1. K-means

1.1. Practical implement the code

1.1.1 Implementing "Our" k-means

We developed the function kmeans_cw.m which is returning a matrix containing the cluster centres appended with another matrix which is returning cluster allocation in so called R matrix. Our code is the following:

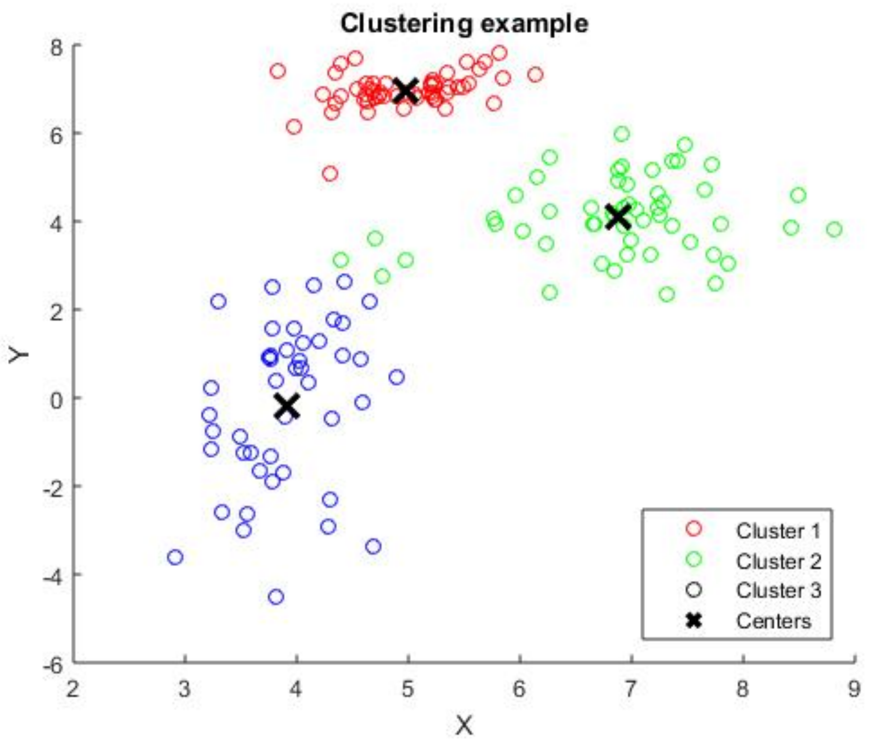
```
function [ clusters ] = kmeans_cw( data, k )
   d = size(data);
   %l x n
   l = d([1]);
   n = d([2]);
   r = zeros(l,k); % l x k matrix
   for j = 1:k
      r(randi(l,1,1),j) = 1; %assign 1 at random in each column
   end
   clusters = data' * r; % l x k
   clusters_old=clusters;
   isittrue=true;
   while (isittrue)
   for i = 1:l
         dist = 2*sum(max(data).^2)*5; %set upper bound
         x = data(i,:);

         for j = 1:k
            c = clusters(:,j);
            new_dist = (x' - c).^2;

            if(sum(new_dist) < dist)
               dist = sum(new_dist);
               idx = j;
            end
         end
        r(i,idx) = 1;
      end
     clusters = (data'*r) * (diag(1./sum(r)));
      clusters_old;
      if(sum((clusters - clusters_old).^2) ==0)
         isittrue = false;
         clusters = [clusters; r];
      end
      clusters_old = clusters;
      r = zeros(l,k); %reset
   end
end
```

1.1.2. "Our" k-means testing and visualizing, efficiency computing

Using our algorithm we developed the following clustering, plotted below.



We calculated occe 100 times and obtained the following results:

$$mean(occe) = \quad 0.0839$$

$$sd(occe) = 0.1378$$

Note: these values are subject to change due to the random nature of initial cluster allocation.

1.1.3. IRIS dataset

We did this question using firstly kmean_cw.m and after that using kmeans Matlab function.

During the first run we obtained:

$$mean(occe) = \ 0.1952$$

$$sd(occe) = 0.1321$$

Note: these values are subject to change due to the random nature of initial cluster allocation.

During the second run we obtained:

$$mean(occe) = \ 0.0116$$

$$sd(occe) = 0.0045$$

Unsurprisingly, Matlab function is better than ours, although the results obtained with kmeans_cw.m are satisfactory.

1.2. Questions

1.2.1. Proof of equation (3)

Let's assume that the length of $\mathbf{x_i}$ is n. Then for

$$\mathbf{c}_j = \frac{\sum_{i=1}^{l} r_{ij}\mathbf{x_i}}{\sum_{i=1}^{l} r_{ij}}$$

We can recognise that $\sum_{i=1}^{l} r_{ij}\mathbf{x_i}$ will return a vector with length n, where the k-th entry (for $1 \leq k \leq n$) will be the sum of the k-th dimension of all $\mathbf{x_i}$s that are in the j-th cluster. $\sum_{i=1}^{l} r_{ij}$ is the sum of all elements in the j-th cluster. The ratio between the two will give us a vector corresponding to Centroid for the given cluster.

The sum that we would like to minimize with respect to the j-th cluster is the following:

$$\sum_{i=1}^{l} r_{ij}(\mathbf{x}_\mathrm{i} - \bar{\mathbf{x}}_{\boldsymbol{j}})^2$$

Since r_{ij} is an indicator function and $\bar{\mathbf{x}}_{\boldsymbol{j}}$ is the minimizing new j-th cluster centre. The first and the second derivative of this function with respect to $\bar{\mathbf{x}}_{\boldsymbol{j}}$ are the following:

$$\frac{d}{d\,\bar{\mathbf{x}}_{\boldsymbol{j}}} \sum_{i=1}^{l} r_{ij}(2\bar{\mathbf{x}}_{\boldsymbol{j}} - 2\mathbf{x}_\mathrm{i})$$

$$\frac{d^2}{d\,\bar{\mathbf{x}}_{\boldsymbol{j}}^{\,2}} \sum_{i=1}^{l} r_{ij}\, 2 \geq 0$$

Since the second derivative is always positive, the minimum is reached for

$$\frac{d}{d\,\bar{\mathbf{x}}_{\boldsymbol{j}}} \sum_{i=1}^{l} r_{ij}(2\bar{\mathbf{x}}_{\boldsymbol{j}} - 2\mathbf{x}_\mathrm{i}) = 0$$

$$\bar{\mathbf{x}}_{\boldsymbol{j}} = \frac{\sum_{i=1}^{l} r_{ij}\mathbf{x_i}}{\sum_{i=1}^{l} r_{ij}}$$

Which is exactly the centroid, as required.

1.2.2. K-means convergence

To understand how K-means converges in a finite number of steps one must first analyse the inertia of such an algorithm where inertia is defined as a monotonically decreasing function:

$$\frac{i}{n}\sum_{j=1}^{K}\sum_{i=1}^{n}\|x_i - c_k\|^2 \qquad (1)$$

Equation 1 is proved by assuming $X^t$is the current position of the algorithm $X_1^t \ldots . X_K^t$. We can also denote $C_1^t \ldots . C_K^t$ as the centroids. From this point one can assign a function, let's call this G^t such that:

$$\emptyset(X^t) \geq \sum_{j=1}^{K}\sum_{x_i \in x_i^t} \left\| x_i, C^t{}_{G_{x_i}^{t+1}} \right\|^2$$

$$\geq \sum_{j=1}^{K}\sum_{x_i \in x_i^t} \left\| x_i, C_j^{t+1} \right\|^2$$

thus:

$$\emptyset(X^t) \geq \emptyset(X^{t+1})$$

There is only a finite quantity of partitions in which inertia strictly decreases. This is as a result of the number of patricians being finite such that:

$$\binom{n}{k}$$

Thus any sequence $\emptyset(X^t)_{t\in N}$ will indeed be characterised by a finite number of steps to convergence such that there will exist some t where:

$\emptyset(X^t) = \emptyset(X^{t+1})$
(2)

Therefore there must exist a point in the k means algorithm that $X^{t+1} = X^t$ and thus convergence has occurred, otherwise a wrong classification would occur.

It should be noted that this definition does not express the order of convergence in time the and only eludes to the fact that the upper bound is defined by $\binom{n}{k}$. Convergence is sensitive to initialisation which suggests computing the algorithm a many times and computing the mean value will allow an answer closer to the true behaviour of the algorithm to be presented.

Alternative 1:

Why it is not correct to use arbitary distances: because **k-means may stop converging with other distance functions**. The common proof of convergence is like this: the assignment step *and* the mean update step both optimize the *same* criterion. There is a finite number of assignments possible. Therefore, it must converge after a finite number of improvements. To use this proof for other distance functions, you must show that the *mean* (note: k-*means*) minimizes your distances, too.

The same is written in Wiki as well: https://en.wikipedia.org/wiki/K-means_clustering

1.3. Extensions

1.3.1 (p, k)-means

By substituting p with 2 and setting $sign(x_i)|x_i|{=}x_i$ we can recognise the squared Euclidean distances as required.

Let's define two diagonal matrices X_i and C_j, where $X_i = diag(\mathbf{x}_i) = diag(x_1, \dots, x_n)$ and $C_j = diag(c_1, \dots, c_k)$. Assume that all x_i's are positive and so the first iteration of cluster centers are consisting of positive elements as they are randomly selected across all $\mathbf{x}_i$s. For this reason we can define the distances between j-th cluster center and i-th observation as:

$$d_p(\mathbf{c}_j, \mathbf{x}_i) = tr(C_j{}^p) - tr(X_i{}^p) - p.tr\left((C_j - X_i).X_i{}^{p-1}\right)$$

Our rule for updating cluster centers ($\mathbf{c}_j$) should be based on the idea of minimising the distances between every element in the cluster and the new cluster center. In other words we want to minimize with respect to C_j the following expression:

$$\sum_{i=1}^{l} r_{ij} d_p(\mathbf{c}_j, \mathbf{x}_i)$$

Where r_{ij} is indicator function similar to (2) in the Coursework 2 discription. It is 1 if $\mathbf{x}_i$ is part of the j-th cluster and 0 otherwise, assuming that we put every observation to closest cluster center.

$$\frac{d}{dC_j}\sum_{i=1}^{l} r_{ij} d_p(\mathbf{c_j}, \mathbf{x_i}) = \frac{d}{dC_j}\sum_{i=1}^{l} r_{ij}\left(tr(C_j{}^p) - tr(X_i{}^p) - p.tr\left((C_j - X_i).X_i{}^{p-1}\right)\right)$$

$$= \sum_{i=1}^{l} r_{ij}\left(pC_j{}^{p-1} - pX_i{}^{p-1}\right)$$

Since $C_j = C_j{}^T$ and $X_i = X_i{}^T$ as diagonal matrices.

$$\frac{d^2}{dC_j{}^2}\sum_{i=1}^{l} r_{ij} d_p(\mathbf{c_j}, \mathbf{x_i}) = \frac{d}{dC_j}\sum_{i=1}^{l} r_{ij}\left(pC_j{}^{p-1} - pX_i{}^{p-1}\right) = \sum_{i=1}^{l} r_{ij}\left(p(p-1)C_j{}^{p-2}\right)$$

Since $r_{ij} = \{0,1\}, p \geq 1, p-1 \geq 0, C_j$ is a positive semidefinite then $\frac{d}{dC_j}\sum_{i=1}^{l} r_{ij} d_p(\mathbf{c_j}, \mathbf{x_i}) = 0$ will give us a minimum.

$$\frac{d}{dC_j}\sum_{i=1}^{l} r_{ij} d_p(\mathbf{c_j}, \mathbf{x_i}) = 0$$

$$\sum_{i=1}^{l} r_{ij}\left(pC_j{}^{p-1} - pX_i{}^{p-1}\right) = 0$$

$$\sum_{i=1}^{l} r_{ij} p C_j{}^{p-1} = \sum_{i=1}^{l} r_{ij} p X_i{}^{p-1}$$

$$C_j{}^{p-1}\sum_{i=1}^{l} r_{ij} = \sum_{i=1}^{l} r_{ij} X_i{}^{p-1}$$

$$C_j{}^{p-1} = \frac{\sum_{i=1}^{l} r_{ij} X_i{}^{p-1}}{\sum_{i=1}^{l} r_{ij}}$$

$$C_j = \sqrt[p-1]{\frac{\sum_{i=1}^{l} r_{ij} X_i{}^{p-1}}{\sum_{i=1}^{l} r_{ij}}}$$

Which will return a positive semidefinite i.e. we will always get positive $c_1, \ldots, c_n$.

In other words, for every $\mathbf{x_i}$ part of cluster j we can find the new cluster center elements as

$$c_1 = \sqrt[p-1]{\frac{\sum_{i=1}^{l} r_{i1} (x_1)_i{}^{p-1}}{\sum_{i=1}^{l} r_{i1}}}$$

$$\ldots$$

$$c_n = \sqrt[p-1]{\frac{\sum_{i=1}^{l} r_{in} (x_n)_i{}^{p-1}}{\sum_{i=1}^{l} r_{in}}}$$

IMPORTANT: http://www.psi.toronto.edu/matrix/calculus.html

We can argue that convergence is reached as there are finite number of cluster centers and every time we have constrain of minimising distances – similar to those in the first example.

1.3.2. k-means segmentation

K-means segmentations are an interpretation of K-means algorithm in terms of clustering ordered data. For this reason this method is also known as sequential or online K-means. The segmentation relies on the number of k chosen which dictates the number of clusters present (*or segmentations in this case*). The goal of this algorithm is to efficiently segment the data as it presents itself to the algorithm (*online classification task*) and minimises some error formulation (*see e.q. xxx*). Centroids and segmentation boundaries must be initiated, then the centroids must move towards the optimal position in line with the error condition. It is assumed the data is a vector $1\ x\ d$.

The proposed algorithm is as follows:

- Initiate boundaries of segmentations
    - This is simply splitting the data into k equal segments
- Initiate the centroids in the centre of each segment
    - This could be done in many ways such as random allocation, bias allocation or centre the points in each segment. Standardised k-means randomised the centroids, however in an online setting it has been decided to place the centroids in the centre of each segment to easily verify a movement towards the mean.
- Initiate centroid position storage matrix
- While (centroid position continues to update and thus continue to converge)[1]
    - For each segment:
        - Update the centroid's position by taking the mean of the data points in the respective segment.
        - Store position
    - If (centroid's position has not updated)
        - Break - optimal location has been discovered
- End

The proposed algorithm will allow the centroid to move in the direction of the mean and thus update in the direction which is optimal for each segmentation and satisfies K-mean's goal which is best described as follows:

"The goal of K-means clustering is to find the cluster centres $c_1, \dots, c_k$ that partition the data in such a way that the sum of the squared-distances of each data point to its closest centre is minimal"[2]

In an online setting, this minimisation formulation equates to:

$$argmin_{\mathrm{i}_1,\mathrm{i}_2,\dots,\mathrm{i}_{k-1};\, \mathrm{c}_1,\dots,\mathrm{c}_k} \sum_{j=1}^{k} \sum_{p=\mathrm{i}_{j-1}+1}^{\mathrm{i}_j} \left\| x_p - c_j \right\|^2 \qquad \text{e. q. xxx}$$

This requires the centroids to be as close to the data points in each cluster as possible and thus minimised the error formulation. **The proposed algorithm satisfies this requirement as it moves the centroids towards the mean point with respect to each segment essentially duplicating the behaviour of standardised k-means in an online setting as the centroid tends towards the mean data point of each respective segmentation**. Therefore the algorithm succeeds in finding the $argmin$ of the data with respect to k such that optimal clusters are found minimising the squared distance between that of the data points and the respective centroid.

---

[1] This is implemented as a while loop as the algorithm is online and thus the amount of data is tantamount on the API used and the rate of convergence.

[2] Source: Homework 2 (Group) – Mark Herbster (*This Coursework*)

The proposed algorithm is polynomial in k (*clusters*), n (*dimensions*) and l (*data vector length*) as it requires only a while loop and an inner for loop equating to a complexity of $\mathbf{O}(n.l.k)$ with the other operations equating to $\mathbf{O}(1)$.

2. PCA

2.1. Exercises

2.1.1. Implementing PCA

We have created a function PCA.m that takes mxn data matrix as well as k = number of new principle components. It returns transformed data.

Code for 2.1.1.

```
function [LinearMap] = PCA(X,k)
% X is mxn data matrix
% k - new principle components
[m n]=size(X);
for i=1:n
   X(:,i)=X(:,i)-sum(X(:,i))/m;
end
 [EigVectors,EigValues]=eig(cov(X));
FirstKEigVec=EigVectors(:,1:k);
PrincipleAxes=FirstKEigVec;
LinearMap=X*PrincipleAxes;
```

2.1.2. K means (kmeansPCA.m)

For this question we decided to use our already developed function kmeans_cw.m as it returns all the output that we need. We simply parse these values and rearrange them to provide an output of *(n+1)*x*1* where the very first element is the sum of the value of our objective function with respect to every data entry i.e. the value of the objective function for the current clustering and the remaining n elements are cluster indices.

Code for 2.1.2.

```
function [r_vector] = kmeansPCA( data, k )
result=kmeans_cw(data, k);
[m n]=size(data);
clusters = result(1:n,:);
r_matrix = result((n+1):end,:);
r_vector=r_matrix*(1:k)';
for i=1:m
   objective(i)=sum((data(i,:)-(clusters(:,r_vector(i)))').^2);
end
 objective=sum(objective);
 r_vector=[objective;r_vector];
```

2.1.3. IRIS dataset – ADD K-MEANS ACCURACY FOR EACH OF THE 5 DATASETS

(a) Give the 3 smallest "occes" – AMEND VALUES IN TABLE

We are going to use our functions KmneasPCA.m and PCA.m After creating the PCA, we perform KmenasPCA.m 100 times. The smallest 3 for every PCA dimension are the following:

	OCCE	Objective	OCCE	Objective	OCCE	Objective
K=1	0.1067	78.9408	0.1067	78.9408	0.1067	78.9408
K=2	0.4333	78.9408	0.4333	78.9451	0.4333	78.9451
K=3	0.4333	78.9408	0.4333	78.9408	0.4333	142.8593

K=4	0.3933	143.4537	0.4333	78.9408	0.4333	78.9408
K=5	0.4333	78.9408	0.4333	78.9408	0.4333	78.9451

With the three smallest to be given for K=1. Code for allocating the three smallest occes using Matlab is provided.

(b) Mean and Standard deviation of "occes" and "objective". –
AMEND VALUES IN TABLE

	Mean(OCCE)	Sd(OCCE)	Mean(Objective)	Sd(Objective)
K=1	0.1226	0.0249	92.5157	26.4606
K=2	0.4484	0.0130	0.6878	0.00036
K=3	0.4482	0.0133	0.6879	0.00031
K=4	0.4451	0.0140	0.6950	0.0719
K=5	0.4467	0.0132	0.6879	0.00031

(c) The OCCE's is plotted as a function of the rank where rank is defined as the sorted ascending index on the objective values. This allows the following dataset to be created:

$$(Objective, OCCE)$$

From this point the objective is replaced with the rank (*essentially the index*) where the data structure is updated to:

$$(Rank, OCCE)$$

The following is then plotted:

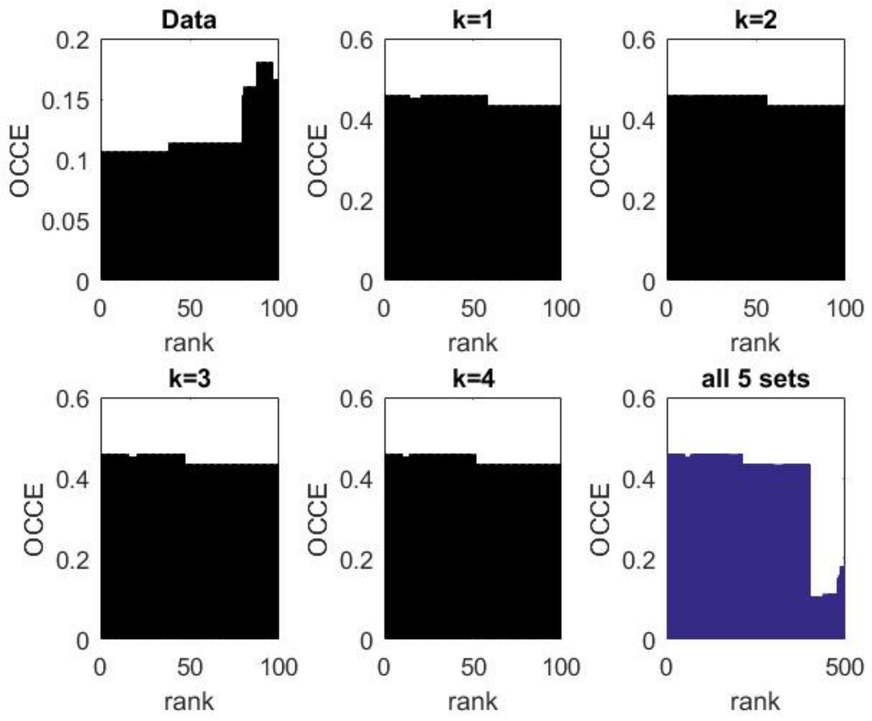


Figure 1: OCCE Visualisation

The above shows the bar charts for 100 runs at each value of k where the bottom right chart exhibits all of the charts in one which allows a direct comparison of the change in behaviour. [insert analysis on observed behaviour change dependent on k]

2.1.4 Visualisation

The clustered data is visualised in **two** and **three** dimensions and are exhibited in Figure 3 and 4.

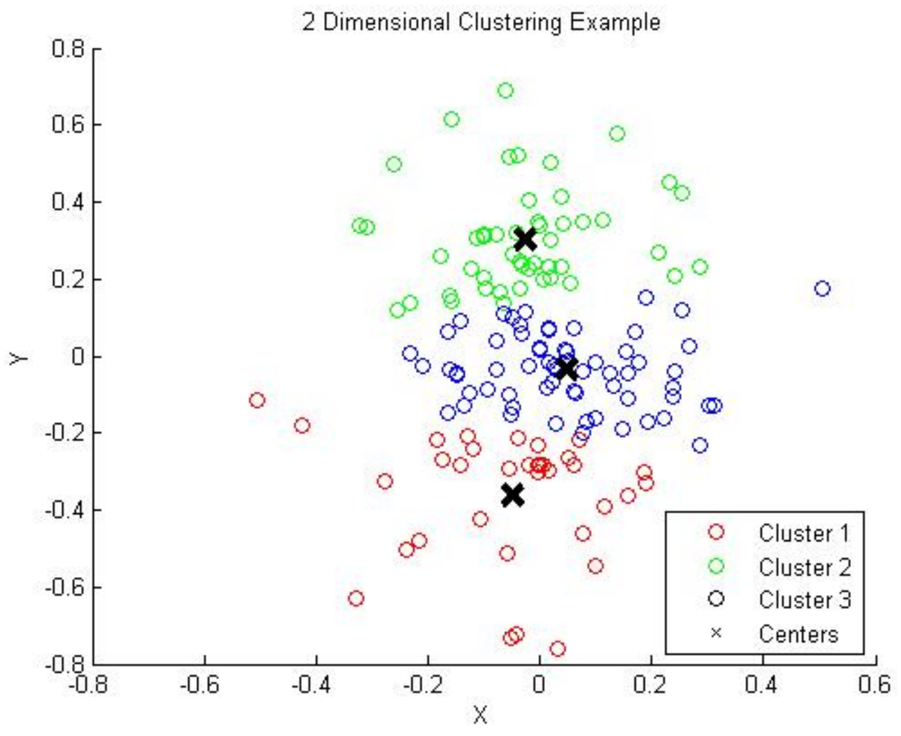


Figure 2: 2-dimensional visualization

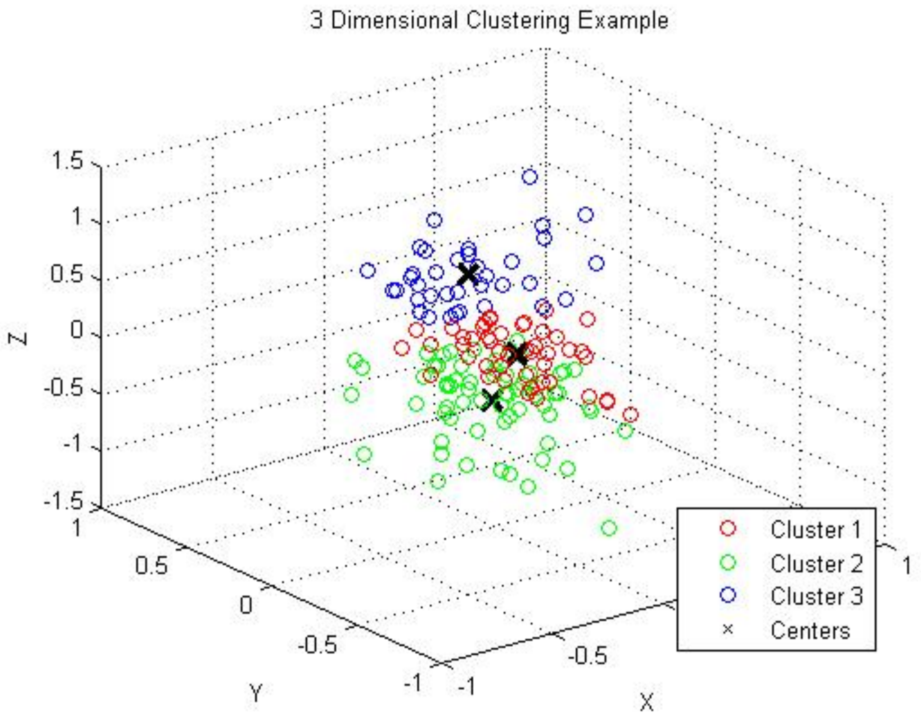


Figure 3: 3-dimensional visualization

2.2 Computing the OCCE

Using k! iterations to check for the smallest OCCE is a rather slow approach. In general we are interested in the mode of the elements in a cluster. This way we do not have to compare always the actual cluster name/index. We have used this approach for 2.1.3 and now we have created a function OptimisedOCCE.m where is generalised for every type of dataset. It takes a vector of predicted clusters using clustering and actual clusters. Assuming mode is $O(n)$, where n is the number of elements in the cluster, we can perform a single for loop in k $O(k)$ where we calculate occe by adding to the previous value the result of taking all elements in predicted cluster, that are equal to the looped cluster (i) and then comparing their values with the mode of this operation. This action is again polynomial in n, and independent from k. For this reason our algorithm is $O(k)$.

Code for 2.2.