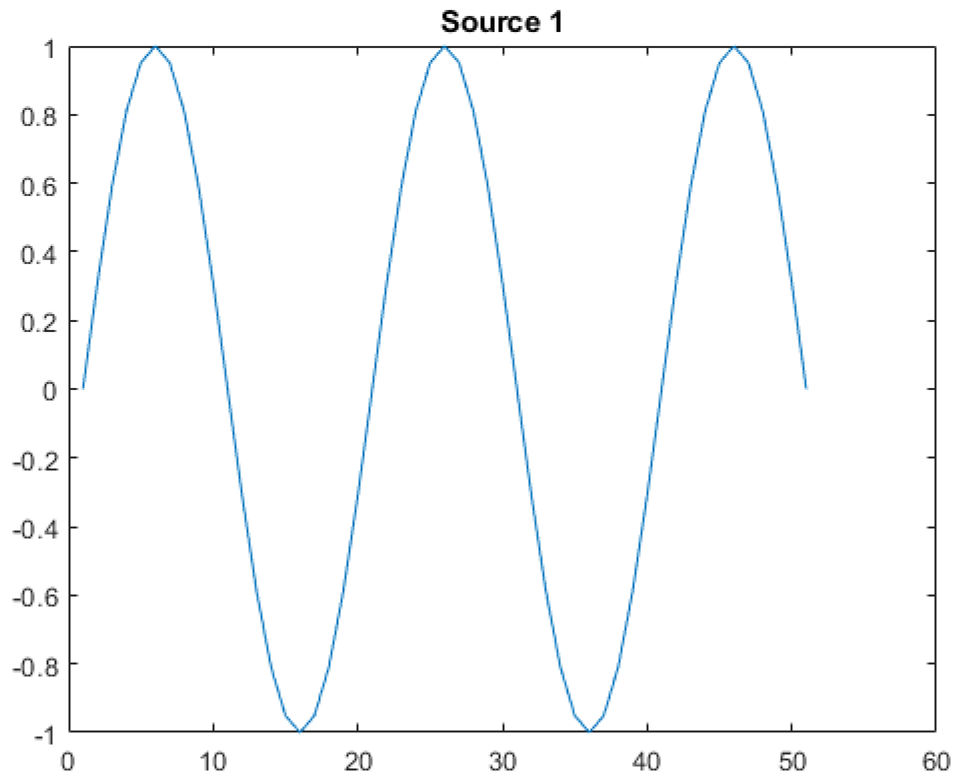
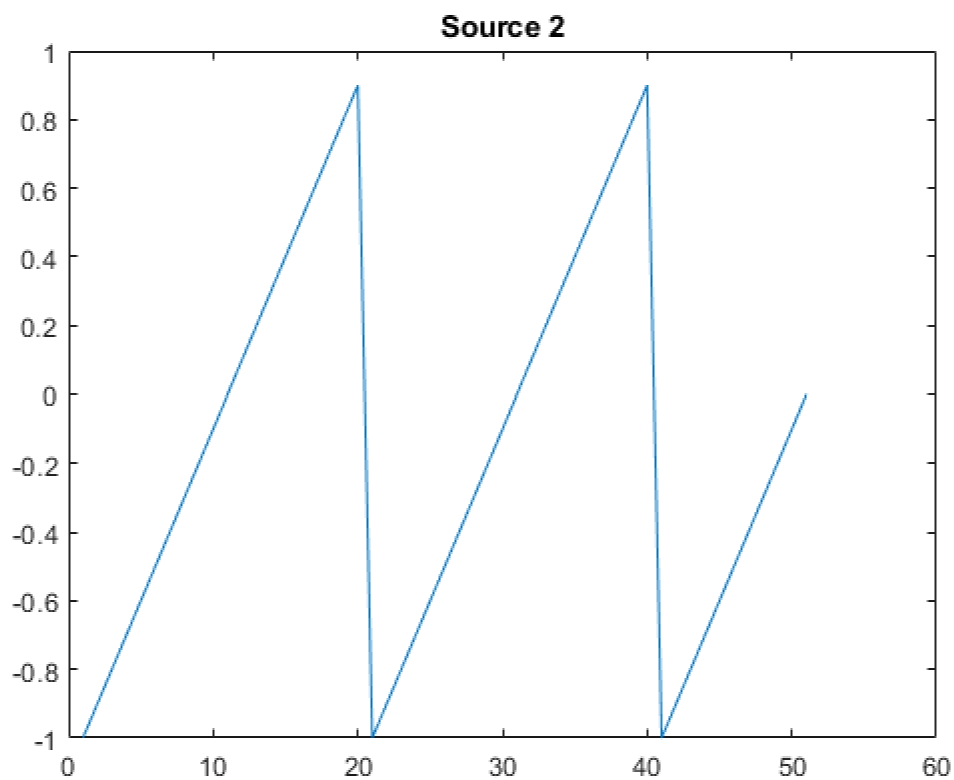
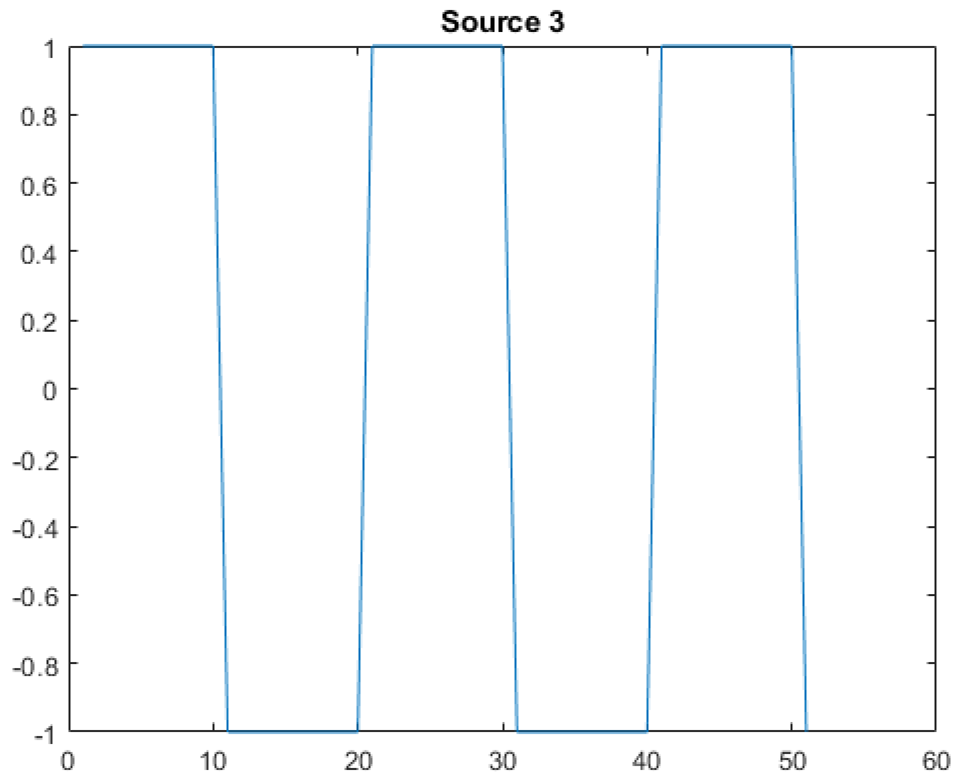
Table of Contents

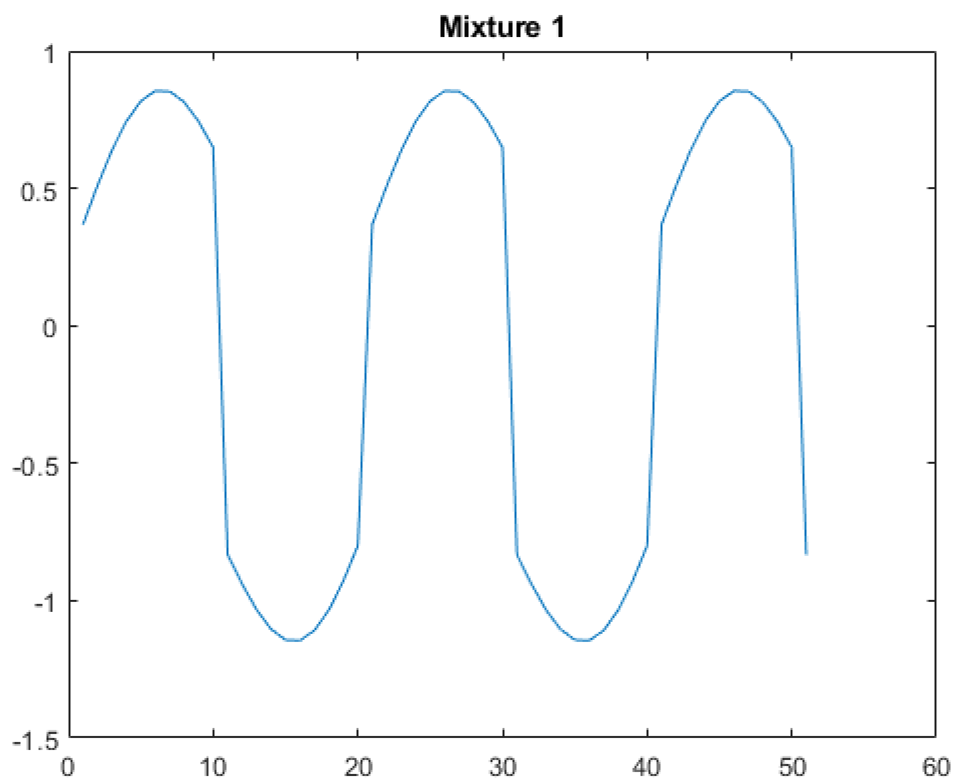
Generating Signals and Mitures

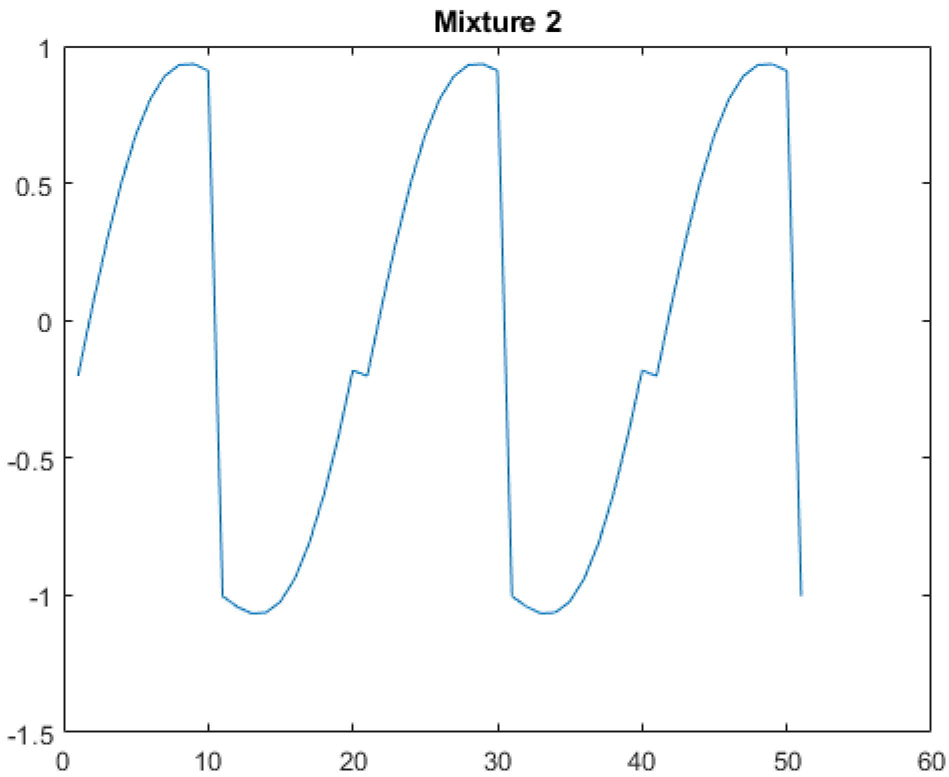
```
clc;
clear all;
close all;

% 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



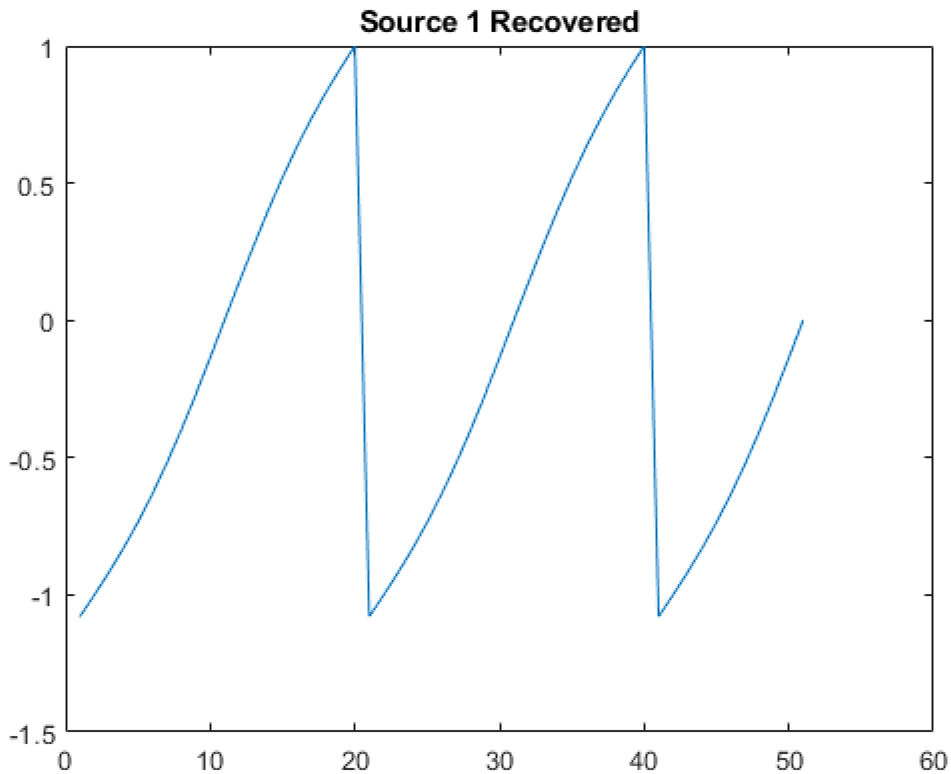
Obtaining Signals back from given Mixtures

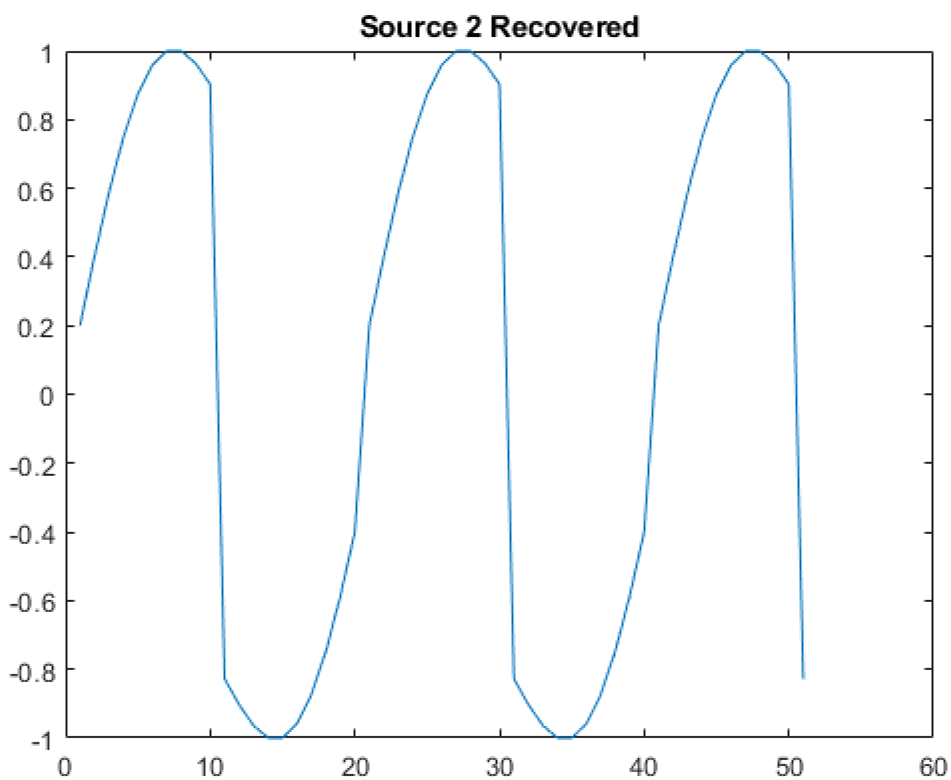
```
% A has been initialised with the values suggested in the paper
% Columns of A i.e ai = (cos(alphai), sin(alphai)) with alpha between
% 0 and pi.
alpha1 = pi/4;
alpha2 = pi/2;
alpha3 = 3*pi/4;
A = [cos(alpha1), cos(alpha2), cos(alpha3); -cos(alpha1), -
cos(alpha2), sin(alpha3)];

% 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
```

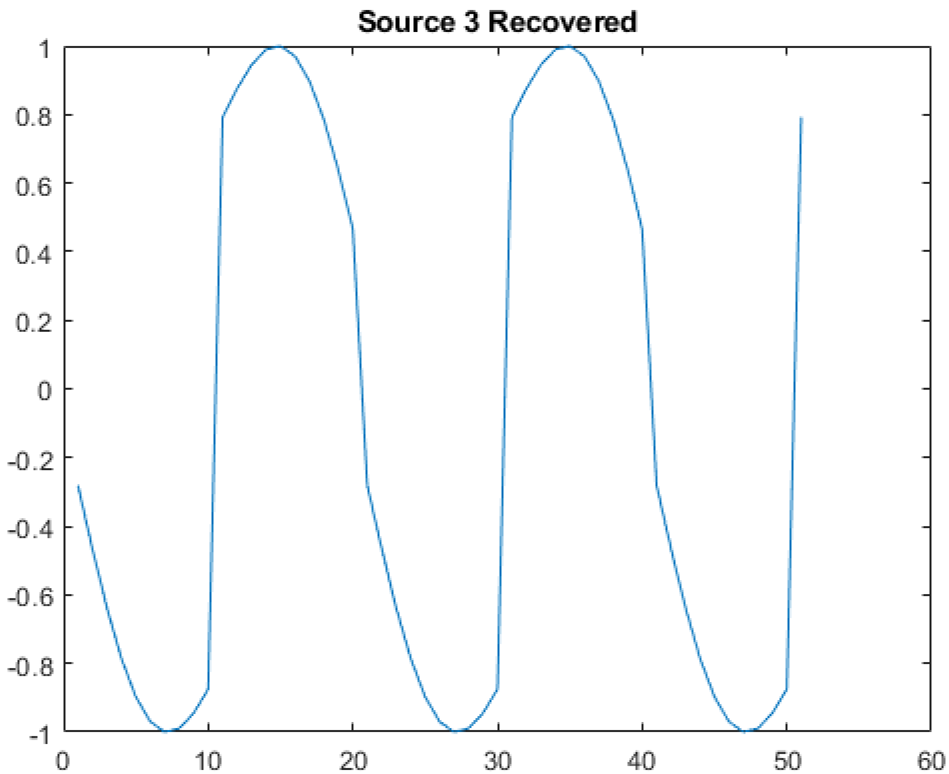
```
% has been updated 50 times to obtain suitable values of weights for
 further
% calculations.
% Pie function as mentioned in the paper has been used.

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



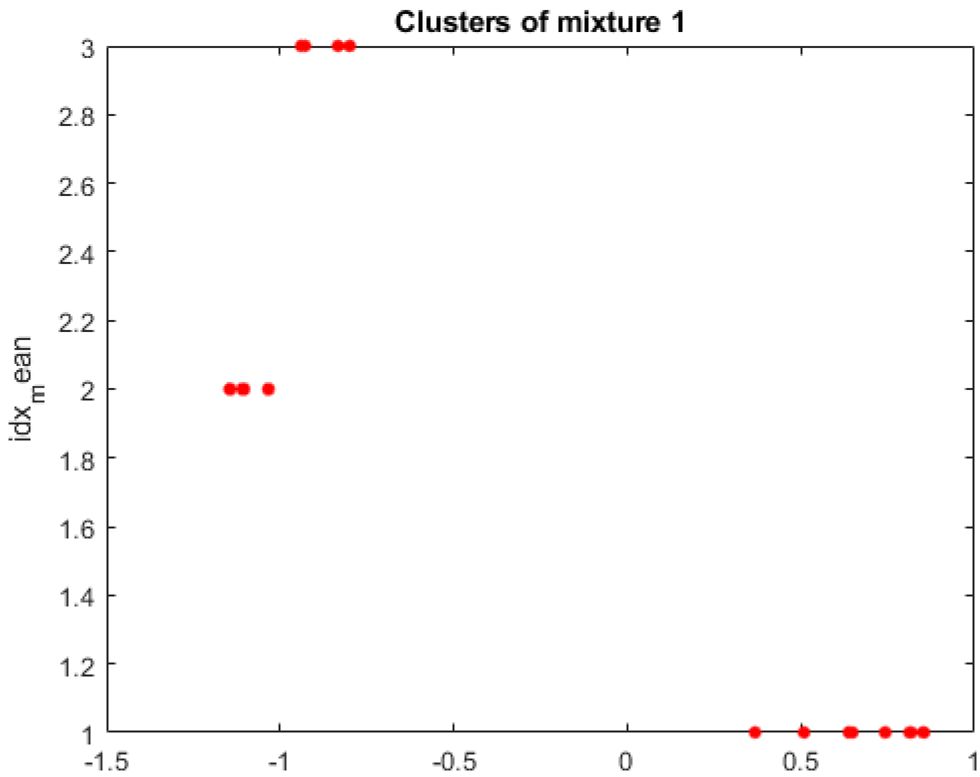
K means Clustering

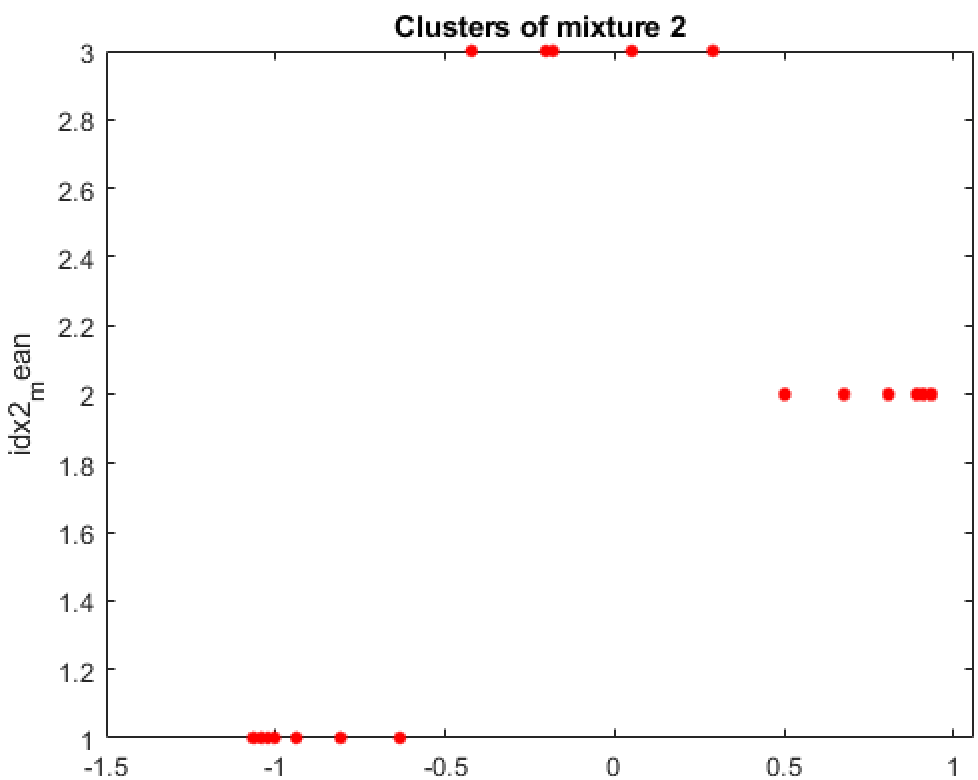
```
% Taking transpose of mixture x1
x1 = x1_bar';
% Taking transpose of mixture x2
x2 = x2_bar';

% 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;
```

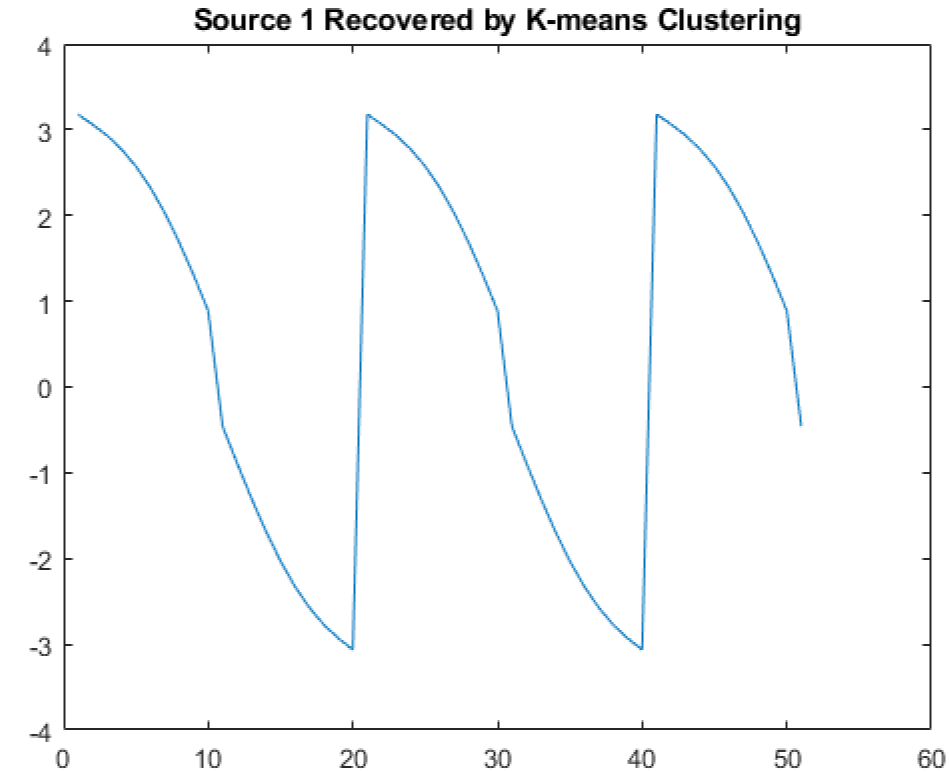
```
p3_mean = 0;

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];
```

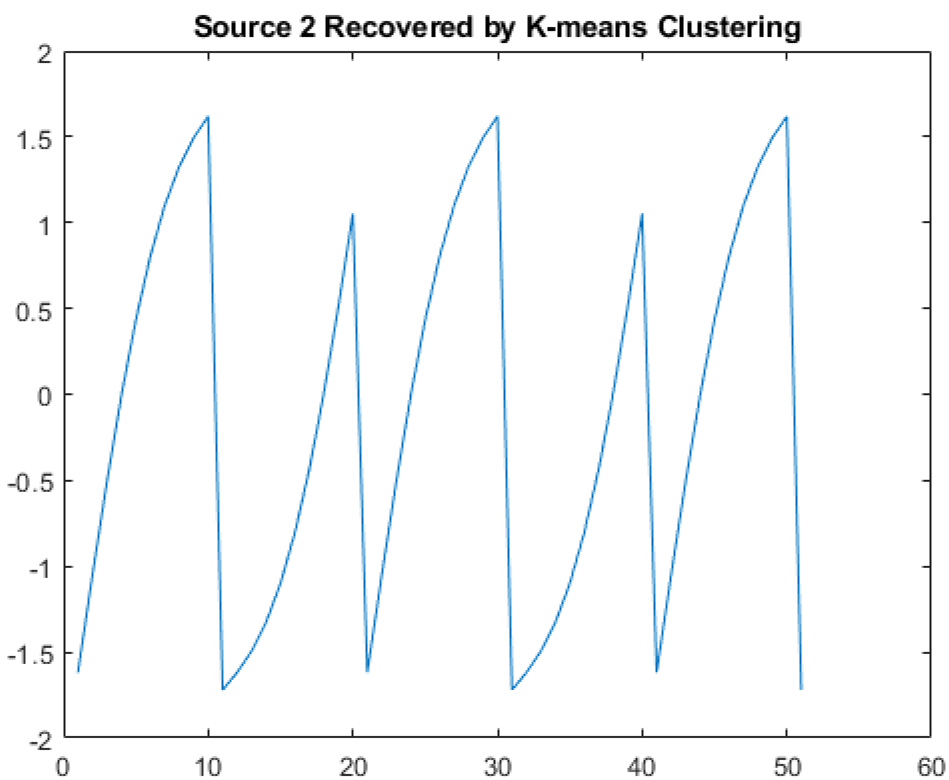
```
% Permuation matrix
P = eye(3);

% 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');
```

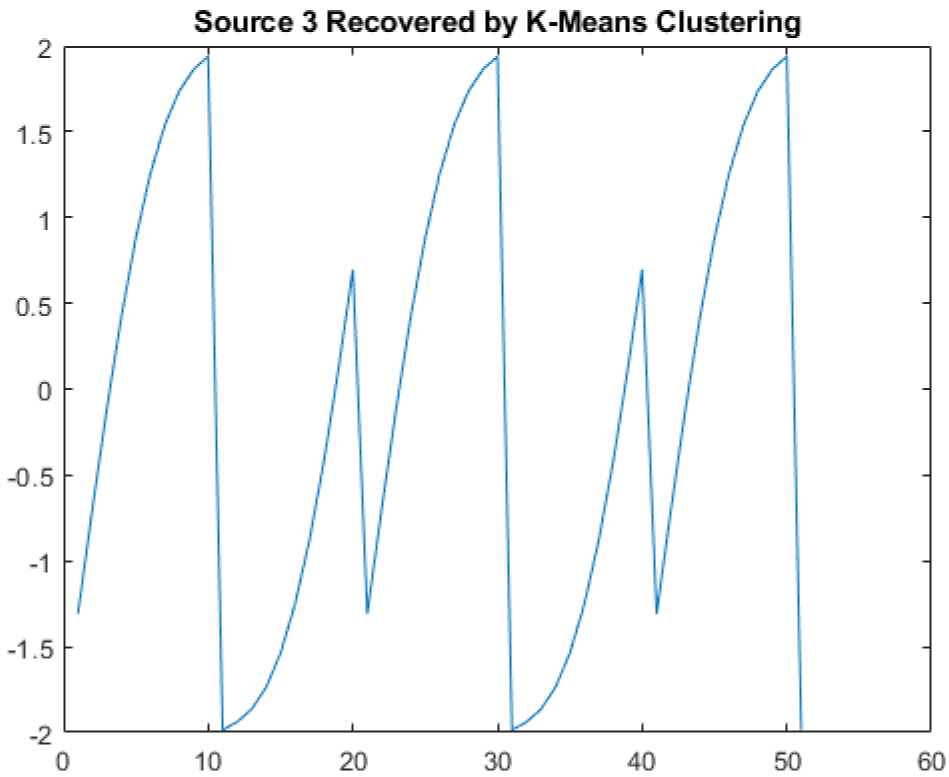
```
    1.5631
```


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



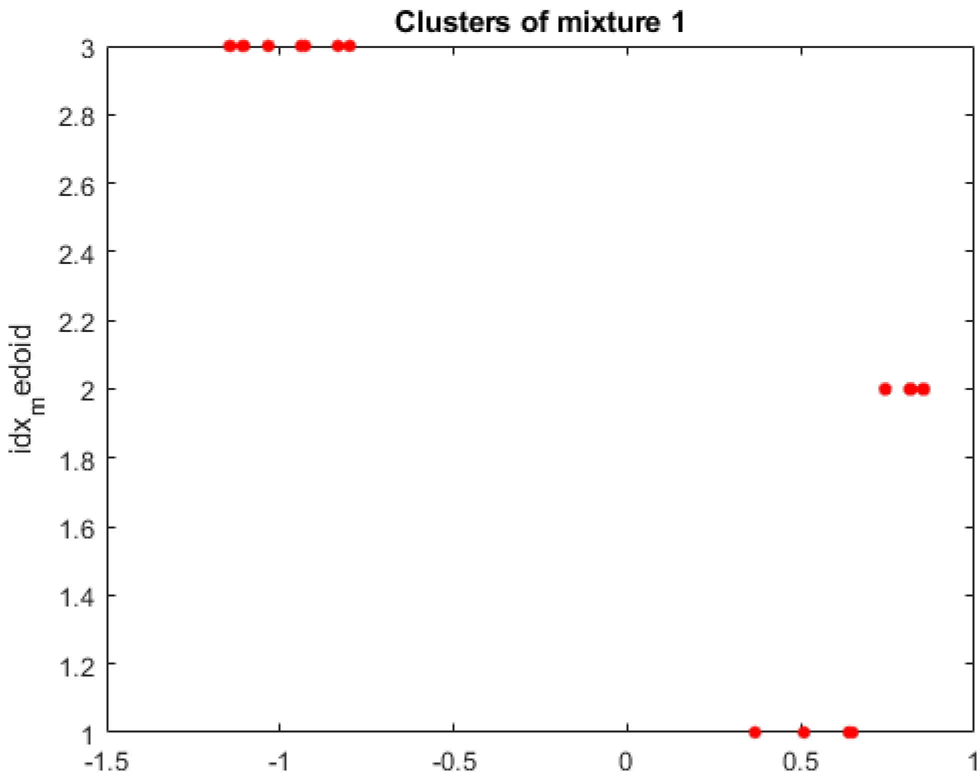
K-medoids Clustering

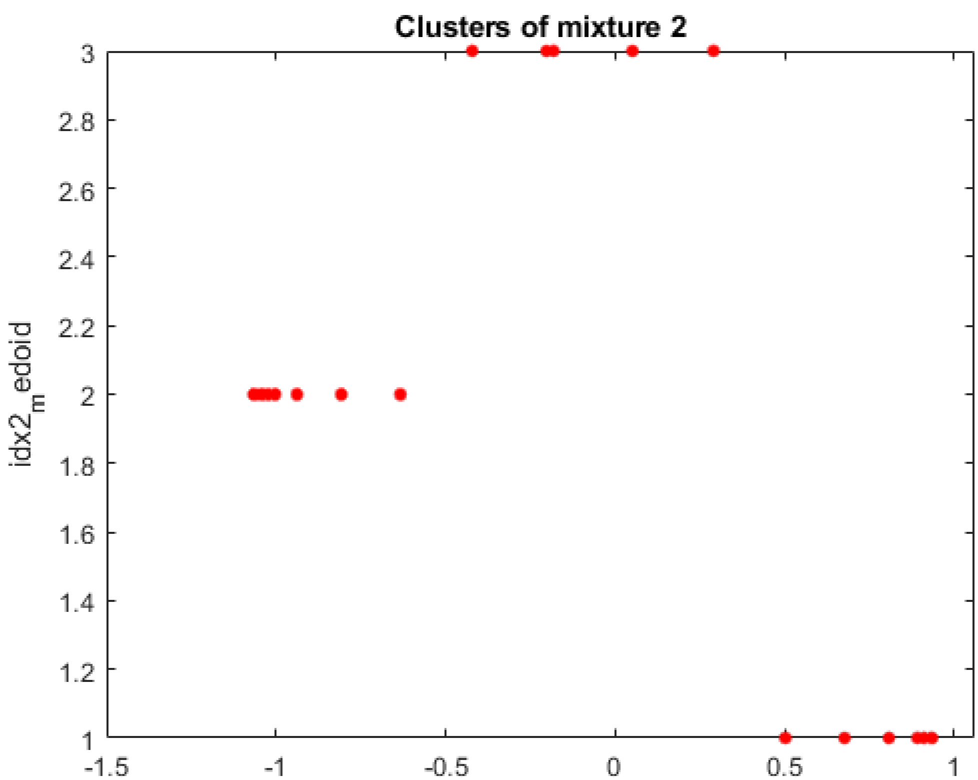
```
% Taking transpose of mixture x1
x1 = x1_bar';
% Taking transpose of mixture x2
x2 = x2_bar';

% 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;
```

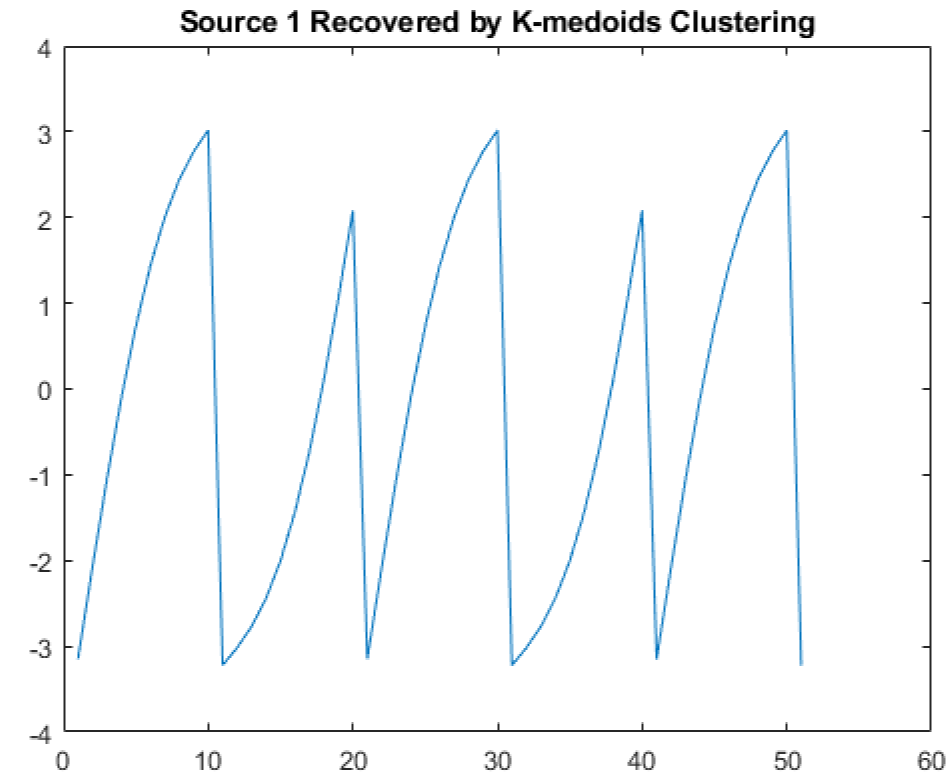
```
p3_medoid = 0;

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];
```

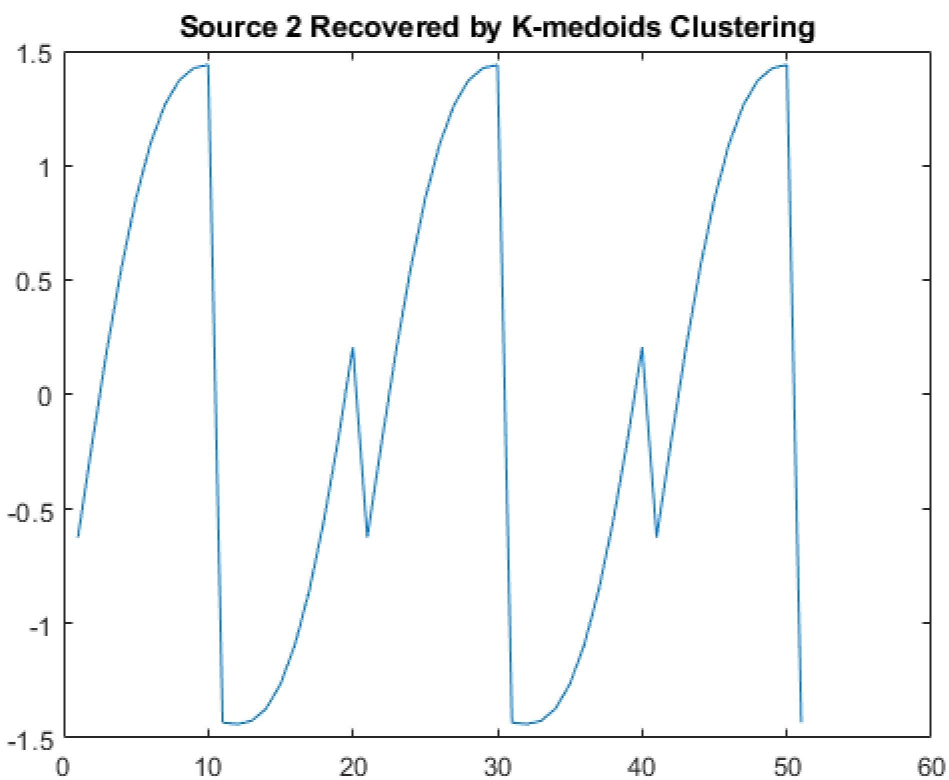
```
% Permuation matrix
P = eye(3);

% 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');
```

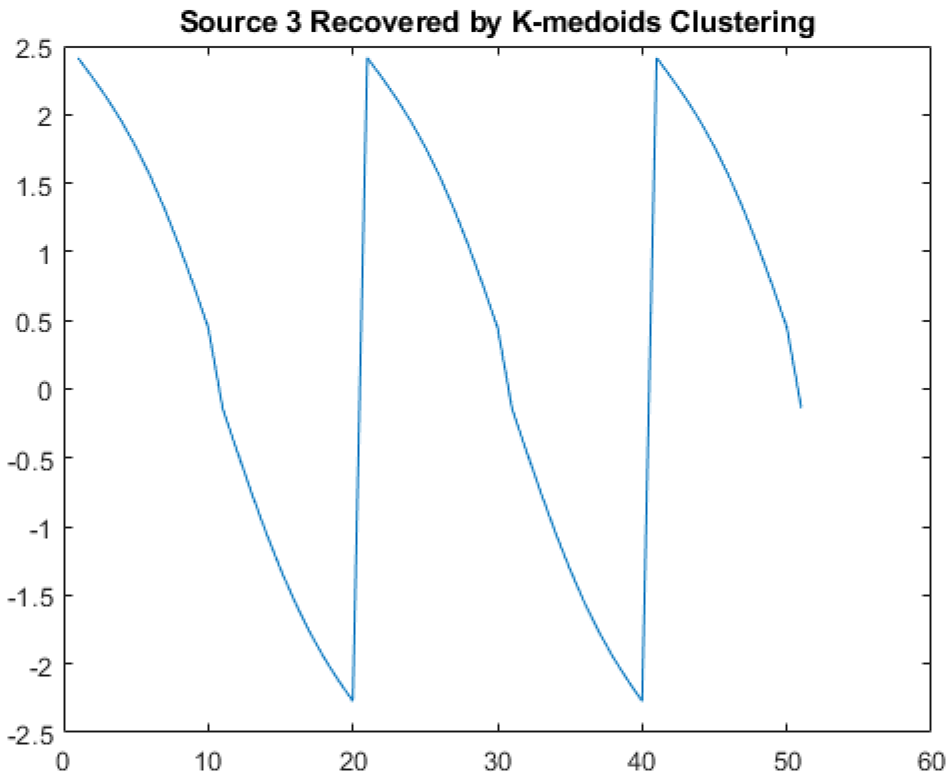
```
    1.5496
```


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