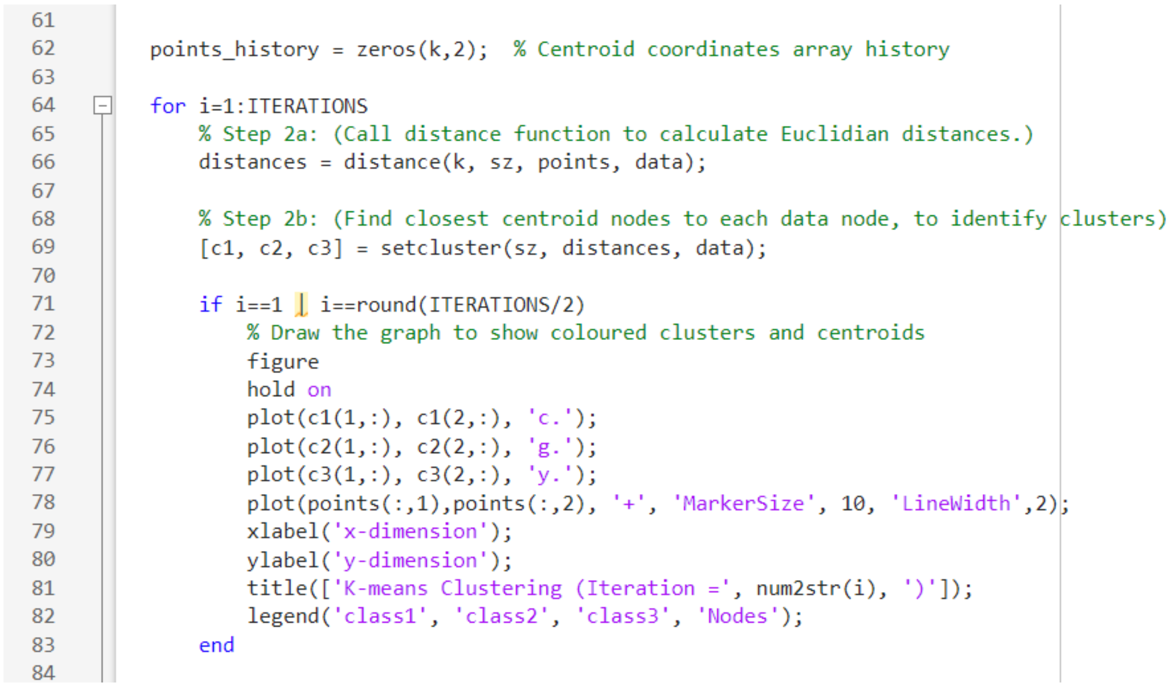
As seen in the code below, Line 49 checks to see if the randomly generated index of the dataset already exists (which means that a centroid for that data point already exists).

Line 46 implements the Forgy method as a data point from the dataset is randomly chosen, and within lines 51 and 52, the [X,Y] values are copied across to the previously initialized points array.



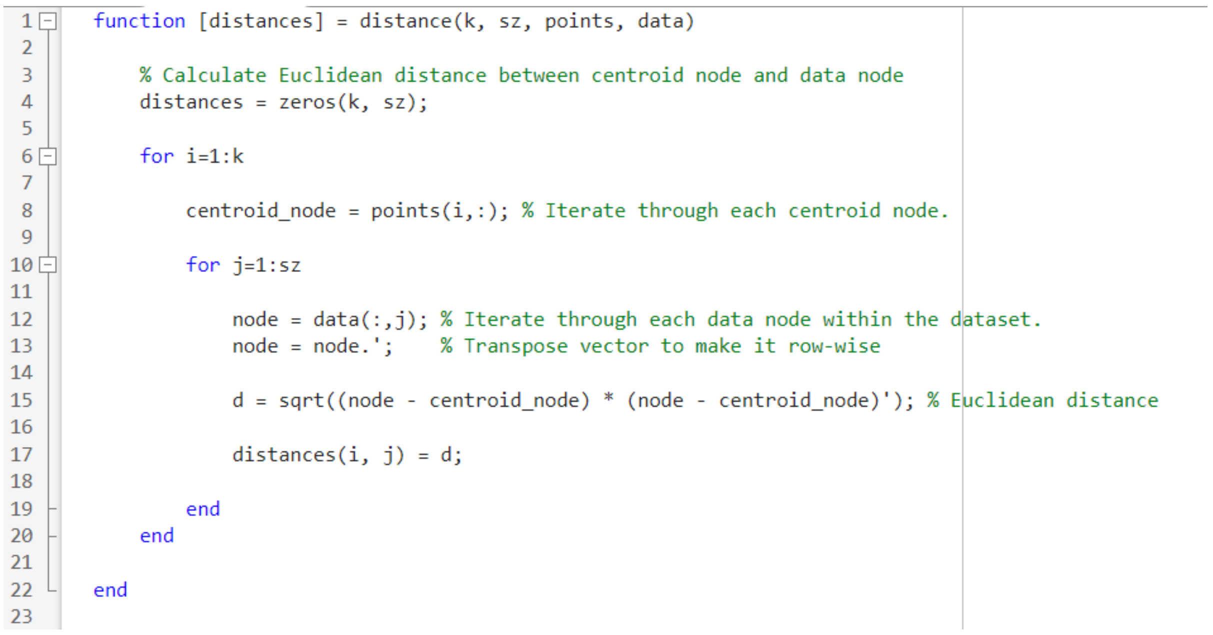
```
40  % Step 1: ( Get initial centroid nodes)
41  for i=1:k
42      % Get a random value from dataset (Forgy method)
43  
44      while true
45  
46          init_k = randi([0, sz]); % Randomly choose X,Y coord
47  
48          % Don't want multiple centroids with same starting value
49          if ismember(init_k, previously_selected) == 0
50  
51              points(i,1) = data(1, init_k); % Store X
52              points(i,2) = data(2, init_k); % Store Y
53              previously_selected(i) = init_k;
54              break
55          end
56  
57          fprintf("%i already selected as a centroid\n", init_k);
58  
59      end
60  end
```

Line 62 creates an array that stores the previous iteration's centroid values. These values are compared to the newly created centroids (Step 5) to see if the algorithm has converged, upon which the algorithm terminates (Step 6).



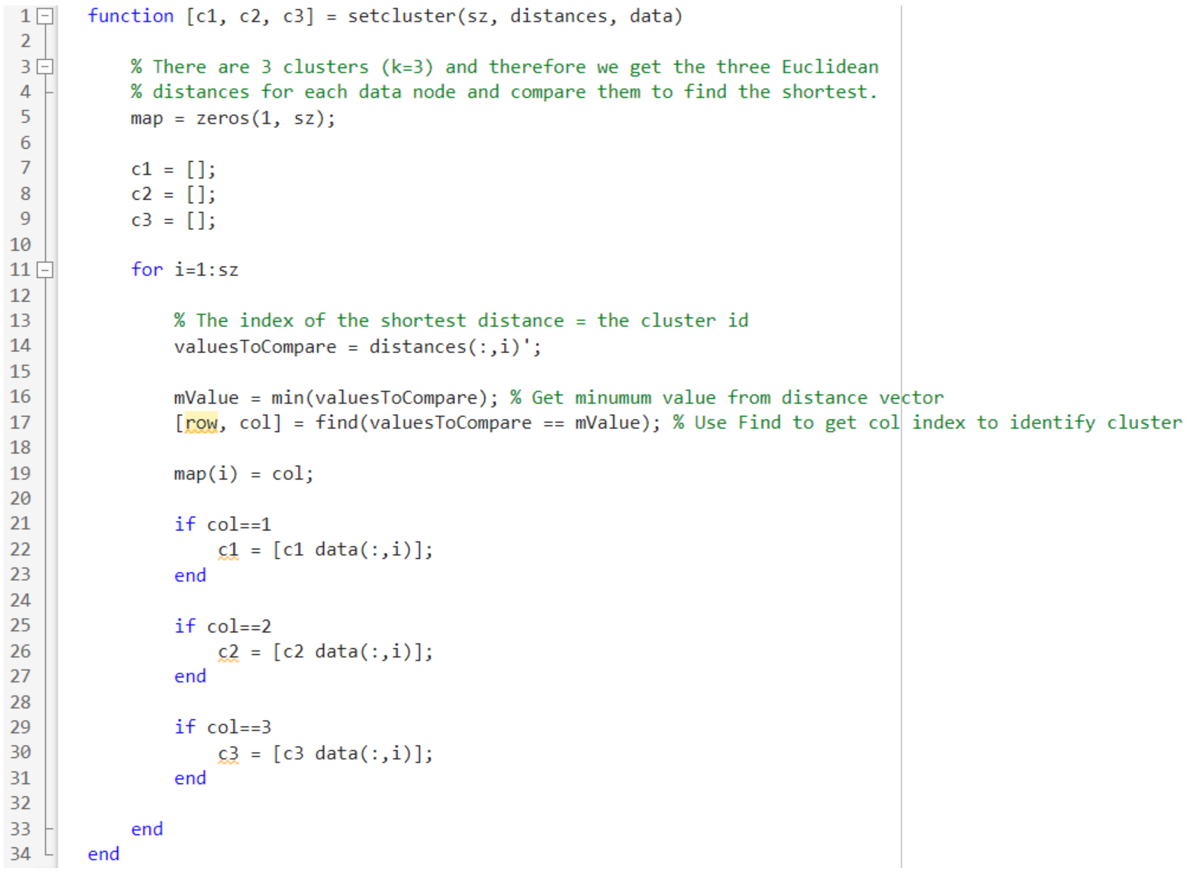
```
61  
62  points_history = zeros(k,2);  % Centroid coordinates array history
63  
64  for i=1:ITERATIONS
65      % Step 2a: (Call distance function to calculate Euclidian distances.)
66      distances = distance(k, sz, points, data);
67  
68      % Step 2b: (Find closest centroid nodes to each data node, to identify clusters)
69      [c1, c2, c3] = setcluster(sz, distances, data);
70  
71      if i==1 | i==round(ITERATIONS/2)
72          % Draw the graph to show coloured clusters and centroids
73          figure
74          hold on
75          plot(c1(1,:), c1(2,:), 'c.');
76          plot(c2(1,:), c2(2,:), 'g.');
77          plot(c3(1,:), c3(2,:), 'y.');
78          plot(points(:,1),points(:,2), '+', 'MarkerSize', 10, 'LineWidth',2);
79          xlabel('x-dimension');
80          ylabel('y-dimension');
81          title(['K-means Clustering (Iteration =', num2str(i), ')']);
82          legend('class1', 'class2', 'class3', 'Nodes');
83      end
84  
```

Line 64 is the outermost loop and runs for a specified number of ITERATIONS. Within this image, steps 3 and 4 are enacted – Euclidean distance is calculated, and clusters are formed. Separate function files are used to simplify the code.



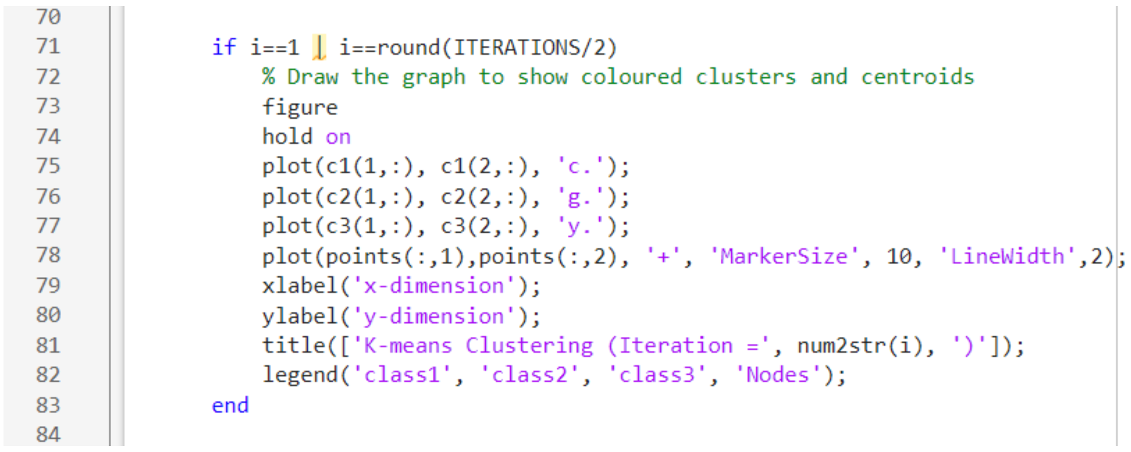
```
 1  function [distances] = distance(k, sz, points, data)
 2  
 3      % Calculate Euclidean distance between centroid node and data node
 4      distances = zeros(k, sz);
 5  
 6      for i=1:k
 7  
 8          centroid_node = points(i,:); % Iterate through each centroid node.
 9  
10          for j=1:sz
11  
12              node = data(:,j); % Iterate through each data node within the dataset.
13              node = node.';    % Transpose vector to make it row-wise
14  
15              d = sqrt((node - centroid_node) * (node - centroid_node)'); % Euclidean distance
16  
17              distances(i, j) = d;
18  
19          end
20      end
21  
22  end
23  
```

For each centroid, the function loops through all data points and calculates the Euclidean distance (line 15). These distances are stored within an array of k*sz size, where k is the number of centroids, and sz represents the total number of data points. One of the greatest flaws of K-means is that this method of implementation is computationally intensive and for Big Data would be very inefficient and slow.



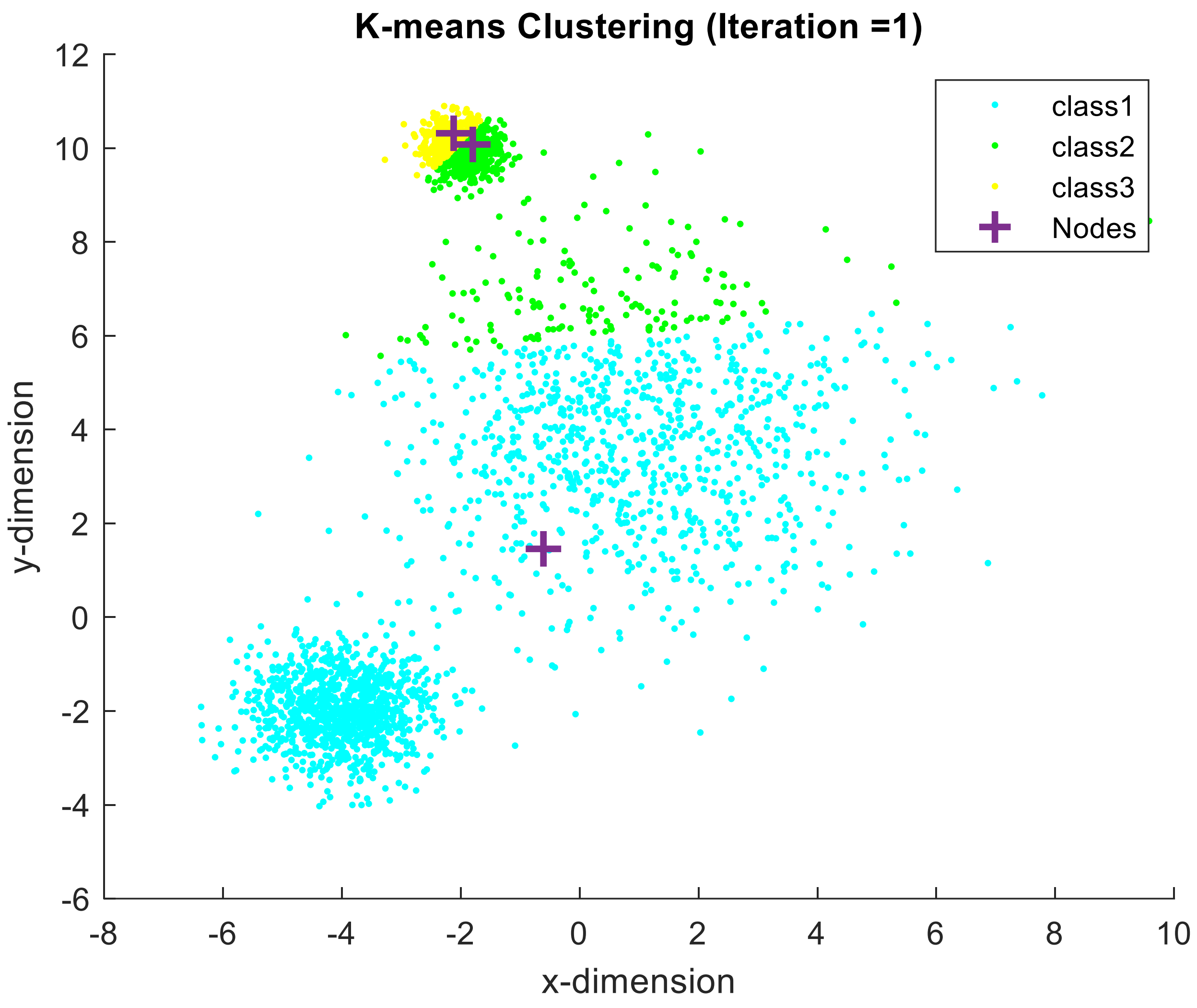
```
function [c1, c2, c3] = setcluster(sz, distances, data)

    % There are 3 clusters (k=3) and therefore we get the three Euclidean
    % distances for each data node and compare them to find the shortest.
    map = zeros(1, sz);

    c1 = [];
    c2 = [];
    c3 = [];

    for i=1:sz

        % The index of the shortest distance = the cluster id
        valuesToCompare = distances(:,i)';

        mValue = min(valuesToCompare); % Get minumum value from distance vector
        [row, col] = find(valuesToCompare == mValue); % Use Find to get col index to identify cluster

        map(i) = col;

        if col==1
            c1 = [c1 data(:,i)];
        end

        if col==2
            c2 = [c2 data(:,i)];
        end

        if col==3
            c3 = [c3 data(:,i)];
        end

    end
end
```

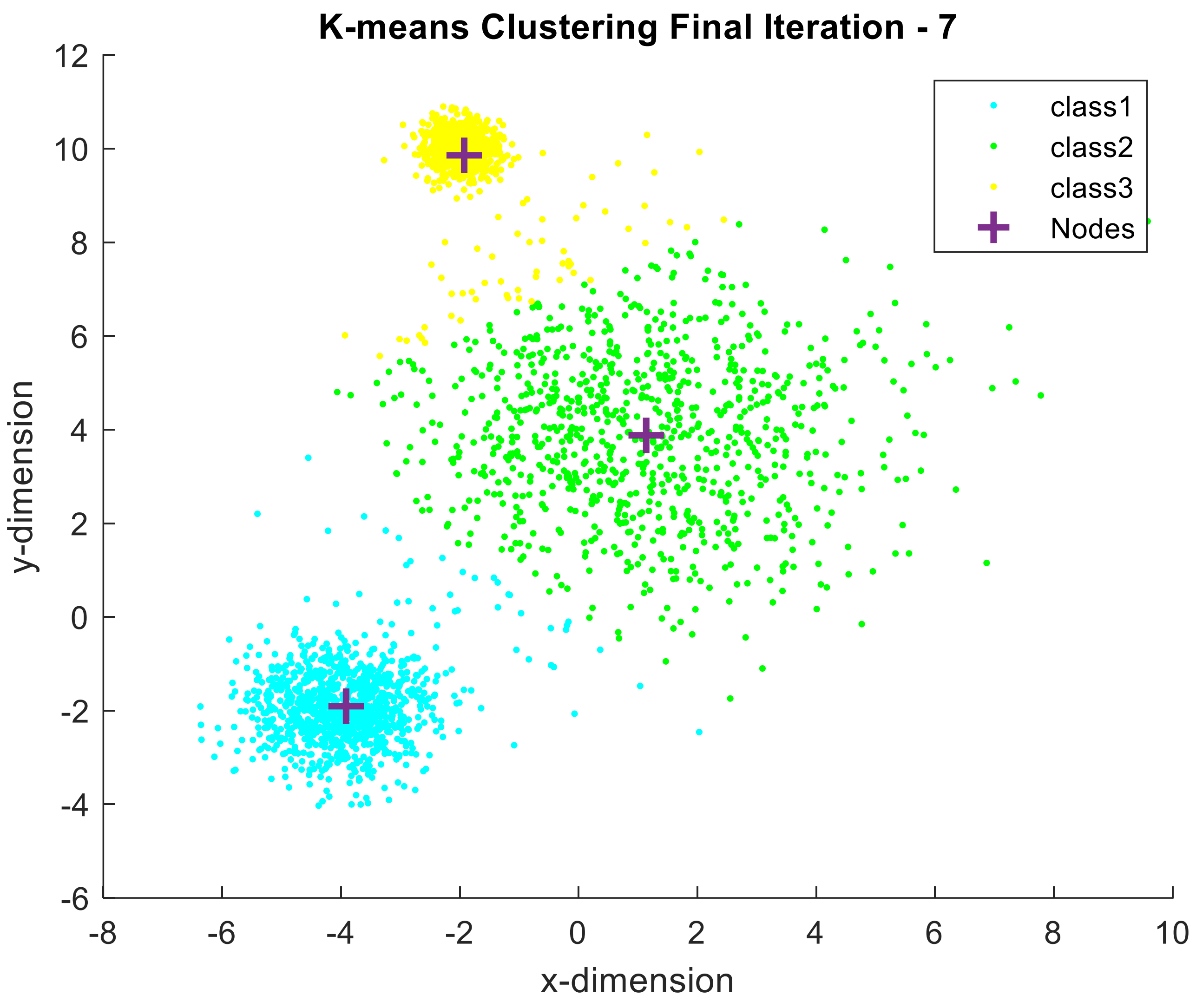
This function takes all the Euclidean distances as an argument, and for each data point determines which centroid is the closest separating them into 3 arrays for each three clusters.



```

        if i==1 | i==round(ITERATIONS/2)
            % Draw the graph to show coloured clusters and centroids
            figure
            hold on
            plot(c1(1,:), c1(2,:), 'c.');
            plot(c2(1,:), c2(2,:), 'g.');
            plot(c3(1,:), c3(2,:), 'y.');
            plot(points(:,1),points(:,2), '+', 'MarkerSize', 10, 'LineWidth',2);
            xlabel('x-dimension');
            ylabel('y-dimension');
            title(['K-means Clustering (Iteration =', num2str(i), ')']);
            legend('class1', 'class2', 'class3', 'Nodes');
        end

```

Graphs are drawn to the screen if the algorithm is on its first iteration, or if it’s halfway through the maximum number of iterations set. Matlab will not output graphs abundantly, but the changes within clusters can still be seen. A final graph is drawn using the same code, but outside of the i:ITERATIONS loop.

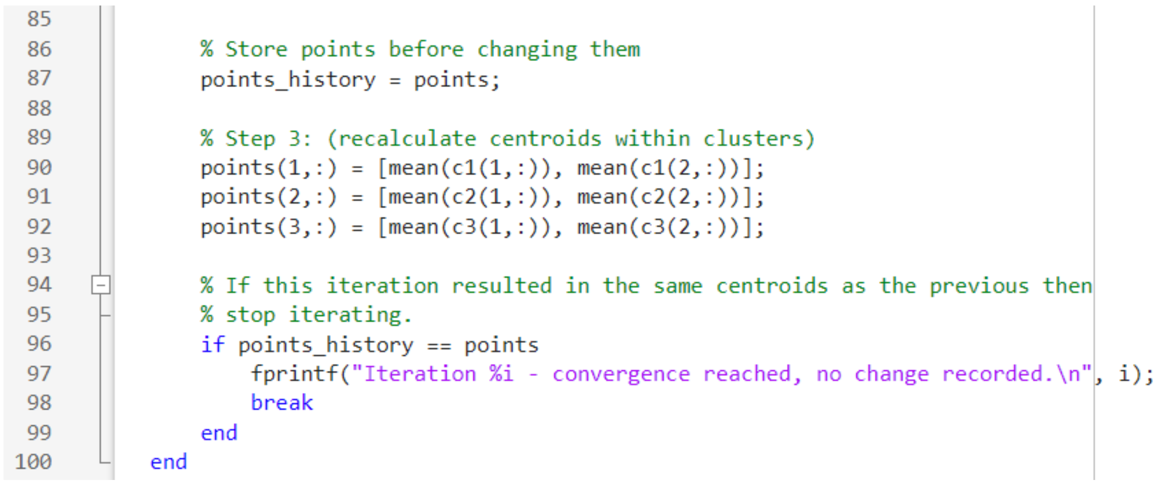

K-means Clustering (Iteration =1)
12
10
8
6
4
2
0
-2
-4
-6
-8
-6
-4
-2
0
2
4
6
8
10
y-dimension
x-dimension
class1
class2
class3
Nodes



As the last graph is labelled 'Final Iteration', it can be concluded that convergence occurred at this iteration – a graph for ITERATIONS/2 has not been produced.

Upon analysing the difference between the graphs, initially the centroids were very close, but then spread out and formed near accurate clusters. However, the struggle to accurately identify outliers, which is noticeable when juxtaposed with Figure 2.2. For example, the yellow cluster has obviously claimed some of the green cluster's original data points. Overall, this graph shows that K-means is "non-robust to outliers" (BISHOP, 2016). Another factor to consider is that with this algorithm, although each cluster was made with a set number of samples, the K-means algorithm does not consider this, and therefore that's why the blue and yellow clusters expand.

Finally, the code ends with storing the centroid points before calculating the new ones. This enables for convergence to be identified, resulting in the i:ITERATIONS loop to be forcibly terminated (lines 96 – 99). Lines 90 to 92 recalculate the centroids by calculating the mean of all data points X values, and then Y values. These new centroid values are stored within the points array, and the program loops.



```

        % Store points before changing them
        points_history = points;

        % Step 3: (recalculate centroids within clusters)
        points(1,:) = [mean(c1(1,:)), mean(c1(2,:))];
        points(2,:) = [mean(c2(1,:)), mean(c2(2,:))];
        points(3,:) = [mean(c3(1,:)), mean(c3(2,:))];

        % If this iteration resulted in the same centroids as the previous then
        % stop iterating.
        if points_history == points
            fprintf("Iteration %i - convergence reached, no change recorded.\n", i);
            break
        end
    end
```

The console within the MATLAB IDE displays the following:

```
>> main
Iteration 7 - convergence reached, no change recorded.
>>
```

Discussions

Through implementing the *K-means* design from first principles and creating overlapping clusters, a large limitation of the algorithm has been exposed:

1. Outliers can be incorrectly clustered.
2. Clusters can expand or shrink in size.

These limitations are largely due to the inability for clusters to overlap – this is called hard clustering, wherein each data point belongs to exactly one centroid. In a practical use, this limitation would be very inefficient - overlaps between two categories might not be identified as belonging to both groups

There are three other limitations of *K-means* that with configuration, this experiment could show:

1. Increasing the amount of data point samples, greatly exposes the inefficiency of calculating Euclidean distances between each data point and each K centroid.
2. Demonstrate a result stuck in local minimum.
3. If the most optimal K value is not known, then it must be guessed.

References

BISHOP, C.H.R.I.S.T.O.P.H.E.R.M. (2016) Pattern recognition and machine learning. SPRINGER-VERLAG NEW YORK.

By: IBM Cloud Education (no date) What is supervised learning?, IBM. Available at: https://www.ibm.com/cloud/learn/supervised-learning

By:IBM Cloud Education (no date) What is unsupervised learning?, IBM. Available at: https://www.ibm.com/cloud/learn/unsupervised-learning

Cheng, T.-Y. et al. (2021) “Beyond the Hubble sequence – exploring galaxy morphology with unsupervised machine learning,” Monthly Notices of the Royal Astronomical Society, 503(3), pp. 4446–4465. Available at: https://doi.org/10.1093/mnras/stab734.

Fradkov, A.L. (2020) “Early history of machine learning,” IFAC-PapersOnLine, 53(2), pp. 1385–1390. Available at: https://doi.org/10.1016/j.ifacol.2020.12.1888.

Appendix

main.m

```
% Mean and standard deviation for each cluster
m1 = [-4;-2];   std1 = 0.75;
m2 = [1;4];     std2 = 2.0;
m3 = [-2;10];   std3 = 0.3;

N = 1000;   % Sample size (for each cluster)

d1 = std1 * randn(2, N) + repmat(m1, 1, N); % 2D Gausssian cluster 1
d2 = std2 * randn(2, N) + repmat(m2, 1, N); % 2D Gausssian cluster 2
d3 = std3 * randn(2, N) + repmat(m3, 1, N); % 2D Gausssian cluster 3

data = [d1 d2 d3]; % Merge the three cluster classes into one dataset
sz = size(data, 2); % Find how many coord pairs are within dataset

% Plot the data with colours (unmerged)
figure
hold on
plot(d1(1,:), d1(2,:), 'r.');
plot(d2(1,:), d2(2,:), 'g.');
plot(d3(1,:), d3(2,:), 'b.');
xlabel('x-dimension');
ylabel('y-dimension');
title('Three original clusters before being merged');

% Plot the data with colours (unmerged)
figure
hold on
plot(data(1,:), data(2,:), 'r.');
xlabel('x-dimension');
ylabel('y-dimension');
title('Three clusters merged into one dataset (no labels)');

k = 3;               % Centroids (set to 3 because there are 3 clusters)
ITERATIONS = 15;     % Set how many iterations to run the algorithm for.
points = zeros(k,2); % Initial K centroid coordinates array

previously_selected = zeros(1,k); % Store chosen K coordinate indexes within
Dataset

% Step 1: ( Get initial centroid nodes)
for i=1:k
    % Get a random value from dataset (Forgy method)

    while true

        init_k = randi([0, sz]); % Randomly choose X,Y coord

        % Don't want multiple centroids with same starting value
        if ismember(init_k, previously_selected) == 0

            points(i,1) = data(1, init_k); % Store X
            points(i,2) = data(2, init_k); % Store Y
            previously_selected(i) = init_k;
            break
        end

        fprintf("%i already selected as a centroid\n", init_k);

    end
end

points_history = zeros(k,2);  % Centroid coordinates array history

for i=1:ITERATIONS
    % Step 2a: (Call distance function to calculate Euclidian distances.)
    distances = distance(k, sz, points, data);

    % Step 2b: (Find closest centroid nodes to each data node, to identify
clusters)
    [c1, c2, c3] = setcluster(sz, distances, data);

    if i==1 | i==round(ITERATIONS/2)
        % Draw the graph to show coloured clusters and centroids
        figure
        hold on
        plot(c1(1,:), c1(2,:), 'c.');
        plot(c2(1,:), c2(2,:), 'g.');
        plot(c3(1,:), c3(2,:), 'y.');
        plot(points(:,1),points(:,2), '+', 'MarkerSize', 10, 'LineWidth',2);
        xlabel('x-dimension');
        ylabel('y-dimension');
        title(['K-means Clustering (Iteration =', num2str(i), ')']);
        legend('class1', 'class2', 'class3', 'Nodes');
    end

    % Store points before changing them
    points_history = points;

    % Step 3: (recalculate centroids within clusters)
    points(1,:) = [mean(c1(1,:)), mean(c1(2,:))];
    points(2,:) = [mean(c2(1,:)), mean(c2(2,:))];
    points(3,:) = [mean(c3(1,:)), mean(c3(2,:))];

    % If this iteration resulted in the same centroids as the previous then
    % stop iterating.
    if points_history == points
        fprintf("Iteration %i - convergence reached, no change recorded.\n", i);
        break
    end
end

% Final optimized graph
% Will always output to show converged result
figure
hold on
plot(c1(1,:), c1(2,:), 'c.');
plot(c2(1,:), c2(2,:), 'g.');
plot(c3(1,:), c3(2,:), 'y.');
plot(points(:,1),points(:,2), '+', 'MarkerSize', 10, 'LineWidth',2);
xlabel('x-dimension');
ylabel('y-dimension');
title(['K-means Clustering Final Iteration - ', num2str(i) ]);
legend('class1', 'class2', 'class3', 'Nodes');
```

distance.m

```
function [distances] = distance(k, sz, points, data)

    % Calculate Euclidean distance between centroid node and data node
    distances = zeros(k, sz);

    for i=1:k

        centroid_node = points(i,:); % Iterate through each centroid node.

        for j=1:sz

            node = data(:,j); % Iterate through each data node within the dataset.
            node = node.';    % Transpose vector to make it row-wise

            d = sqrt((node - centroid_node) * (node - centroid_node)'); %
Euclidean distance

            distances(i, j) = d;

        end
    end

end
```

setcluster.m

```
function [c1, c2, c3] = setcluster(sz, distances, data)

    % There are 3 clusters (k=3) and therefore we get the three Euclidean
    % distances for each data node and compare them to find the shortest.
    map = zeros(1, sz);

    c1 = [];
    c2 = [];
    c3 = [];

    for i=1:sz

        % The index of the shortest distance = the cluster id
        valuesToCompare = distances(:,i)';

        mValue = min(valuesToCompare); % Get minumum value from distance vector
        [row, col] = find(valuesToCompare == mValue); % Use Find to get col index
to identify cluster

        map(i) = col;

        if col==1
            c1 = [c1 data(:,i)];
        end

        if col==2
            c2 = [c2 data(:,i)];
        end

        if col==3
            c3 = [c3 data(:,i)];
        end

    end
end
```