Median Based Clustering for Underdetermined Blind Signal Processing

Aditya Puranik, Vaibhav Sachdeva

July 4, 2020

Abstract

In underdetermined blind source separation, more sources are to be extracted from less observed mixtures without knowing both sources and mixing matrix. K-means-style clustering algorithms are commonly used to do this algorithmically given sufficiently sparse sources,but in any case other than deterministic sources, this lacks theoretical justification. After establishing that mean-based algorithms converge to wrong solutions in practice, we propose a median-based clustering scheme. Theoretical justification as well as algorithmic realizations (both online and batch) are given and illustrated by some examples.

1 Introduction

Blind Source Separation (BSS) is the separation of a set of source signals from a set of mixed signals, without the aid of information (or with very little information) about the source signals or the mixing process. The goal here is to identify the mixing matrix $\mathbf{A}$ by **Blind Mixing model Recovery** (BMMR) and the $\mathbf{n}$-dimensional source random vector $\mathbf{s}$ by **Blind Source Recovery** (BSR) from an observed mixture random vector $\mathbf{x}$ where $\mathbf{x}$ is:

$$\mathbf{x} = \mathbf{A}\mathbf{s}$$

where,

$\mathbf{x}$ = m-dimensional mixture random vector
$\mathbf{s}$ = n-dimensional source random vector with component density p(s)
$\mathbf{A}$ = mixing matrix having independent columns each of unit norm

More sources are to be extracted from less observed mixtures in **Underdetermined Blind Source Separation** ($n < m$) without knowing both the sources and the mixing matrix. Blind Source Separation (BSS) is predominantly based on independent source assumptions. Assuming that statistically independent sources have at most one Gaussian variable, it is known that $\mathbf{A}$ is uniquely determined by $\mathbf{x}$. The most commonly used algorithms are based on sparse sources that are correctly identified by a clustering of k-means. But, mean-based clustering can only classify the appropriate $\mathbf{A}$ if the data density approaches a delta

distribution. Mean-based clustering has no equivariance property which suggests that the performance would be independent of **A**. Thus, a **median-based** approach has been chosen.

Blind Source Separation follows a two step approach:

1. Geometric matrix recovery of **A** through Blind Mixing Model recovery (BMMR).
2. Source extraction through Blind source recovery (BSR).

2 Background

The most commonly used overcomplete algorithms rely on sparse sources , which can be identified by clustering, usually by k-means or some extension. However, apart from the fact that mean-based clustering lacks theoretical justifications, it only identifies the correct A if the data density approaches a delta distribution and hence, at times converges to the wrong solution. Also, mean based clustering does not possess any equivariance property which suggests that performance will be independent of **A**. However, apart from the fact that theoretical justifications have not been found, mean-based clustering only identifies the correct **A** if the data density approaches a delta distribution. In Fig. 1, we illustrate the deficiency of mean-based clustering; we get an error of up to 5 per mixing angle, which is rather substantial considering the sparse density and the simple, complete case of m=n=2. Moreover, the figure indicates that median-based clustering performs much better. Indeed, mean-based clustering does not possess any equivariance property (performance independent of **A**).

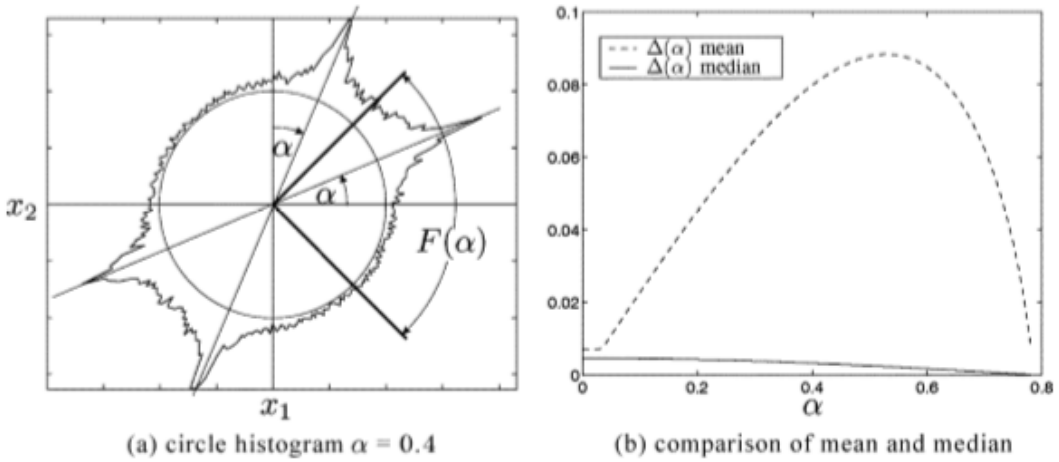


Figure 1: Mean- versus median-based clustering. We consider the mixture x of two independent gamma-distributed sources ($\gamma = 0.5, 10^5$ samples) using a mixing matrix A with columns inclined by α and ($\pi/2$ - α), respectively. (a) Mixture density for $\alpha = 0.4$ after projection onto the circle. (b) For α [0; $\pi/4$), we compare the error when estimating **A** by the mean and the median of the projected density in the receptive field F(α)=(-$\pi/4$; $\pi/4$) of the known column a of **A**. The former is the k-means convergence criterion.

3 Methods

3.1 Geometric Matrix Recovery

Without loss of generality, we assume that $\mathbf{A}$ has pairwise linearly independent columns, and $m < n$. BMMR tries to identify $\mathbf{A}$ in $\mathbf{x} = \mathbf{As}$ given $\mathbf{x}$, where $\mathbf{s}$ is assumed to be statistically independent. Obviously, this can only be done up to equivalence, where $\mathbf{B}$ is said to be equivalent to $\mathbf{A}$, $\mathbf{B} \sim \mathbf{A}$, if $\mathbf{B}$ can be written as $\mathbf{B} = \mathbf{APL}$ with an invertible diagonal matrix $\mathbf{L}$ (scaling matrix) and an invertible matrix $\mathbf{P}$ with unit vectors in each row (permutation matrix). Hence, we may assume the columns a_i of $\mathbf{A}$ to have unit norm. For geometric matrix-recovery, we use a generalization of the geometric independent component analysis (ICA) algorithm. Let $\mathbf{s}$ be an independent n-dimensional, Lebesgue-continuous, random vector with density p_s describing the sources. As $\mathbf{s}$ is independent, p_s factorizes into $p_s(s_1, ..., s_n) = p_{s1}(s_1).p_{s2}(s_2)..p_{sn}(s_n)$ with the one-dimensional marginal source density function p_{si}. We assume symmetric sources, i.e., for $s \in \Re$ and $i \in [1 : n] :=$ 1,...,n , in particular $\mathbf{E(s)} = 0$. The *geometric blind mixing model recovery* (BMMR) algorithm for symmetric distributions goes as follows :

Pick *2n* starting vectors on the unit sphere $w_1, w_1', ..., w_n, w_n'$ such that w_i) and w_i' are opposite each other, i.e.w_i=$-w_i'$ for i=1,.....,n and such that the w_i are pairwise linearly independent vectors. These w_i are called **neurons**.

All the neurons are updated using the following iteration rule using **k-means** algorithm.

$$w_i(t+1) = \pi(w_i(t) + \eta(t)\pi(y(t) - w_i(t)))$$

where,

$$y(t) = x(t)/|x(t)|$$

learning rate: $\eta : N \to \Re$ such that $\eta(t) > 0$.

$$\pi(x) := x/|x|$$

such that $w_i'(t+1) = -w_i'(t+1)$. We will mean by median in the above algorithm and prove the equivariance and convergence property of the performance.

3.2 BSR

Using the results from the BMMR step, we can assume that an estimate of $\mathbf{A}$ has been found. In order to solve the overcomplete BSS problem, we are therefore left with the task of reconstructing the sources using the mixtures $\mathbf{x}$ and the estimated matrix (BSR). Since $\mathbf{A}$ has full-rank, the equation yields the $n - m$-dimensional affine vector space as solution space for $\mathbf{s(t)}$. Hence, if $n > m$, the source-recovery problem is ill-posed without further assumptions. Using a maximum-likelihood approach, an appropriate assumption can be derived.

The maximum-likelihood algorithm states that the probability of observing $\mathbf{x}$ given $\mathbf{A}$ and $\mathbf{s}$ can be written as $P(\mathbf{x}|\mathbf{s}; \mathbf{A})$.

Using Bayes Theorem the posterior probability of $\mathbf{s}$ is -

$$P(\mathbf{s}|\mathbf{x}, \mathbf{A}) = \frac{P(\mathbf{x}|\mathbf{s}, \mathbf{A}) \times P(\mathbf{s})}{P(\mathbf{x})}$$

such that $\mathbf{x} = \mathbf{A}\mathbf{s}$.
For reconstructing $\mathbf{s}$, we will maximize the posterior probabilities

$$\mathbf{s} = argmax\ P(\mathbf{s}|\mathbf{x}, \mathbf{A}) = argmax\ P(\mathbf{x}|\mathbf{s}, \mathbf{A}) \times P(\mathbf{s})$$

3.3 K-Means

K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). It is a centroid-based algorithm, or a distance-based algorithm, where we calculate the distances to assign a point to a cluster. In K-Means, they connect each cluster with a centroid. The K-Means algorithm 's main goal is to minimize the sum of distances between the points and their respective centroid clusters. Every centroid of a cluster is a set of function values that identify the groups that result. Examining the weights of the centroid characteristic can be used to define qualitatively what sort of group each cluster represents.

3.3.1 Algorithm

1. Choose the number of clusters $\mathbf{k}$.
2. Place K points within the space occupied by the clustered objects. Those points reflect centroids of the initial group.
3. Assign each object to the group that has the closest centroid.
4. Recompute the centroids of newly formed clusters.
5. Repeat steps 3 and 4 until centroids stop moving any longer. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

3.3.2 Stopping Criterion

There are widely three stopping criterion to stop the iteration of K-means algorithm:

- Centroids of newly formed clusters do not change.
- Points remain in the same cluster
- Maximum number of iterations are reached, where maximum iterations can be decided by the trainer according to computational complexity.

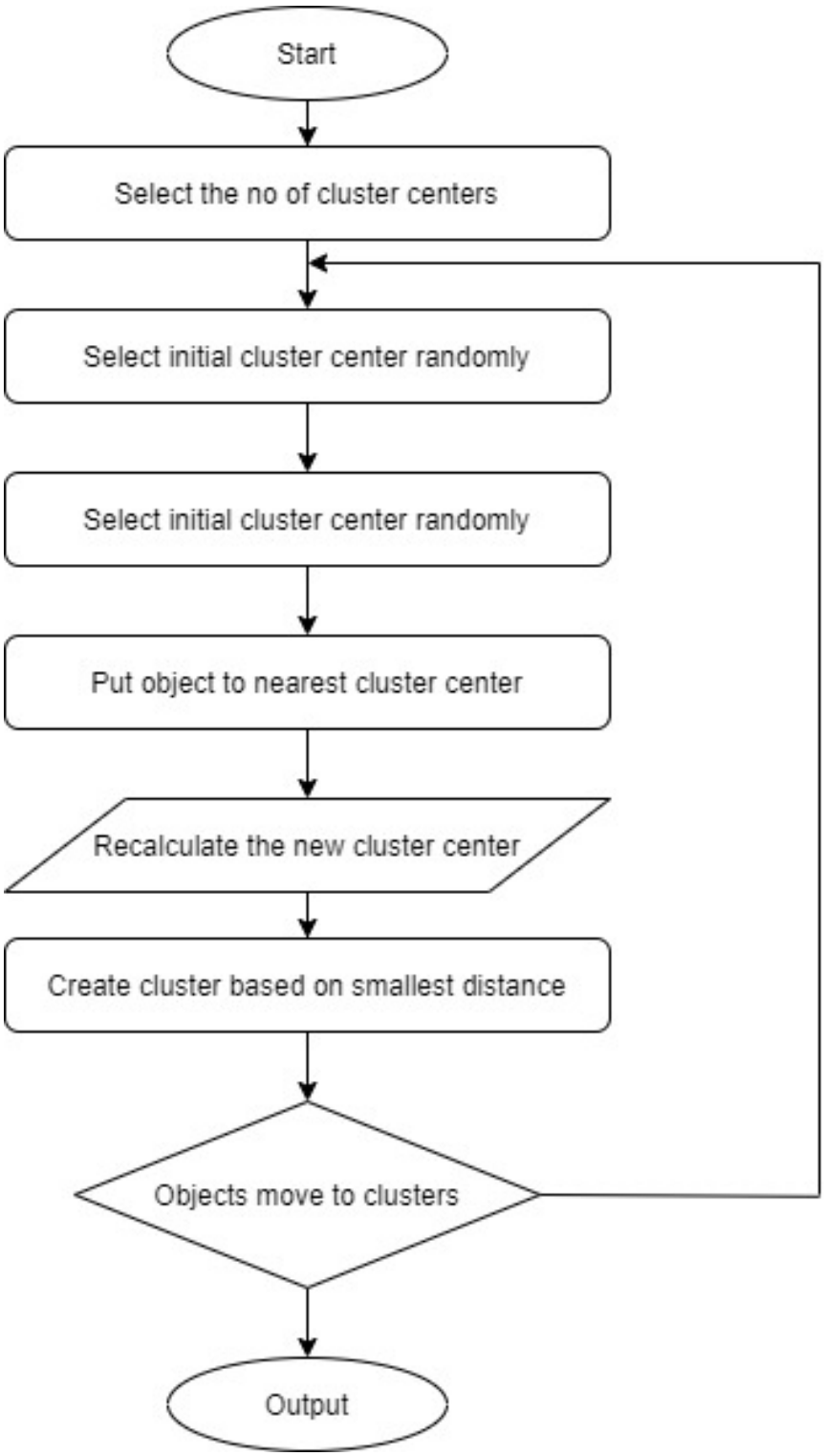


Figure 2: K-Means

3.4 K-medoids

Partitioning Around Medoids or the **K-medoids** algorithm is a partitional clustering algorithm which is slightly modified from the K-means algorithm. They both attempt to minimize the squared-error but the K-medoids algorithm is more robust to noise than K-means algorithm. In K-means algorithm, they choose means as the centroids but in the K-medoids, data points are chosen to be the medoids/centroids. K-means attempts to minimize the total squared error, while k-medoids minimizes the sum of dissimilarities between points labeled to be in a cluster and a point designated as the center of that cluster. In contrast to the k-means algorithm, k-medoids chooses data points as centers (medoids or exemplars).

The **difference** between k-means and k-medoids is **analogous** to the difference between mean and median: where mean indicates the average value of all data items collected, while median indicates the value around that which all data items are evenly distributed around it. The basic idea of this algorithm is to first compute the K representative objects which are called as medoids. After finding the set of medoids, each object of the data set is assigned to the nearest medoid. That is, object i is put into cluster v_i, when medoid mv_i is nearer than any other medoid m_w.

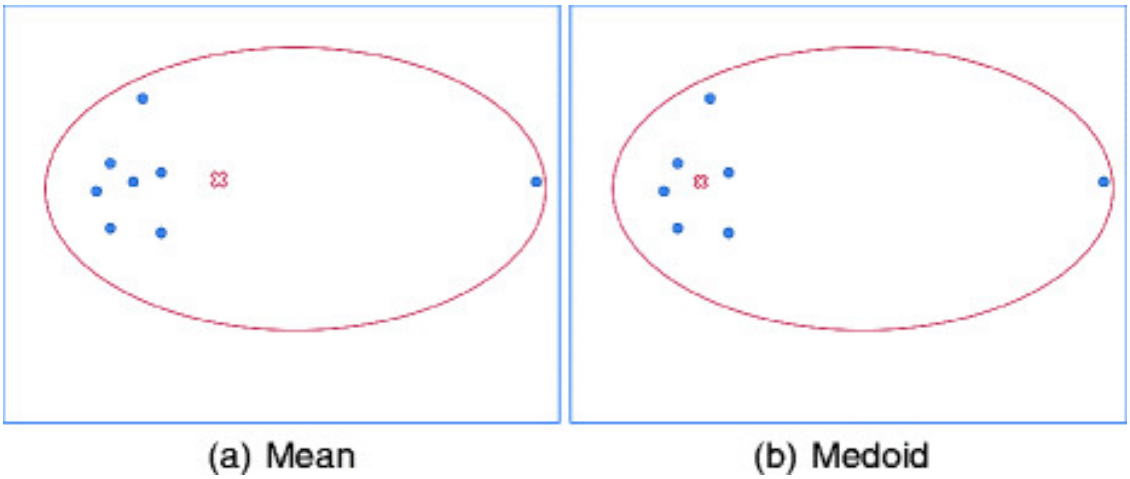


Figure 3: Comparison between K-means and K-medoids. Here we can see that, the outlier affects the K-means a lot more than K-medoids. In K-medoids, we stick with point in the data set and thus K-medoid is more robust to noise and outlier.

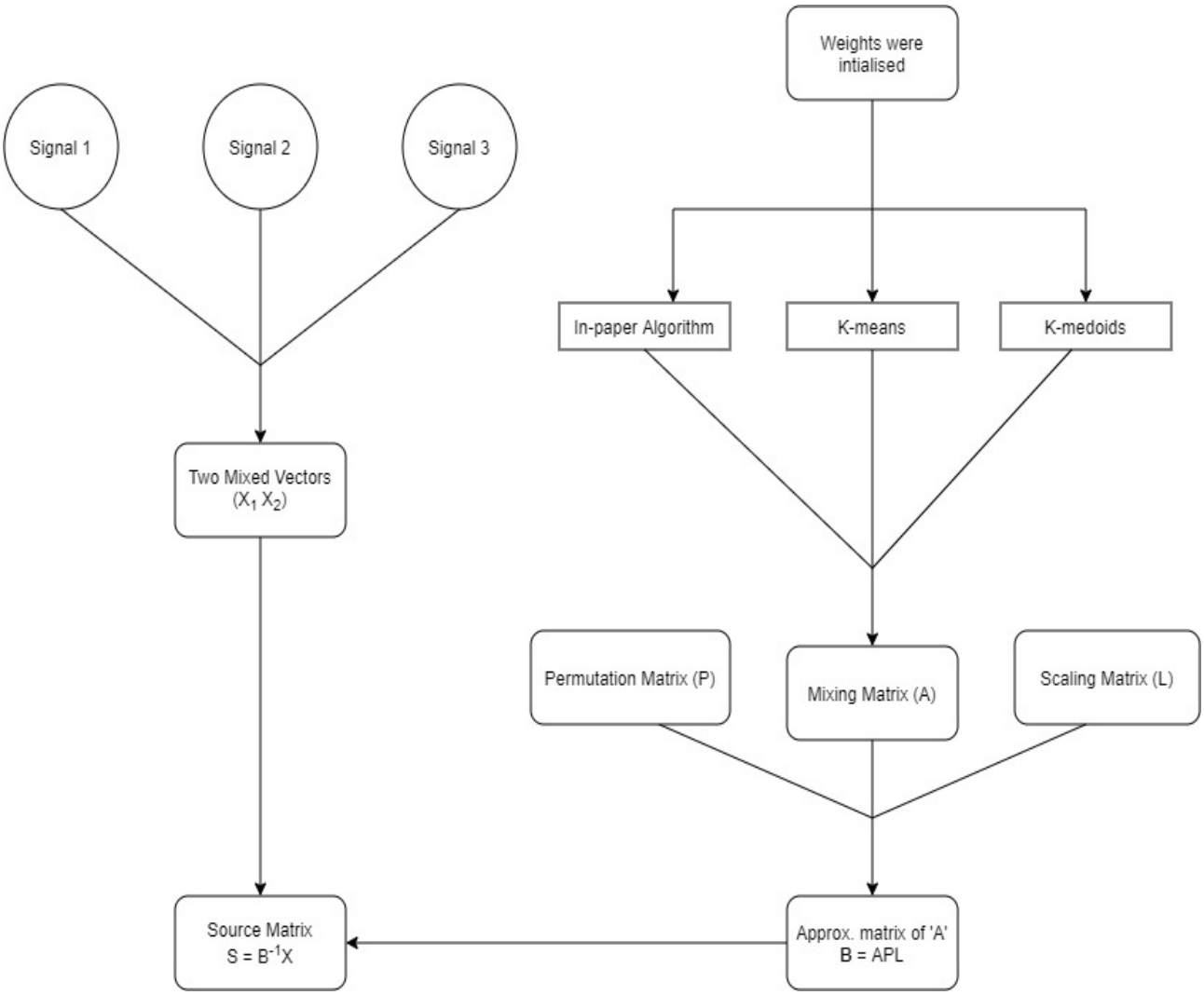


Figure 4: Flow Chart of the BSS Process

4 Results and Conclusion

Model	**Mean Error E(A, $\hat{A}$)**
In-paper Algorithm	1.5097
K-Means	1.5631
K-Mediods	1.5496

Table 1: Performance of various algorithms

Our algorithm approximates the matrix well with E(A, $\hat{A}$) = 1.5097. The paper mentions the mean error range when using random (2 × 3)-matrices with coefficients uniformly taken from [-1,1] as E = 1.9 ± 0.73, which our implementation justifies.

Also, K-medoids seems to be a better model here than K-means, since it gives more consistent result and is more robust to noise and outliers. Although, there are few cases in which K-means performs better than K-medoids, but consistency wise K-medoids is much better.

Model	**Probability)**
In-paper Algorithm	0.441
K-Means	0.267
K-Mediods	0.292

Table 2: Probability of getting the minimum error over 1000 iterations

We encountered an irregularity in the output error, with sometimes either of the In-built K-means or K-medoids clustering algorithm giving better results (lesser error) than the In-Paper Algorithm. So, in order to get a clear picture of the algorithm that gives the best performance (minimum error), we ran our code 1000 times and observed the probability of each algorithm giving the least error as compared to others. It can be clearly seen from the results that 440 out of 1000 times, the In-Paper algorithm gives the least error followed by K-Medoids with 292 times and K-Means with 267 times. Thus, it can be concluded that the In-Paper Algorithm is the best approach for Underdetermined Blind Source Separation.

5 Learning Outcomes

5.1 Aditya Puranik - 1710110025

This project served as a learning curve for me in a way that I understood the complexity involved and assumptions needed to isolate a single speech signal from a mixture of signals and gave me a peek into the marvels of our brain. Here, we looked at a an **Under-determined BSS** problem.

Under-determined Blind Source Separation is the process of extracting source signals from their mixed signals, so that the sources are more in number than their mixtures. It is done without understanding the individual signals or the way their mixture is produced. We discussed in this paper the technique to extract the same. We have considered 3 sources **s1**, **s2**, and **s3** and obtained 2 mixtures **x1** and **x2** in our implementation. Thus our mixing matrix **A** has 6 weights.

We assumed all the sources to be statistically independent. For the separation, we followed $x = As$ for the linear combination of the signals, where **s** is our source vector, and **A** is the mixing matrix. BSS was implemented in two steps:

1. **Geometric matrix recovery of A**: This was achieved through BMMR, in which we identify **A** in $x = As$ where we are given **x**, the mixture matrix.

2. **Blind Source Recovery**: In BSR, we try to recover the source signals from the obtained mixing matrix and the mixture signals.

I also looked into Independent Component Analysis (ICA) method, which is an important step for taking assumptions for BMMR step. Independent component analysis (ICA) is a computational method for separating a multivariate signal into additive sub-components. This is done by assuming that the sub-components are non-Gaussian signals and that they are statistically independent from each other. A simple application of ICA is the "cocktail party problem", where the underlying speech signals are separated from a sample data consisting of people talking simultaneously in a room. Usually the problem is simplified by assuming no time delays or echoes.

Firstly, we looked at the online K-means algorithms which was mentioned in the paper to find the updated weights. We chose online training over batch for our data-set because each data point is given one-by-one for training resulting in better performance. Secondly, we approximated the mixing matrix by using the appropriate Permutation and Scaling matrices with A. For this we use, B=APL, where, P is the permutation matrix and L, the corresponding scaling matrix. P is an invertible matrix comprising of unit vectors in each row, and L is an invertible diagonal matrix. Through these we assume, columns of a_i of A have unit norm.

K-means and K-medoids algorithms can be used for clustering purposes. Medoid is always a point in the dataset that gives minimum average error with respect to the remaining data points. In contrast to this, the mean of the dataset may not be a point in the dataset, and gives the minimum Euclidean distance error with respect to the data points.

Mixing matrix A in a given mixture is the weights of each source, or the composition of each source in a mixture. We initialized A with some random weights to find the mixing

matrix A, and applied K-means and K-medoid clustering algorithm on the mixtures to find the weight of each source. The number of clusters formed will be equal to the number of retrieved sources. On comparing, the K-medoids and K-means, I could conclude that K-medoids is more robust to outliers and noise. Also, K-medoids was more consistent with its results and thus I think it is a more bankable algorithm if there is a lot of noise in the system.

As mentioned earlier, this project is not just an academic project which runs in background of an process. The particular algorithm covered in this have many real life applications. Like in forensics and crime intelligence cells, to recognize the victim's voice or hear clearly the dialog occurring which might have sensitive information from a blurry audio or overlapping voices is an important task. This technique can be used in that particular case to extract only relevant information in that particular situation.

5.2 Vaibhav Sachdeva - 1710110369

This project has been great learning experience and has helped me understand the application of various algorithms to real life problems.
Our project was about underdetermined blind source separation (BSS). In this, more sources are to be extracted from less observed mixtures without knowing the sources as well as the mixing matrix. In our project all sources were assumed to be independent of each other. Hence, the mixtures we have with us are a linear combination of the independent source signals. We follow the equation, x=As for the separation of sources. Here, x is the mixture matrix, s is the source vector, and A is the mixing matrix. We implemented BSS in two steps.

The first step was Blind Mixing Model Recovery (BMMR), which is concerned with the geometric recovery of the mixing matrix A. The second step was Blind Source Recovery (BSR), which helps us to recover the source signals from the obtained mixing matrix and the mixture signals. Apart from this, I was also introduced to the concept of Independent Component Analysis which is a statistical and computational technique for revealing hidden factors that underlie sets of random variables, measurements, or signals. Independent Component Analysis defines a generative model for the observed multivariate data, which is typically given as a large database of samples. ICA was used for making some important assumptions without which solving the underdetermined problem couldn't have been possible.

Mixing matrix A constitutes of weights of each source in a given mixture, or the proportion of each source in a mixture. In order to find the mixing matrix A, we initialized A with some random weights (as given in the paper) and applied the weight updation rule given in the paper in order to obtain the optimum weights. Apart from that, we also used inbuilt (MATLAB) K-means and K-medoid clustering algorithm on the mixtures in order to find the weight of each source. The number of clusters formed will be equal to the number of sources to be recovered. Upon comparing the three algorithms used for updating weights, I was able to conclude that the in-paper algorithm and inbuilt K-medoids proved to be better as compared to K-means in certain aspects. Medoid is always a point in the dataset that gives minimum average error with respect to the remaining data points. In contrast to this, the mean of the dataset may not be a point in the dataset, and gives the minimum Euclidean distance error with respect to the data points. This makes the clustering approach based on medoid more robust to noise and outliers. These conclusions were consistent with the results we obtained on implementing our project.

We found the approximated mixing matrix using appropriate Permutation and Scaling matrices with A. For this we used the equation, B=APL, where, P is the permutation matrix and L is the corresponding scaling matrix. Once we have the approximate of the mixing matrix we can simply apply x=As and obtain the corresponding sources (using pseudo-inverse method). The project has not only introduced me to an entirely new domain of Blind Source Separation but has also helped me understand the various real life applications of the above used algorithms.

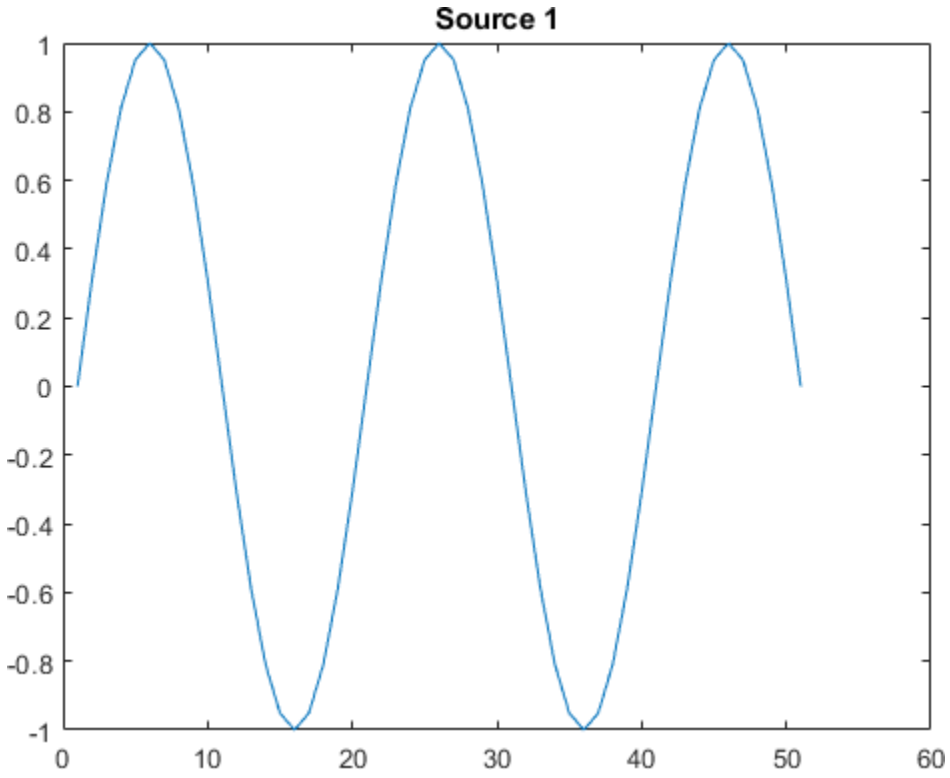
One such application is when we have to separate the audio sources given their noisy mixtures. Suppose three music instruments are being played simultaneously in a hall and we have two mixtures of these sources coming from two microphone recordings; we can almost fully recover the three sources from the available mixtures using BSS. Here, our BSS will be underdetermined because the number of mixtures is less than the number of sources.

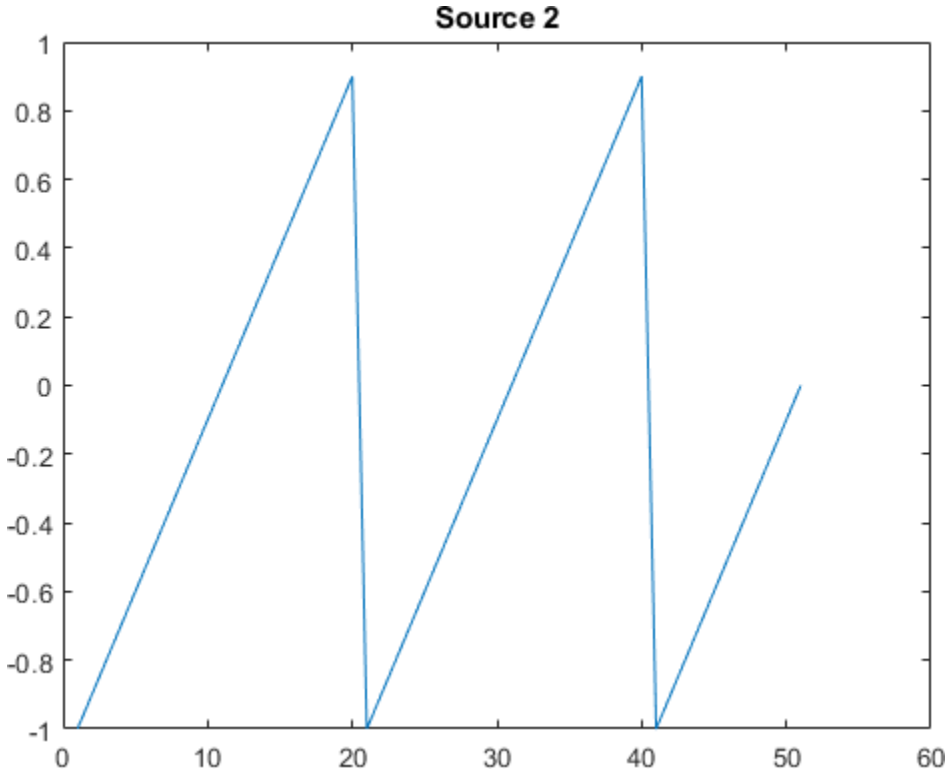
After reading and implementing this paper to an extent, I feel that we can use these methods providing considerably good techniques to recover the underlying and priorly unknown source signals from different kinds of random mixtures. Thus all the above mentioned points account to my learning outcomes.

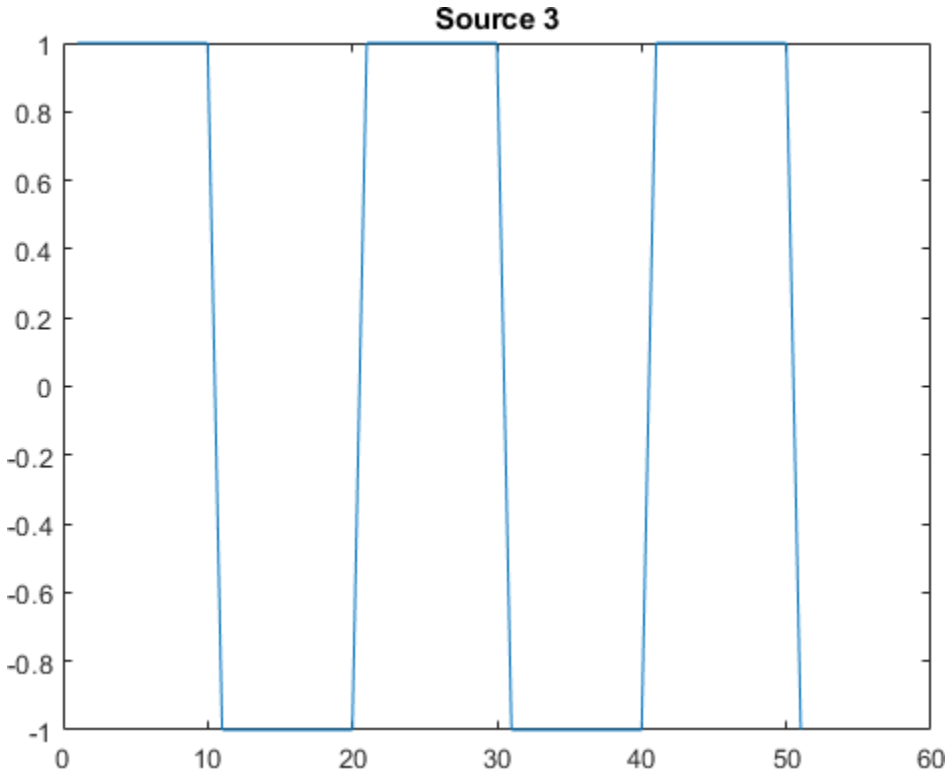
Table of Contents

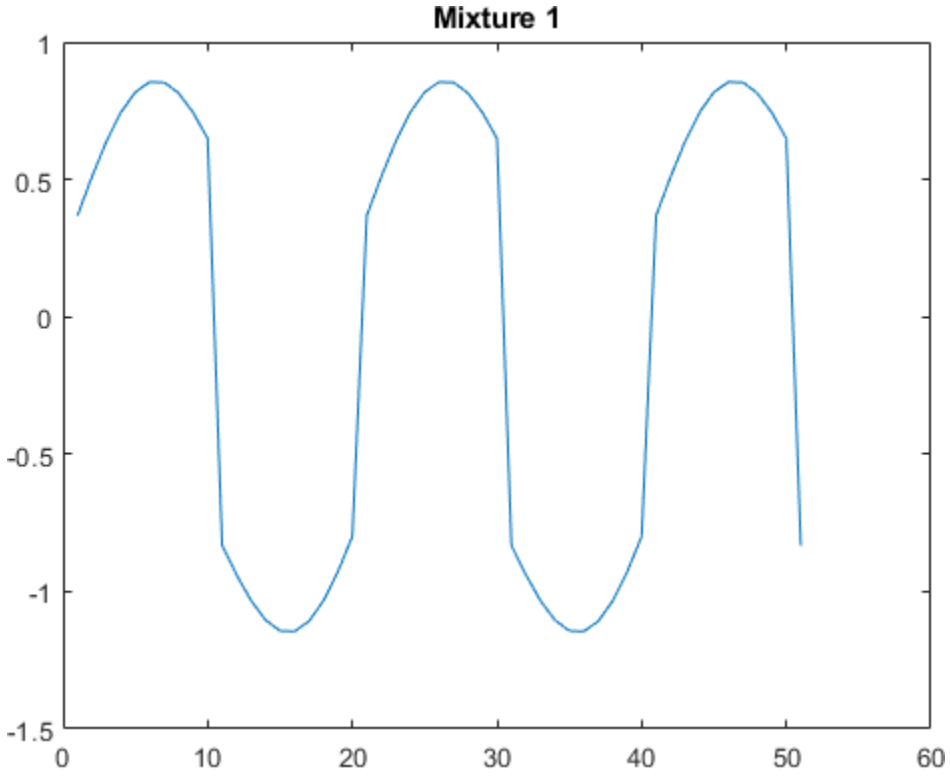
Generating Signals and Mitures

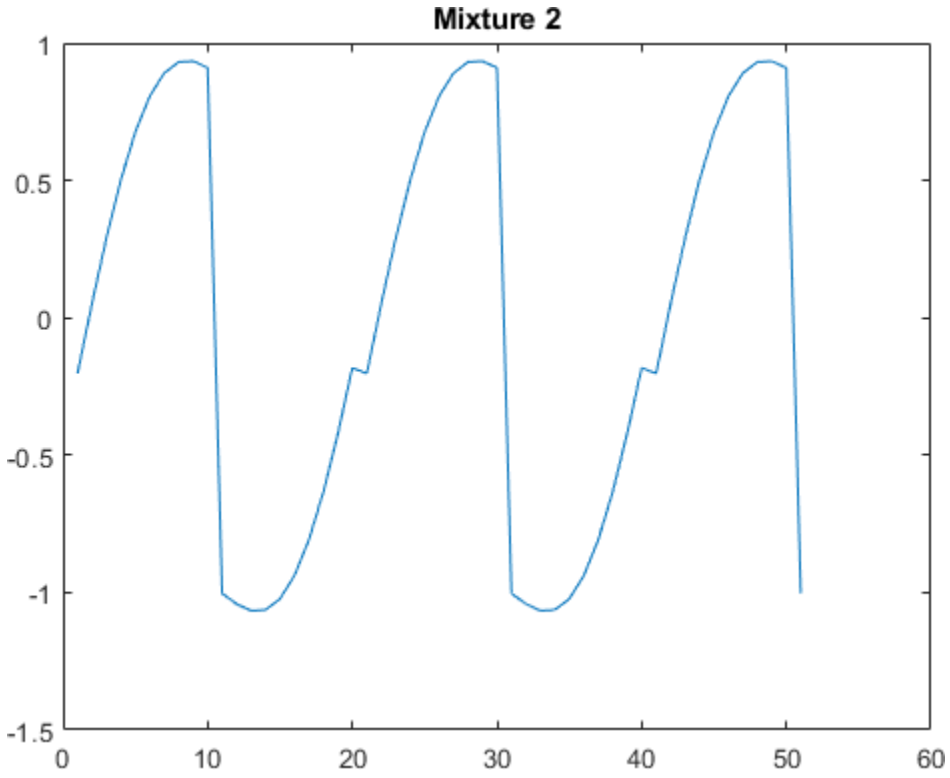
```
clc;
clear all;
close all;

% Defining the time period and frequency for the signals.
tp = 0:0.1:5;
freq = 0.5;

% Genrating 3 signals. Three types of waveforms have been used -
% Sine, Sawtooth and Square
signal1 = sin(2*pi*freq*tp);
signal2 = sawtooth(2*pi*freq*tp);
signal3 = square(2*pi*freq*tp);

% Plotting the 3 source sigals
figure, plot(signal1),title('Source 1');
figure, plot(signal2),title('Source 2');
figure, plot(signal3),title('Source 3');

% Generating two mixtures of signals by multiplying each signal with a
% random number and adding them up.
x1 = rand(1)*signal1 + rand(1)*signal2 + rand(1)*signal3;
x1 = x1/max(x1);
x2 = rand(1)*signal1 + rand(1)*signal2 + rand(1)*signal3;
x2 = x2/max(x2);
x1_bar = x1 - mean(x1);
x2_bar = x2 - mean(x2);

% Plot for generated mitures
figure, plot(x1_bar),title('Mixture 1');
figure, plot(x2_bar),title('Mixture 2');

% Combining the mixtures into a matrix
X = [x1 ; x2];

% Combining the sources into a matrix
S = [signal1 ; signal2 ; signal3];
```


Source 1
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
10
20
30
40
50
60


Source 2
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
10
20
30
40
50
60


Source 3
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
10
20
30
40
50
60


Mixture 1
1
0.5
0
-0.5
-1
-1.5
0
10
20
30
40
50
60



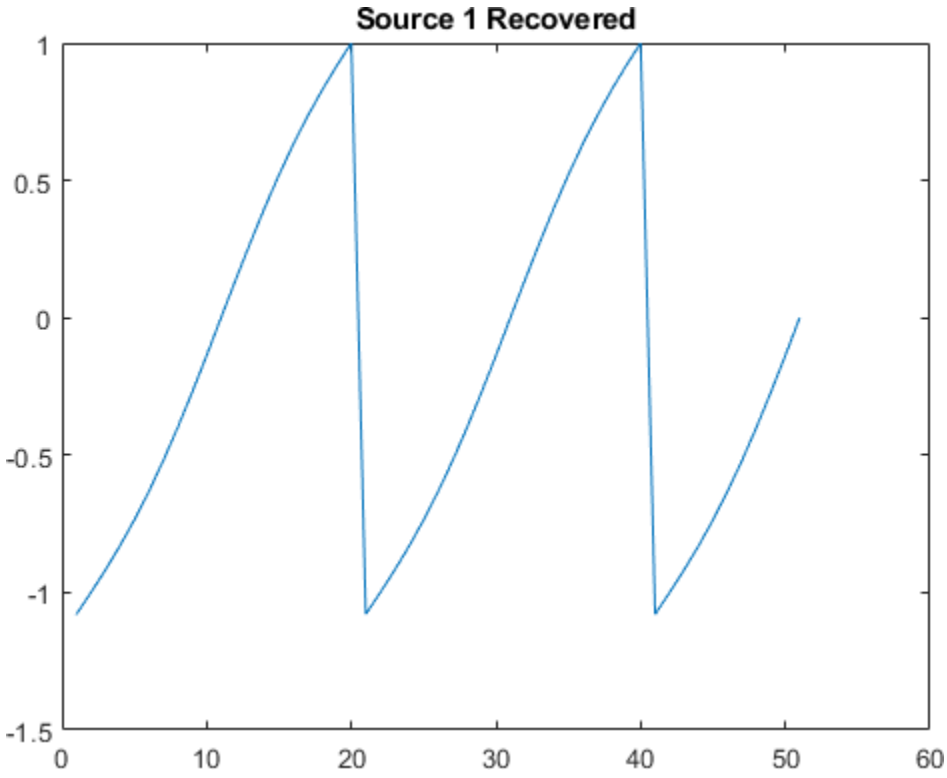
Obtaining Signals back from given Mixtures

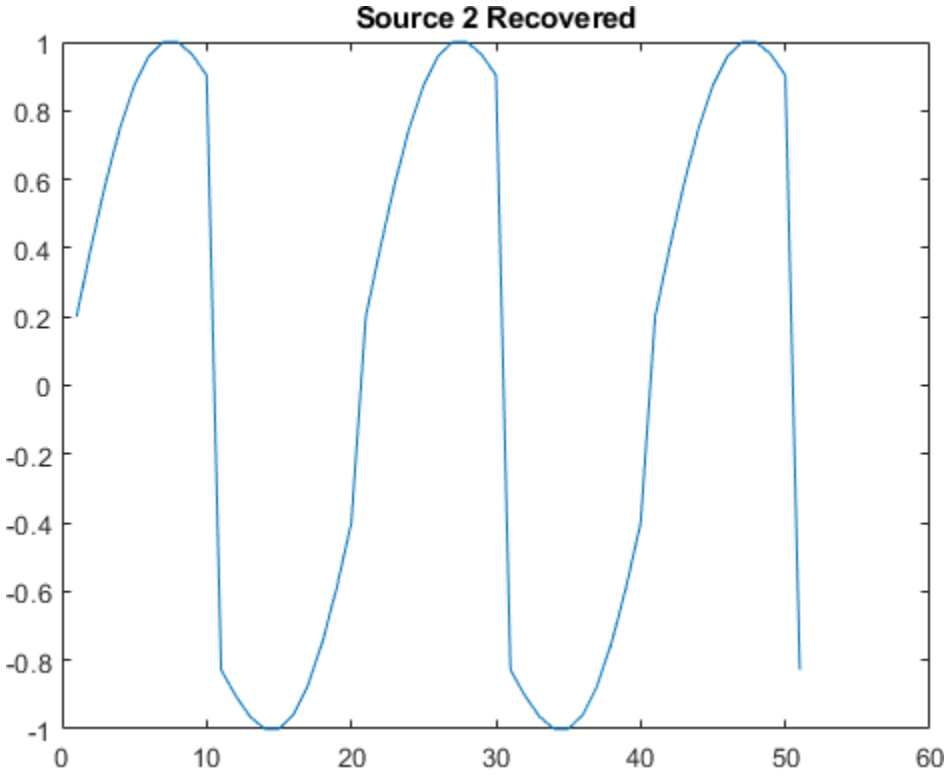
```
% A has been initialised with the values suggested in the paper
% Columns of A i.e ai = (cos(alphai), sin(alphai)) with alpha between
% 0 and pi.
alpha1 = pi/4;
alpha2 = pi/2;
alpha3 = 3*pi/4;
A = [cos(alpha1), cos(alpha2), cos(alpha3); -cos(alpha1), -
cos(alpha2), sin(alpha3)];

% Initializing Weights(Neurons) as suggested in the paper

w1 = [cos(alpha1)];
w_1 = [-cos(alpha1)];
w2 = [cos(alpha2)];
w_2 = [-cos(alpha2)];
w3 = [cos(alpha3)];
w_3 = [sin(alpha3)];

% Learning rate
n = 0.5;

% There are a total of 6 weights in the mixing matrix and each weight
```

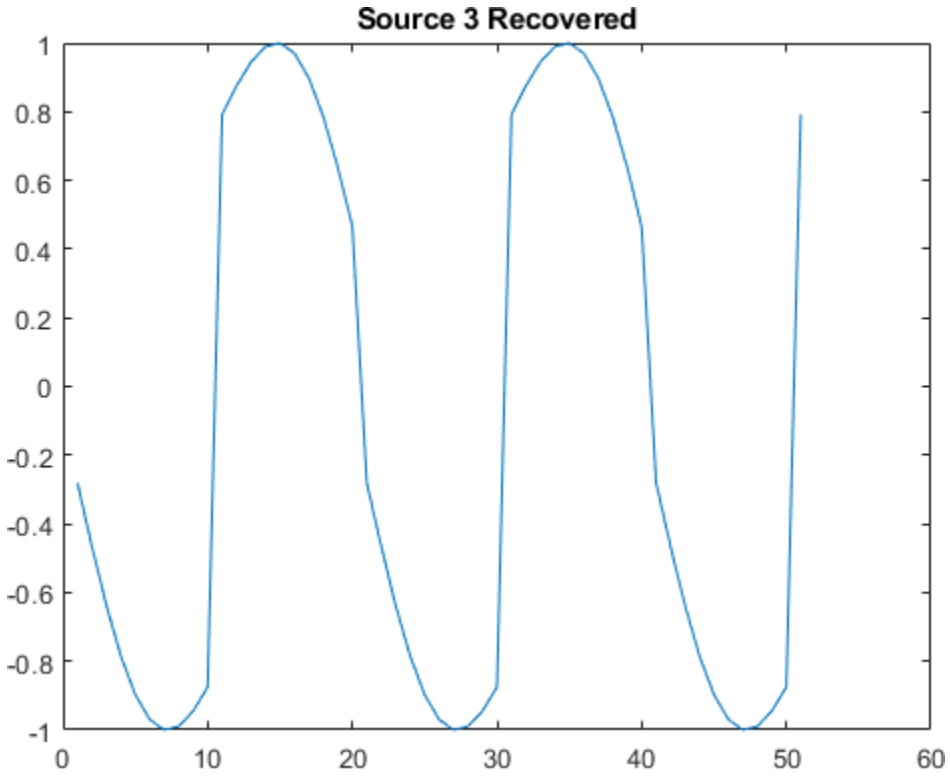
```
% has been updated 50 times to obtain suitable values of weights for
 further
% calculations.
% Pie function as mentioned in the paper has been used.

for i= 2:51 % Updation of weight 1 for x1
    y = pie(x1_bar(1,1:i-1));
    a = w1 + n * pie(-1*y-w1);
    w1 = [1/3 pie(a)];
end
 w1_f = w1(1,51);

for i= 2:51 % Updation of weight 1 for x1
    y = pie(x1_bar(1,1:i-1));
    a = w2 + n * pie(y-w2);
    w2 = [2/3 pie(a)];
end
w2_f = w2(1,51);

for i= 2:51  % Updation of weight 3 for x1
    y = pie(x1_bar(1,1:i-1));
    a = w3 + n * pie(-1*y-w3);
    e1 = i-1;
    if(isnan(a))
        break;
    end
    w3 = [1 pie(a)];
end

if(e1 == 50)
  w3_f = w3(1,51);
else
  w3_f = w3(1,e1);
end

 for i= 2:51  % Updation of weight 1 for x2
    y = pie(x2_bar(1,1:i-1));
    a = w_1 + n * pie(y-w_1);
    w_1 = [-1/3 pie(a)];
 end
  w_1_f = w_1(1,51);

for i= 2:51  % Updation of weight 2 for x2
    y = pie(x2_bar(1,1:i-1));
    a = w_2 + n * pie(y-w_2);
    w_2 = [-2/3 pie(a)];
end
  w_2_f = w_2(1,51);

for i= 2:51  % Updation of weight 3 for x2
    y = pie(x2_bar(1,1:i-1));
    a = w_3 + n * pie(-1*y-w_3);
    e2 = i-1;
    if(isnan(a))
        break;
```

```
    end
    w_3 = [-1 pie(a)];
end

if(e2 == 50)
  w_3_f = w_3(1,51);
else
  w_3_f = w_3(1,e2);
end

%Final mixing matrix has been obtained after getting updated weights
A_obtained = [w1_f, w2_f, w3_f ; w_1_f, w_2_f, w_3_f];

%Scaling matrix L has been initialised
L = [0.5, 0, 0 ; 0, 0.5, 0 ; 0, 0, 1];

%Permuation matrix
P = eye(3);

%Approximation matrix of A
B = A_obtained*P*L;
E = B-A;
error = norm(E);
disp(error);

%Psuedo inverse of B has been used to find the source
S_pred = pinv(B)*X;

%Plotting the obtained source signals
s11 = S_pred(1,:);
s11 = s11/max(s11);
figure,plot(s11),title('Source 1 Recovered');

s22 = S_pred(2,:);
s22 = s22/max(s22);
figure,plot(s22), title('Source 2 Recovered');

s33 = S_pred(3,:);
s33 = s33/max(s33);
figure,plot(s33),title('Source 3 Recovered');
```

```
    1.5097
```


Source 1 Recovered
1
0.5
0
-0.5
-1
-1.5
0
10
20
30
40
50
60


Source 2 Recovered
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
10
20
30
40
50
60



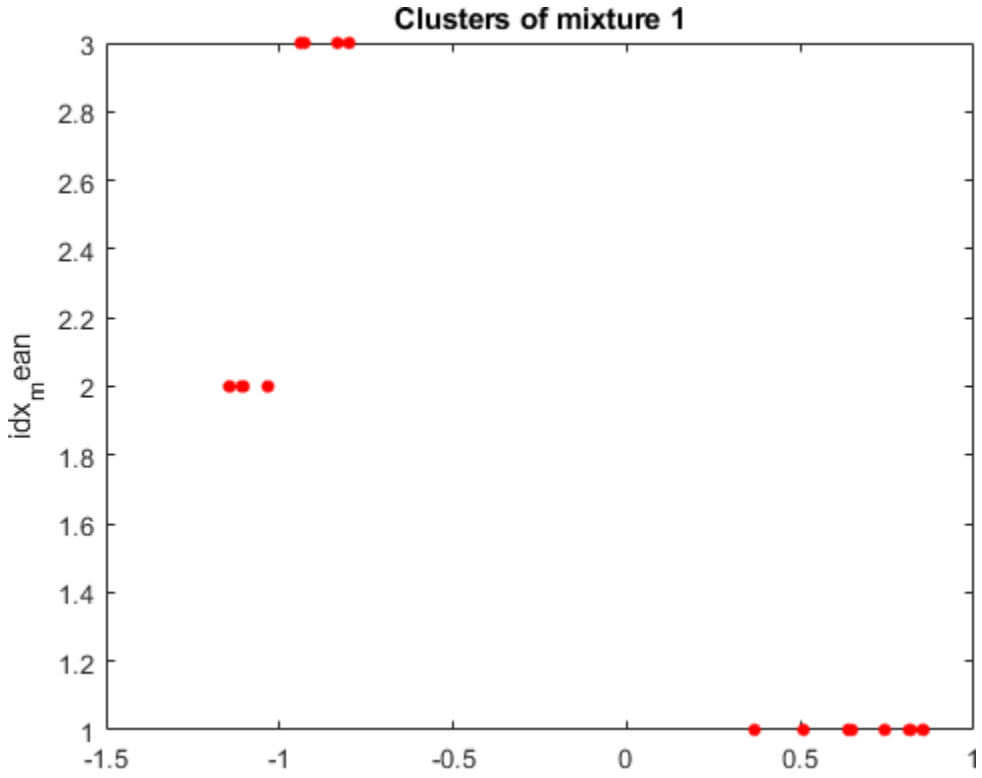
K means Clustering

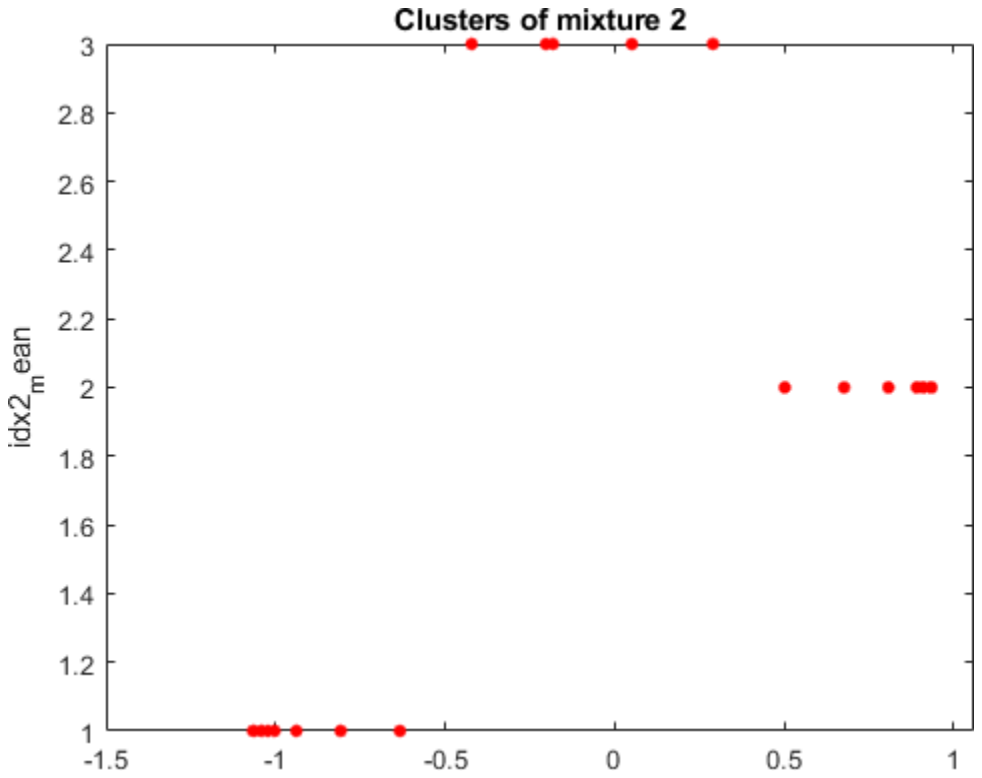
```
% Taking transpose of mixture x1
x1 = x1_bar';
% Taking transpose of mixture x2
x2 = x2_bar';

% All classification of mixture x1 are represented my the vector
 idx_mean
[idx_mean, C_mean] = kmeans(x1,3);
% All classification of mixture x2 are represented my the vector
 idx2_mean
[idx2_mean, C2_mean] = kmeans(x2,3);


% Plotting cluster of mixture x1
figure;
gscatter(x1(:,1),idx_mean),title('Clusters of mixture 1');
% Plotting cluster of mixture x2
figure;
gscatter(x2(:,1),idx2_mean),title('Clusters of mixture 2');

% Initializing frequencies of clusters in mixture x1
p1_mean = 0;
p2_mean = 0;
```

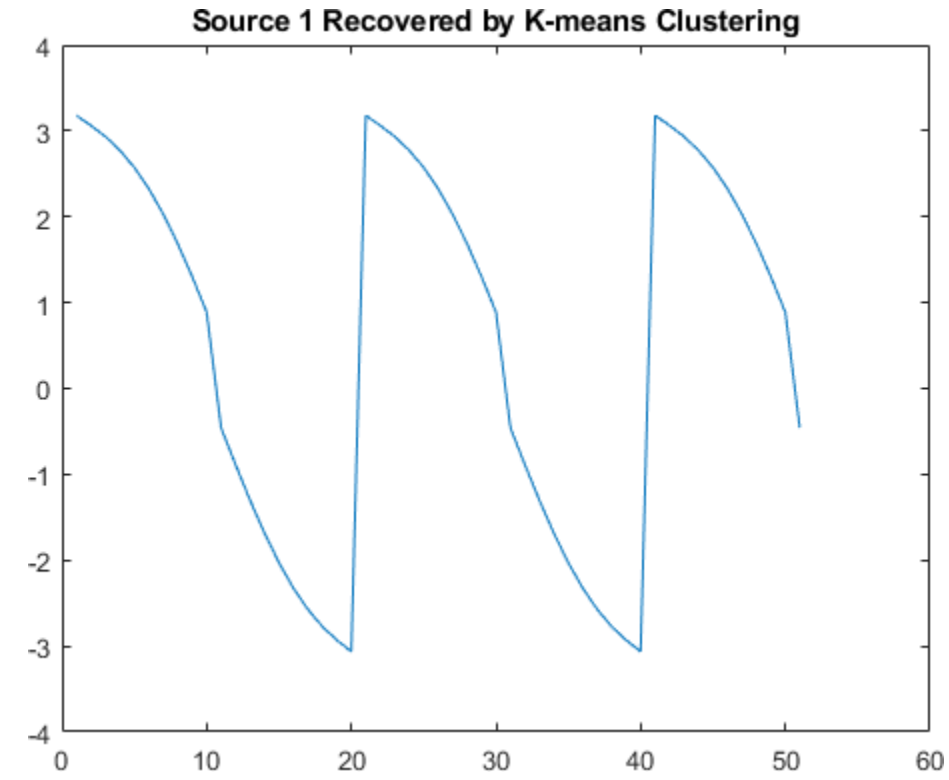
```
p3_mean = 0;

for i=1:51
    if idx_mean(i,1)==1
        p1_mean = p1_mean+1; % Calculating frequency of cluster 1 in
 mixture x1
    elseif idx_mean(i,1)==2
        p2_mean = p2_mean+1; % Calculating frequency of cluster 2 in
 mixture x1
    else
        p3_mean = p3_mean+1; % Calculating frequency of cluster 3 in
 mixture x1
    end
end

% Calculating posterior probabilities using the frequency of each
 cluster
% in the mixture x1
posterior11_mean = p1_mean/51;
posterior21_mean = p2_mean/51;
posterior31_mean = p3_mean/51;

% Initializing frequencies of clusters in mixture x2
p11_mean = 0;
p22_mean = 0;
p33_mean = 0;

for i=1:51
    if idx2_mean(i,1)==1
        p11_mean = p11_mean+1; % Calculating frequency of cluster 1 in
 mixture x2
    elseif idx2_mean(i,1)==2
        p22_mean = p22_mean+1; % Calculating frequency of cluster 2 in
 mixture x2
    else
        p33_mean = p33_mean+1; % Calculating frequency of cluster 3 in
 mixture x2
    end
end

% Calculating posterior probabilities using the frequency of each
 cluster
% in the mixture x2
posterior12_mean = p11_mean/51;
posterior22_mean = p22_mean/51;
posterior32_mean = p33_mean/51;

% Finding mixing matrix A from obtained posterior values after
 clustering
A1 = [posterior11_mean, posterior21_mean, posterior31_mean ;
 posterior12_mean, posterior22_mean, posterior32_mean];

% Scaling matrix L
L = [0.5, 0, 0 ; 0, 0.5, 0 ; 0, 0, 1];
```

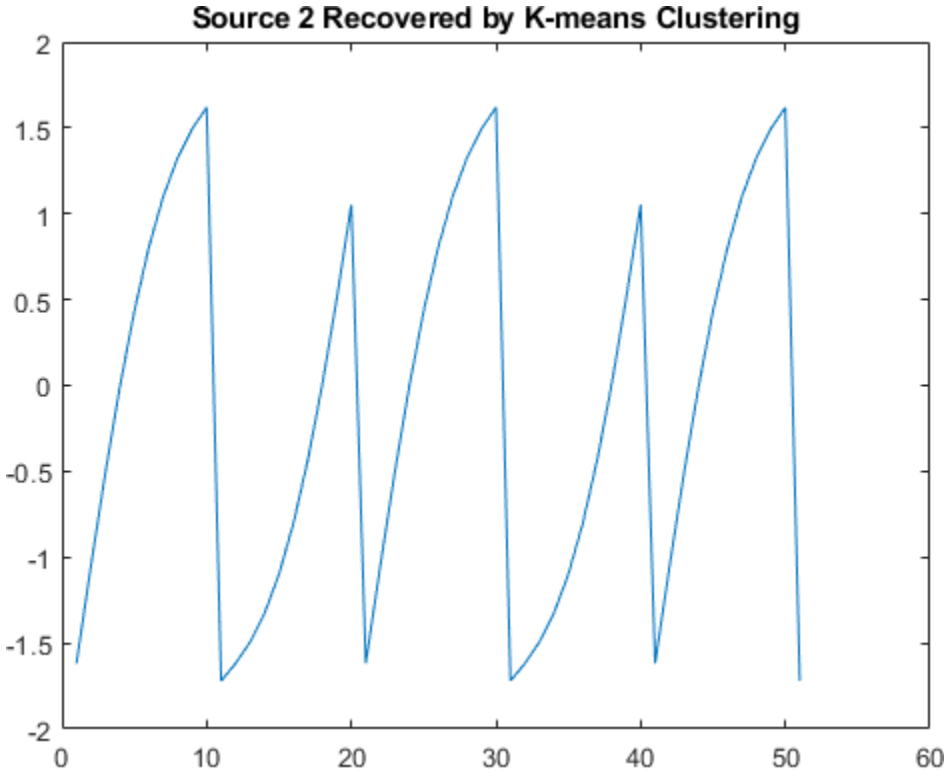
```
% Permuation matrix
P = eye(3);

% Approximation matrix of A1
B_mean = A1*P*L;
E_mean = B_mean-A;
error_mean = norm(E_mean);
disp(error_mean);

% Pseudoinverse of B has been used to approximate the source
S2_mean = pinv(B_mean)*X;

% Plotting the obtained source signals
S_11_mean = S2_mean(1,:);
figure, plot(S_11_mean),title('Source 1 Recovered by K-means
 Clustering');

S_12_mean = S2_mean(2,:);
figure, plot(S_12_mean),title('Source 2 Recovered by K-means
 Clustering');

S_13_mean = S2_mean(3,:);
figure, plot(S_13_mean),title('Source 3 Recovered by K-Means
 Clustering');
```

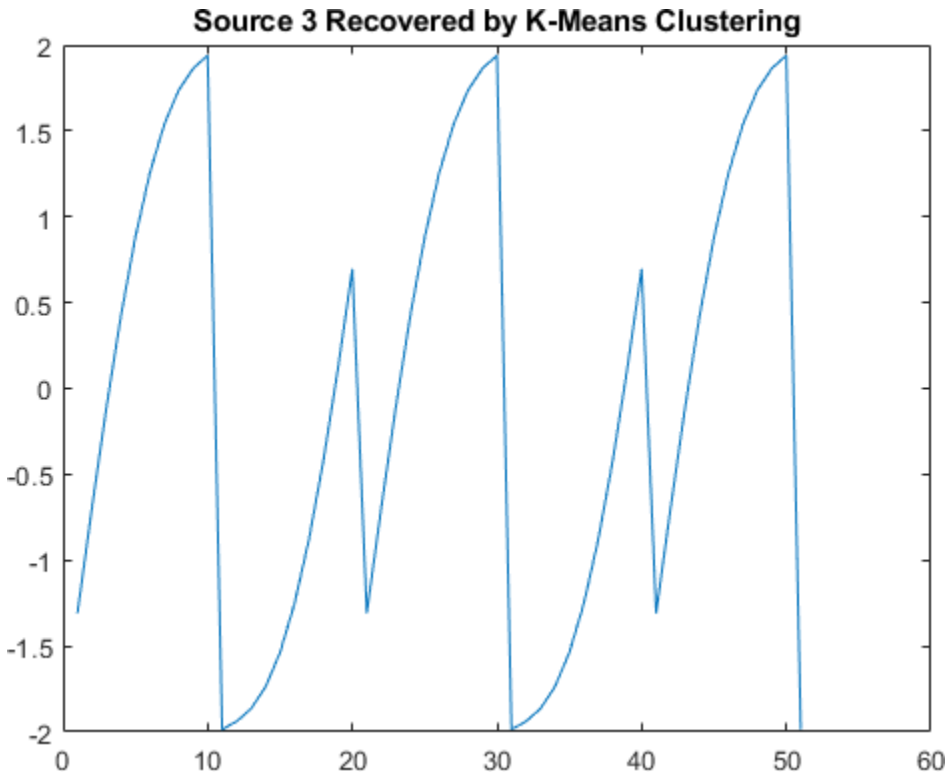
```
    1.5631
```


Clusters of mixture 1
idx_mean
3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
-1.5
-1
-0.5
0
0.5
1


Clusters of mixture 2
idx2_mean
3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
-1.5
-1
-0.5
0
0.5
1


Source 1 Recovered by K-means Clustering
4
3
2
1
0
-1
-2
-3
-4
0
10
20
30
40
50
60


Source 2 Recovered by K-means Clustering
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
10
20
30
40
50
60



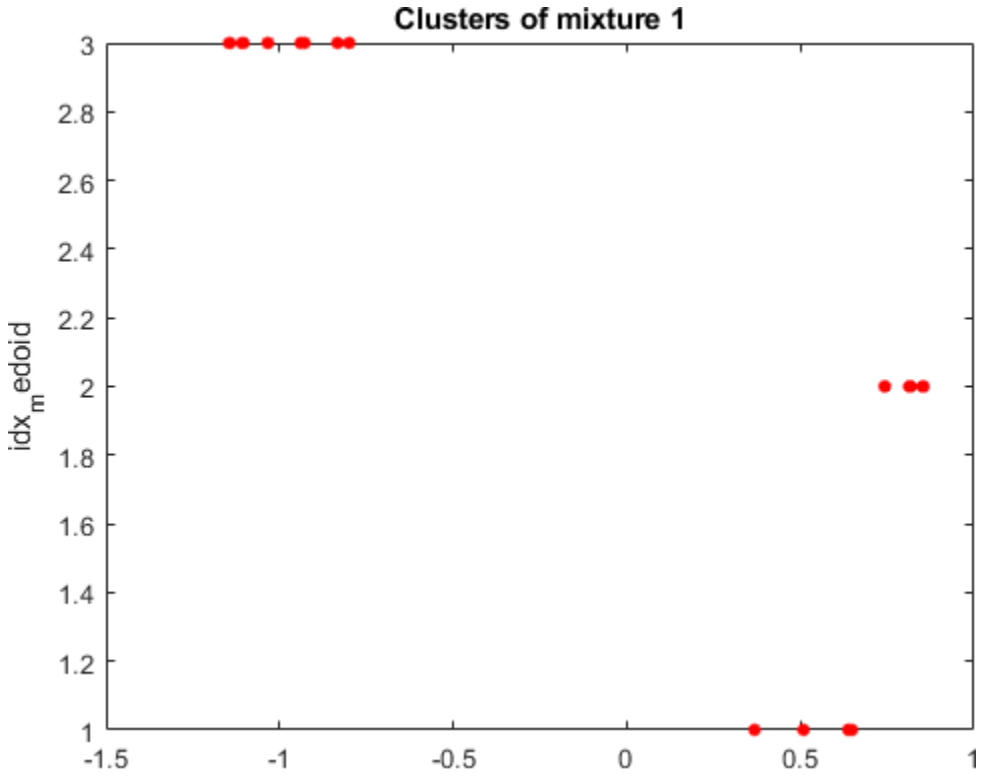
K-medoids Clustering

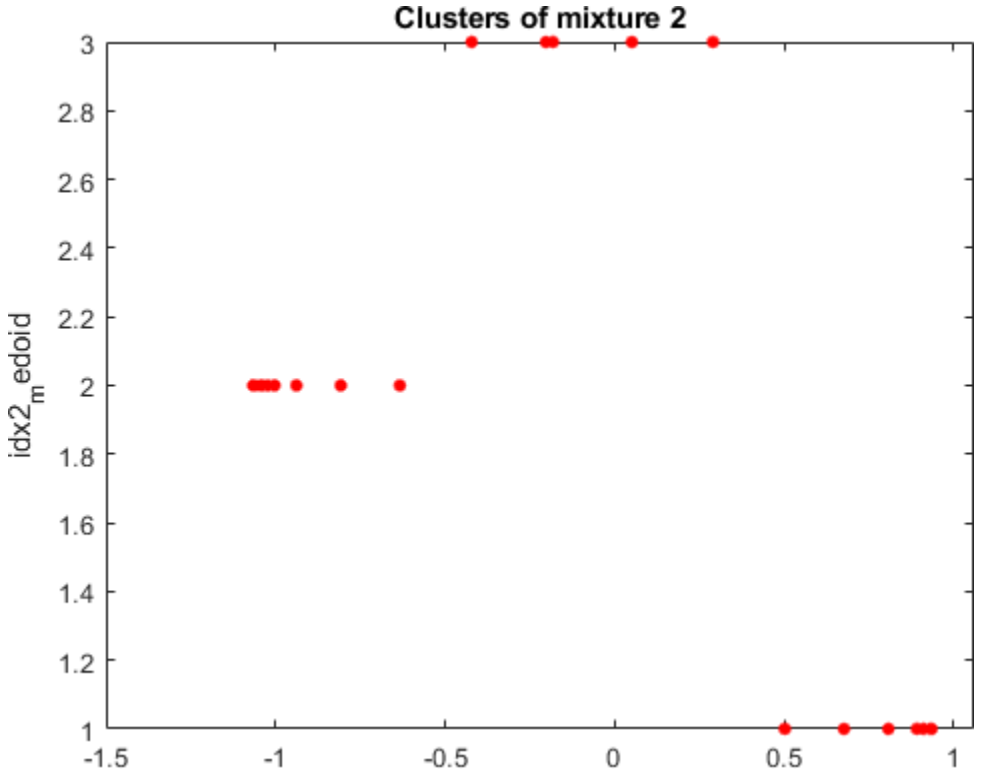
```
% Taking transpose of mixture x1
x1 = x1_bar';
% Taking transpose of mixture x2
x2 = x2_bar';

% All classification of mixture x1 are represented my the vector
 idx_medoid
[idx_medoid, C_medoid] = kmedoids(x1,3);
% All classification of mixture x2 are represented my the vector
 idx2_medoid
[idx2_medoid, C2_medoid] = kmedoids(x2,3);


% Plotting cluster of mixture x1
figure;
gscatter(x1(:,1),idx_medoid),title('Clusters of mixture 1');
% Plotting cluster of mixture x2
figure;
gscatter(x2(:,1),idx2_medoid),title('Clusters of mixture 2');

% Initializing frequencies of clusters in mixture x1
p1_medoid = 0;
p2_medoid = 0;
```

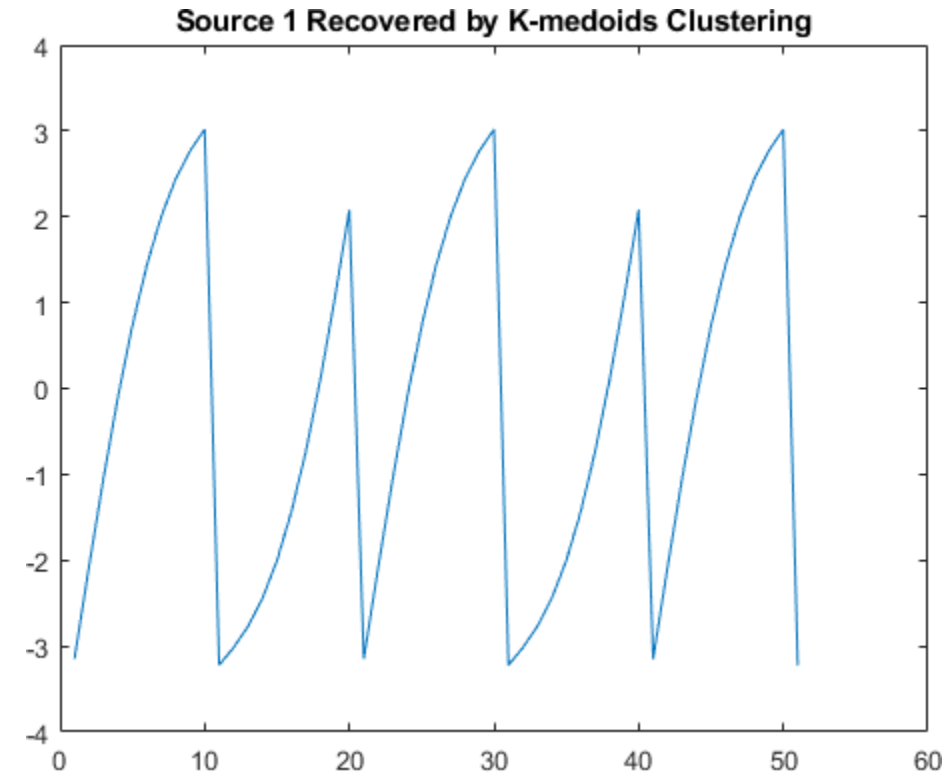
```
p3_medoid = 0;

for i=1:51
    if idx_medoid(i,1)==1
        p1_medoid = p1_medoid+1; % Calculating frequency of cluster 1
 in mixture x1
    elseif idx_medoid(i,1)==2
        p2_medoid = p2_medoid+1; % Calculating frequency of cluster 2
 in mixture x1
    else
        p3_medoid = p3_medoid+1; % Calculating frequency of cluster 3
 in mixture x1
    end
end

% Calculating posterior probabilities using the frequency of each
 cluster
% in the mixture x1
posterior11_medoid = p1_medoid/51;
posterior21_medoid = p2_medoid/51;
posterior31_medoid = p3_medoid/51;

% Initializing frequencies of clusters in mixture x2
p11_medoid = 0;
p22_medoid = 0;
p33_medoid = 0;

for i=1:51
    if idx2_medoid(i,1)==1
        p11_medoid = p11_medoid+1; % Calculating frequency of cluster
 1 in mixture x2
    elseif idx2_medoid(i,1)==2
        p22_medoid = p22_medoid+1; % Calculating frequency of cluster
 2 in mixture x2
    else
        p33_medoid = p33_medoid+1; % Calculating frequency of cluster
 3 in mixture x2
    end
end

% Calculating posterior probabilities using the frequency of each
 cluster
% in the mixture x2
posterior12_medoid = p11_medoid/51;
posterior22_medoid = p22_medoid/51;
posterior32_medoid = p33_medoid/51;

% Finding mixing matrix A from obtained posterior values after
 clustering
A1 = [posterior11_medoid, posterior21_medoid, posterior31_medoid ;
 posterior12_medoid, posterior22_medoid, posterior32_medoid];

% Scaling matrix L
L = [0.5, 0, 0 ; 0, 0.5, 0 ; 0, 0, 1];
```

```
% Permuation matrix
P = eye(3);

% Approximation matrix of A1
B_medoid = A1*P*L;
E_medoid = B_medoid-A;
error_medoid = norm(E_medoid);
disp(error_medoid);

% Pseudoinverse of B has been used to approximate the source
S2_medoid = pinv(B_medoid)*X;

% Plotting the obtained source signals
S_11_medoid = S2_medoid(1,:);
figure, plot(S_11_medoid),title('Source 1 Recovered by K-medoids
 Clustering');

S_12_medoid = S2_medoid(2,:);
figure, plot(S_12_medoid),title('Source 2 Recovered by K-medoids
 Clustering');

S_13_medoid = S2_medoid(3,:);
figure, plot(S_13_medoid),title('Source 3 Recovered by K-medoids
 Clustering');
```

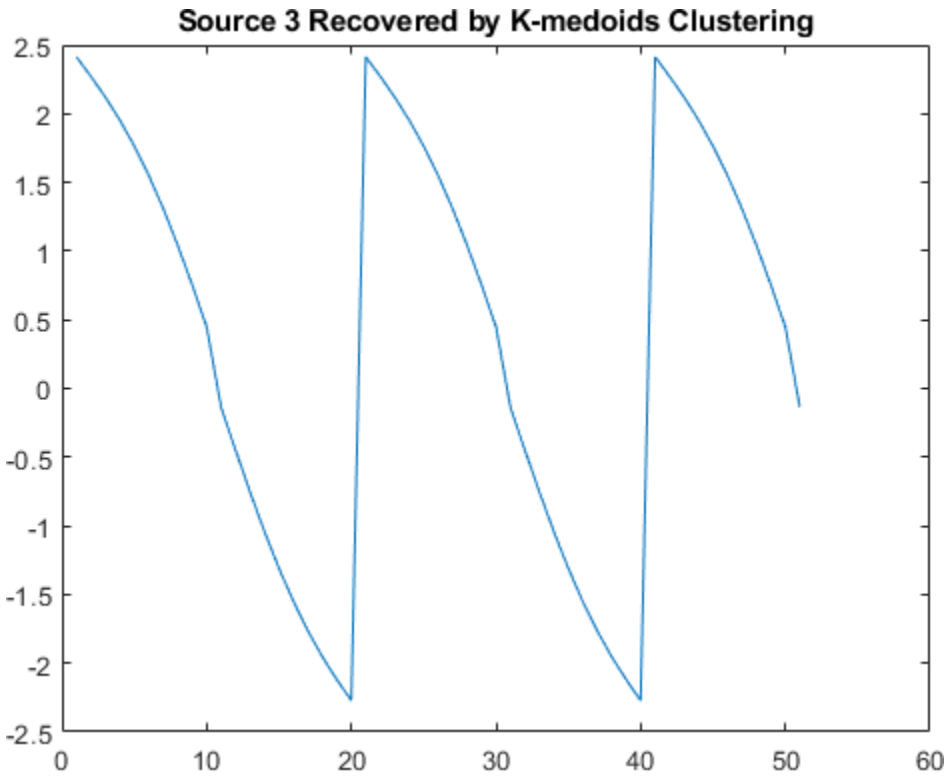
```
    1.5496
```


Clusters of mixture 1
idx_medoid
3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
-1.5
-1
-0.5
0
0.5
1


Clusters of mixture 2
idx2_medoid
3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
-1.5
-1
-0.5
0
0.5
1


Source 1 Recovered by K-medoids Clustering
4
3
2
1
0
-1
-2
-3
-4
0
10
20
30
40
50
60


Source 2 Recovered by K-medoids Clustering
1.5
1
0.5
0
-0.5
-1
-1.5
0
10
20
30
40
50
60



Published with MATLAB® R2018a

Executing the code 1000 times to determine the winning algorithm

```
error_a = []; % empty matrix to store error in In-Paper algorithm (in
 1000 iterations)
error_mean_a = []; % empty matrix to store error in inbuilt K means
 algorithm (in 1000 iterations)
error_medoid_a = []; % empty matrix to store error in inbuilt K medoid
 algorithm (in 1000 iterations)
for i = 1:1000 % loop for 1000 iterations

% Defining the time period and frequency for the signals.
    tp = 0:0.1:5;
    freq = 0.5;

% Genrating 3 signals. Three types of waveforms have been used -
% Sine, Sawtooth and Square
    signal1 = sin(2*pi*freq*tp);
    signal2 = sawtooth(2*pi*freq*tp);
    signal3 = square(2*pi*freq*tp);

% Generating two mixtures of signals by multiplying each signal with a
% random number and adding them up.
    x1 = rand(1)*signal1 + rand(1)*signal2 + rand(1)*signal3;
    x1 = x1/max(x1);
    x2 = rand(1)*signal1 + rand(1)*signal2 + rand(1)*signal3;
    x2 = x2/max(x2);
    x1_bar = x1 - mean(x1);
    x2_bar = x2 - mean(x2);

% Combining the mixtures into a matrix
    X = [x1 ; x2];

% Combining the sources into a matrix
    S = [signal1 ; signal2 ; signal3];
% Obtaining Signals back from given Mixtures

% A has been initialised with the values suggested in the paper
% Columns of A i.e ai = (cos(alphai), sin(alphai)) with alpha between
% 0 and pi.
    alpha1 = pi/4;
    alpha2 = pi/2;
    alpha3 = 3*pi/4;
    A = [cos(alpha1), cos(alpha2), cos(alpha3); -cos(alpha1), -
cos(alpha2), sin(alpha3)];

% Initializing Weights(Neurons) as suggested in the paper

    w1 = [cos(alpha1)];
    w_1 = [-cos(alpha1)];
    w2 = [cos(alpha2)];
```

```
    w_2 = [-cos(alpha2)];
    w3 = [cos(alpha3)];
    w_3 = [sin(alpha3)];

% Learning rate
    n = 0.5;

% There are a total of 6 weights in the mixing matrix and each weight
% has been updated 50 times to obtain suitable values of weights for
 further
% calculations.
% Pie function as mentioned in the paper has been used.

    for i= 2:51 % Updation of weight 1 for x1
        y = pie(x1_bar(1,1:i-1));
        a = w1 + n * pie(-1*y-w1);
        w1 = [1/3 pie(a)];
    end
    w1_f = w1(1,51);

    for i= 2:51 % Updation of weight 1 for x1
        y = pie(x1_bar(1,1:i-1));
        a = w2 + n * pie(y-w2);
        w2 = [2/3 pie(a)];
    end
    w2_f = w2(1,51);

    for i= 2:51  % Updation of weight 3 for x1
        y = pie(x1_bar(1,1:i-1));
        a = w3 + n * pie(-1*y-w3);
        e1 = i-1;
        if(isnan(a))
            break;
        end
        w3 = [1 pie(a)];
    end

    if(e1 == 50)
    w3_f = w3(1,51);
    else
    w3_f = w3(1,e1);
    end

    for i= 2:51  % Updation of weight 1 for x2
        y = pie(x2_bar(1,1:i-1));
        a = w_1 + n * pie(y-w_1);
        w_1 = [-1/3 pie(a)];
    end
    w_1_f = w_1(1,51);

    for i= 2:51  % Updation of weight 2 for x2
        y = pie(x2_bar(1,1:i-1));
        a = w_2 + n * pie(y-w_2);
        w_2 = [-2/3 pie(a)];
    end
```

```
    w_2_f = w_2(1,51);

    for i= 2:51  % Updation of weight 3 for x2
        y = pie(x2_bar(1,1:i-1));
        a = w_3 + n * pie(-1*y-w_3);
        e2 = i-1;
        if(isnan(a))
            break;
        end
        w_3 = [-1 pie(a)];
    end
    if(e2 == 50)
    w_3_f = w_3(1,51);
    else
    w_3_f = w_3(1,e2);
    end

%Final mixing matrix has been obtained after getting updated weights
    A_obtained = [w1_f, w2_f, w3_f ; w_1_f, w_2_f, w_3_f];

%Scaling matrix L has been initialised
    L = [0.5, 0, 0 ; 0, 0.5, 0 ; 0, 0, 1];

%Permuation matrix
    P = eye(3);

%Approximation matrix of A
    B = A_obtained*P*L;
    E = B-A;
    error = norm(E);
    error_a = [error_a, error];
    %disp(error);

%Psuedo inverse of B has been used to find the source
    S_pred = pinv(B)*X;

% K means Clustering

% Taking transpose of mixture x1
    x1 = x1_bar';
% Taking transpose of mixture x2
    x2 = x2_bar';

% All classification of mixture x1 are represented my the vector
 idx_mean
    [idx_mean, C_mean] = kmeans(x1,3);
% All classification of mixture x2 are represented my the vector
 idx2_mean
    [idx2_mean, C2_mean] = kmeans(x2,3);

% Initializing frequencies of clusters in mixture x1
    p1_mean = 0;
    p2_mean = 0;
    p3_mean = 0;
```

```
    for i=1:51
        if idx_mean(i,1)==1
            p1_mean = p1_mean+1; % Calculating frequency of cluster 1
 in mixture x1
        elseif idx_mean(i,1)==2
            p2_mean = p2_mean+1; % Calculating frequency of cluster 2
 in mixture x1
        else
            p3_mean = p3_mean+1; % Calculating frequency of cluster 3
 in mixture x1
        end
    end

% Calculating posterior probabilities using the frequency of each
 cluster
% in the mixture x1
    posterior11_mean = p1_mean/51;
    posterior21_mean = p2_mean/51;
    posterior31_mean = p3_mean/51;

% Initializing frequencies of clusters in mixture x2
    p11_mean = 0;
    p22_mean = 0;
    p33_mean = 0;

    for i=1:51
        if idx2_mean(i,1)==1
            p11_mean = p11_mean+1; % Calculating frequency of cluster
 1 in mixture x2
        elseif idx2_mean(i,1)==2
            p22_mean = p22_mean+1; % Calculating frequency of cluster
 2 in mixture x2
        else
            p33_mean = p33_mean+1; % Calculating frequency of cluster
 3 in mixture x2
        end
    end

% Calculating posterior probabilities using the frequency of each
 cluster
% in the mixture x2
    posterior12_mean = p11_mean/51;
    posterior22_mean = p22_mean/51;
    posterior32_mean = p33_mean/51;

% Finding mixing matrix A from obtained posterior values after
 clustering
    A1 = [posterior11_mean, posterior21_mean, posterior31_mean ;
 posterior12_mean, posterior22_mean, posterior32_mean];

% Scaling matrix L
    L = [0.5, 0, 0 ; 0, 0.5, 0 ; 0, 0, 1];
```

```
% Permuation matrix
    P = eye(3);

% Approximation matrix of A1
    B_mean = A1*P*L;
    E_mean = B_mean-A;
    error_mean = norm(E_mean);
    error_mean_a = [error_mean_a , error_mean];
    %disp(error_mean);

% Pseudoinverse of B has been used to approximate the source
    S2_mean = pinv(B_mean)*X;

% K-medoids Clustering

% Taking transpose of mixture x1
    x1 = x1_bar';
% Taking transpose of mixture x2
    x2 = x2_bar';

% All classification of mixture x1 are represented my the vector
 idx_medoid
    [idx_medoid, C_medoid] = kmedoids(x1,3);
% All classification of mixture x2 are represented my the vector
 idx2_medoid
    [idx2_medoid, C2_medoid] = kmedoids(x2,3);

% Initializing frequencies of clusters in mixture x1
    p1_medoid = 0;
    p2_medoid = 0;
    p3_medoid = 0;

    for i=1:51
        if idx_medoid(i,1)==1
            p1_medoid = p1_medoid+1; % Calculating frequency of
 cluster 1 in mixture x1
        elseif idx_medoid(i,1)==2
            p2_medoid = p2_medoid+1; % Calculating frequency of
 cluster 2 in mixture x1
        else
            p3_medoid = p3_medoid+1; % Calculating frequency of
 cluster 3 in mixture x1
        end
    end

% Calculating posterior probabilities using the frequency of each
 cluster
% in the mixture x1
    posterior11_medoid = p1_medoid/51;
    posterior21_medoid = p2_medoid/51;
    posterior31_medoid = p3_medoid/51;

% Initializing frequencies of clusters in mixture x2
    p11_medoid = 0;
```

```
    p22_medoid = 0;
    p33_medoid = 0;

    for i=1:51
        if idx2_medoid(i,1)==1
            p11_medoid = p11_medoid+1; % Calculating frequency of
 cluster 1 in mixture x2
        elseif idx2_medoid(i,1)==2
            p22_medoid = p22_medoid+1; % Calculating frequency of
 cluster 2 in mixture x2
        else
            p33_medoid = p33_medoid+1; % Calculating frequency of
 cluster 3 in mixture x2
        end
    end

% Calculating posterior probabilities using the frequency of each
 cluster
% in the mixture x2
    posterior12_medoid = p11_medoid/51;
    posterior22_medoid = p22_medoid/51;
    posterior32_medoid = p33_medoid/51;

% Finding mixing matrix A from obtained posterior values after
 clustering
    A1 = [posterior11_medoid, posterior21_medoid, posterior31_medoid ;
 posterior12_medoid, posterior22_medoid, posterior32_medoid];

% Scaling matrix L
    L = [0.5, 0, 0 ; 0, 0.5, 0 ; 0, 0, 1];

% Permuation matrix
    P = eye(3);

% Approximation matrix of A1
    B_medoid = A1*P*L;
    E_medoid = B_medoid-A;
    error_medoid = norm(E_medoid);
    error_medoid_a = [error_medoid_a, error_medoid];
    %disp(error_medoid);

% Pseudoinverse of B has been used to approximate the source
    S2_medoid = pinv(B_medoid)*X;

end

algo_count = 0; % initializing the count to track how many times the
 In-Paper Algo wins
mean_count = 0; % initializing the count to track how many times k
 means wins
medoid_count = 0; % initializing the count to track how many times K
 medoids wins
for i=1:1000 % logic to find the least error obtained in one
 particular iteration and doing the same for 1000 iterations
```

```
    if(error_a(1,i) <= error_mean_a(1,i) && error_a(1,i) <=
 error_medoid_a(1,i))
        algo_count = algo_count+1;
    elseif(error_mean_a(1,i) <= error_a(1,i) && error_mean_a(1,i) <=
 error_medoid_a(1,i))
        mean_count = mean_count+1;
    elseif(error_medoid_a(1,i) <= error_a(1,i) && error_medoid_a(1,i)
 <= error_mean_a(1,i))
        medoid_count = medoid_count+1;
    end
end

% Calculating the Probability of how many times each algorithm wins
p_algo = (algo_count)/1000;
p_mean = (mean_count)/1000;
p_medoid = (medoid_count)/1000;

% Printing the obtained probabilities
fprintf('Probability with which the In-Paper algorithm gives the least
 error : %0.3f\n\n', p_algo);
fprintf('Probability with which K-means algorithm gives the least
 error : %0.3f\n\n', p_mean);
fprintf('Probability with which K-medoid algorithm gives the least
 error : %0.3f\n\n', p_medoid);
```

```
Probability with which the In-Paper algorithm gives the least error :
 0.441

Probability with which K-means algorithm gives the least error : 0.267

Probability with which K-medoid algorithm gives the least error :
 0.292
```

Published with MATLAB® R2018a