# K-means clustering

K-means clustering is a very well-known method of clustering unlabeled data. The simplicity of the process made it popular to data analysts. The task is to form clusters of similar data objects (points, properties etc.). When the dataset given is unlabeled, we try to make some conclusion about the data by forming clusters. Now, the number of clusters can be pre-determined and number of points can have any range.

The main idea behind the process is finding nearest cluster mean and assigning points to their nearest clusters. Initially we start by picking some random centroids (mean values of required clusters). Then we assign all points to some cluster. After assigning all the points, we calculate the cluster mean (centroid) and update it. Now, we have got new centroids of our clusters and all the points labelled to some cluster. Then, we have to again iterate over all points and re-assign the clusters considering the newly updated centroids. And, thus it goes on until all the points labelled does not change their clusters or at least a large (predetermined) number of assignment have been done.

The total algorithm can be summarized as:

1. Randomly pick *n* centroids (*n* being the number of clusters to be created)
2. At each iteration calculate the distance between each point and the centroids
3. Assign the point to its closest cluster (closest centroid in a cluster)
4. After assigning all the points to some cluster, update the centroids
5. Repeat steps 2-4 until there is no update in labelling or a fixed number of iterations completed

You are given with the **kmeansCluster** function:

```
function [] = kmeansCluster()
    max_range = 10;
    min_range = -10;
    num_points = 10000;
    vector = ( max_range - min_range ) .* rand( 1, num_points ) + min_range;
    points = makePoints( vector );
    centroids = initialCentroid( points );
    init_labels =  false( size( points, 1 ), 1 );
    [ labels, c1, c2 ] = makeClusters( points, centroids );
    trial = 0;
    while ~isequal( labels, init_labels ) && trial < 1000
        init_labels = labels;
        centroids = [ mean( c1 ); mean( c2 ) ];
        [ labels, c1, c2 ] = makeClusters( points, centroids );
        trial = trial + 1;
    end
    hold on
    plot( c1( :, 1 ), c1( :, 2 ), 'r.' );
    plot( c2( :, 1 ), c2( :, 2 ), 'b.' );
    plot( centroids(1), centroids(3), 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'r' );
    plot( centroids(2), centroids(4), 'bo', 'MarkerSize', 10, 'MarkerFaceColor', 'b' );
    hold off
end
```

Now, you have to use several functions in this function, they will have to be defined by you. All the functions are described below:

- **makePoints**
  - parameter:
    - *vector* - row vector with *m* points
  - returns:
    - *points* - *m/2 x 2* matrix of which each row represents a point
  - description: A row vector with *m* points will be passed to the function as parameter, you have to reshape the vector to a matrix with *m/2* rows and *2* columns, then return the newly modified matrix.
  - example:
    - `>> vector = [ 2 4 5 10 5 9 21 23 10 22 ];`
    - `>> points = makePoints( vector );`

```
points =

     2     4
     5    10
     5     9
    21    23
    10    22
```

- **initialCentroid**
  - parameter:
    - *point* - *n x 2* matrix with *n* point coordinates
  - returns:
    - *centroids* - *2 x 2* matrix with two random centroids at each row
  - description: The function takes in points and generates two random centroids ranging within range of the input points. For this, you need to find out the maximum and minimum values for x, y across all the points and generate random integer numbers using that range. The points input will be floating numbers, so you have to round them to nearest integers to be used as range for **randi** function.
  - example:
    - `>> centroids = initialCentroid( points );`

```
centroids =

     9    17
    15     9
```

- **makeClusters**
  - parameter:
    - *points* - matrix (*n x 2* matrix)
    - *means* - matrix (*2 x 2* matrix)
  - returns:
    - *label* – *n x 1* logical matrix containing 0's and 1's as labels for $i^{th}$ point
    - *cluster1, cluster2* – matrices with points assigned to that cluster, each row represents points
  - description: This function takes each point from *points* matrix and computes its distance from the two centroids. If the distance to centroid1 is lesser than the distance to centroid2,

it assigned 0 and concatenated vertically with cluster1. Otherwise, it is assigned as 1 and concatenated vertically with cluster2. To compute distance between two points, you also have to define **euclidDist** function which is described later.

- example:
  - `>> [ labels, cluster1, cluster2 ] = makeClusters( points, centroids );`
  - `labels =`

```
5×1 logical array

   1
   0
   0
   0
   0
```

  - `cluster1 =`

```
     5    10
     5     9
    21    23
    10    22
```

  - `cluster2 =`

```
     2     4
```

- **euclidDist**
  - parameter:
    - *point1, point2 – 1 x 2* matrices containing coordinates of points, x in column 1, y in column 2
  - returns:
    - *distance* – Euclidean distance between the points
  - description: This function computes the Euclidean distance between two points using the age-old formula $distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
    You can do this using Matlab shorthand command:

```
distance = sqrt( sum( ( point1 - point2 ) .^ 2 ))
```

  - example:
    - `>> point1 = [ 2, 2 ];`
    - `>> point2 = [ 5, 5 ];`
    - `>> distance = euclidDist( point1, point2 );`
    - `distance =`

```
    4.2426
```

You do not have to worry about other functions used here in the **kmeansCluster** function. What we did was initially created a matrix *label* which has zeros as labels.

```
init_labels =  false( size( points, 1 ), 1 );
```

Then we called the **makeCluster** function for the first time outside the while loop to initially label all the points.

```
[ labels, c1, c2 ] = makeClusters( points, centroids );
```

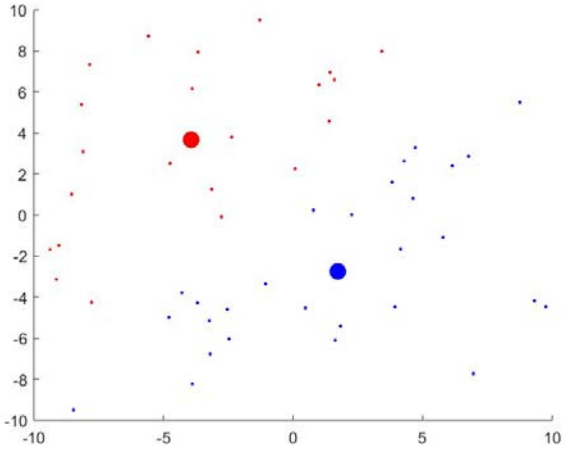
Now, the while loop will run until the labels assigned at each iteration remain same or trials are less than 1000. That is, there will be at most 1000 iterations if points get reassigned every iteration, or otherwise it would stop when there is no other change to labels.

```
while ~isequal( labels, init_labels ) && trial < 1000
```

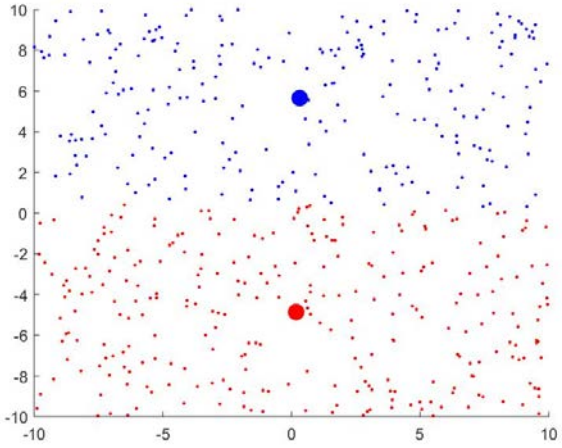
In the while loop, we are calculating new centroids and trying to reassign each point based on those newly updated centroids. Also, trial value is incremented.

```
init_labels = labels;
centroids = [ mean( c1 ); mean( c2 ) ];
[ labels, c1, c2 ] = makeClusters( points, centroids );
trial = trial + 1;
```

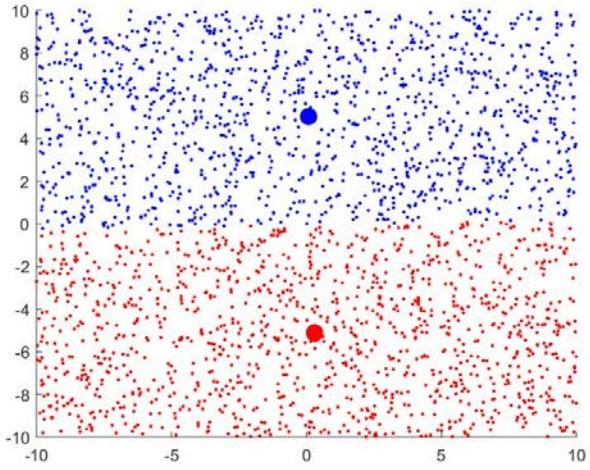
The next lines are for plotting as you can guess obviously. If you complete the functions correctly, you would be seeing a plot of the points clustered into two clusters, red and blue with centroids in solid circles. For different values of *num_points* you will see something like these graphs, though they might not be exactly same since we used random functions to generate our points.



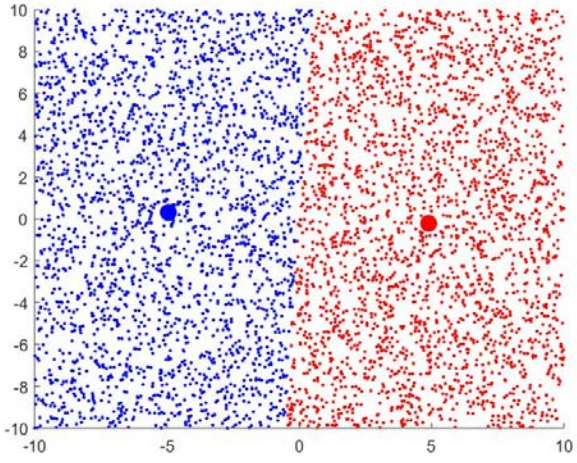
*num_points = 100*



*num_points = 1000*



*num_points = 5000*



*num_points = 10000*

Some resourceful links:

- http://web.stanford.edu/class/ee103/visualizations/kmeans/kmeans.html
- http://www.bytemuse.com/post/k-means-clustering-visualization/
- https://www.naftaliharris.com/blog/visualizing-k-means-clustering/