

A humble attempt
to solve the
“Cocktail-party” problem
Using Blind Source Separation

BY:
JOHAN BYLUND
0136719

Abstract

This report deals with one way of solving a simplified “Cocktail-party” problem, using Blind Source Separation. The specific problem is that of separating one source from a mixture of two. More precise, separating a piece of music from a mixture of music and a human voice, by means of the time independent one-unit approach of blind source separation.

Reviewing alternatives, problems and simplifications. Discussing the theory, the real life association, and the practical approach concerning recording and implementation in Matlab. The result is good and the algorithm is proven to work well in this particular case. However, weaknesses, or shortcomings, camouflaged by a “good” choice of sources can be seen.

Table of contents

Abstract	0
Introduction to the problem	3
Problem statement	3
<i>Specific problem</i>	3
Theory	4
<i>The weight-matrix</i>	4
Practical approach	6
<i>Recording</i>	6
<i>Matlab implementation</i>	7
Actual separation of sources	9
Weights	9
Results	9
Acknowledgements	12
Bibliography	12
<i>Internet:</i>	12
Appendix	13
<i>fftproject.m</i>	13
<i>sep.m</i>	15
<i>sepout.m</i>	15
<i>wchange.m</i>	15

Introduction to the problem

Have you ever been to a cocktail party? If so you most certainly know how hard it can be to extract an interesting conversation from the background sounds of a noisy crowd.

Having said that the aim of this report is to describe an attempt to solve the cocktail party problem. That is, extract one or a few sources from a mix of several.

The methods used goes under the name Blind Signal Separation, or Blind Source Separation.

Problem statement

The general problem addressed by Blind Source Separation is that of separating unobservable, or latent, source signals when mixed signals are observed. In other words to extract one, or a few, signals from a mixture of many.

Specific problem

Understanding that the level of the theory is a fair bit beyond the background knowledge I possess, my first thing was to simplify the general problem. Simplify it without missing the interesting parts, and most important not losing the excitement in that the problem could be implemented with real signals. For it is, as far as I am concerned, a problem that is easy to recognise in real life.

The general problem involves many sources, who are to be separated one by one in the so-called one-unit approach, or many at the same time in the multi-unit approach.

In both cases it is about separating different independent sources that appear linearly superposed, mixed.

By choosing to implement the one-unit approach to decrease complexity and difficulty, still, many sources make even that problem a fairly difficult one.

So to simplify the problem further I did chose to concentrate on extracting one source from a mix of two. Also making effort to find signals that are different and seem to have little in common, but still cover about he same spectra in the frequency domain. Hopefully without losing the excitement about the final result.

Theory

As mentioned before the theory behind the real Blind Source Separation is pretty difficult and involves massive likelihood estimations and even neural networks and learning algorithms. However the basic main flowchart of the simplified two-source problem could be modelled as follows.

$$\begin{bmatrix} \bar{S}_1 \\ \bar{S}_2 \end{bmatrix} \rightarrow \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \rightarrow \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \end{bmatrix} \rightarrow \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} \rightarrow \begin{bmatrix} \bar{y}_1 \\ \bar{y}_2 \end{bmatrix} \quad (1)$$

Where \bar{S}_1 and \bar{S}_2 are the two vectors containing the sampled points of the original sources – in either time or frequency domain.

$$\bar{S}_1 = [S_{11} \ S_{12} \ \Lambda \ S_{1(n-1)} \ S_{1n}], \bar{S}_2 = [S_{21} \ S_{22} \ \Lambda \ S_{2(n-1)} \ S_{2n}] \quad (2)$$

These two vectors are then mixed through a “mixing-matrix” A , which can be either time dependant or time independent. The time dependant model concerns the case where sources and sensors move in space as a function of time, which makes it more complex (Paraga, 1996). The problem approached in this report settles with space-fixed time independent sources and sensors. When dealing with real, not artificially mixed signals the mixing matrix, A , remains unknown. \bar{x}_1 and \bar{x}_2 are the mixed, or observed, signals. That is, \bar{x}_1 and \bar{x}_2 both contain a mix, specified by A , of information from both of the original sources, \bar{S}_1 and \bar{S}_2 . In the real case \bar{x}_1 and \bar{x}_2 are the two signals to begin with.

The next crucial step is then to estimate a weight-matrix, W , to apply on the mixed signals. After applying the weight matrix, W , to the observed signals, \bar{y}_1 and \bar{y}_2 are the resulting estimation of the original source signals, \bar{S}_1 and \bar{S}_2 .

The above process can be re-written in matrix form as follows:

$$\begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} * \begin{bmatrix} \bar{S}_1 \\ \bar{S}_2 \end{bmatrix} \text{ and } \begin{bmatrix} \bar{y}_1 \\ \bar{y}_2 \end{bmatrix} = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} * \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \end{bmatrix} \quad (3)$$

Further there is no information about whether the resulting extracted signals \bar{y}_1 and \bar{y}_2 corresponds to \bar{S}_1 or \bar{S}_2 . The only thing that is known is that one of the sources is intended to be separated from the mix of the two.

The weight-matrix

Although the mixed data have a linear structure, the main difficulty of the task lies in the fact that the mixture-matrix, the set of coefficients in the linear superposition, is unknown.

The main objective of the blind source separation is then to find the linear filter, or the weight-matrix, W , such that the output \bar{x}_1 and \bar{x}_2 gives a reconstruction of the original sources. In the ideal case, with a known mixing-matrix A , the linear filter, W should be the inverse of A .

The different methods for estimation of the components of W are numerous. Various gradient methods are the most commonly used. This report concentrates on the use of the *Natural Gradient method*¹. A method chosen because the parameter space isn't Euclidean, but has a Riemannian metric structure in most cases. In such a case the steepest direction of the function is given by the natural gradient, instead of by the ordinary gradient (Amari, 1997).

Assume that the sources, \bar{S}_1 and \bar{S}_2 , are independent at different samples, and that the expected value of them both equals zero. Then the joint probability density function, $r(S)$, written in the general product form, looks as follows.

$$r(S) = \prod_{i=1}^m r_i(S_i) \quad (4)$$

In the case discussed here, with two sources, $m = 2$.

As we are dealing with real signals we don't know the original sources, neither their probability distribution densities $r_i(S_i)$, nor the mixing-matrix A .

The weight-matrix, W_n , at each sample n an estimate of A^{-1} , that solves equation (3) above, is modified by the following learning equation.

$$W_{n+1} = W_n - Lf(\bar{x}_n, W_n) \quad (5)$$

Where L is the learning rate, and $f(\bar{x}_n, W_n)$ is a special matrix function that satisfies

$$E[f(\bar{x}, W)] = 0 \quad (6)$$

for any density functions $r(S)$ when $W = A^{-1}$.

Let $l(\bar{x}, W)$ be a loss function whose expectation is minimised at $W = A^{-1}$.

The function f is obtained by taking the gradient of that function, l , with respect to W (Amari, 1997).

$$f(\bar{x}, W) = \nabla l(\bar{x}, W) \quad (7)$$

¹ Also known as the Contravariant Gradient method.

The last part involves theory that lies beyond my background knowledge. Therefore proof or motivation from my side can't be provided.

Practical approach

The first thing that had to be done was to decide which signals to use. Perhaps more truthfully, to find signals that was easy to separate, without losing the real-life association of the problem.

At first I did the mistake of choosing two human voices, but realised fairly quickly that it was a bad choice. The reason for that is that I found out that human voices have a super-gaussian distribution function. Basically that means that the probability is high that two voices, even fairly different sounding ones, are relatively similar in each sample. This of course causes difficulties in the separation algorithm, and should be avoided.

Knowing that, the choice fell upon using my voice and one of Beethoven's compositions.

Recording

The next step was the recording of the two sources simultaneously with two microphones. This was done in VISLAB, at the University of Sydney, with the following configuration (*Figure 1*).

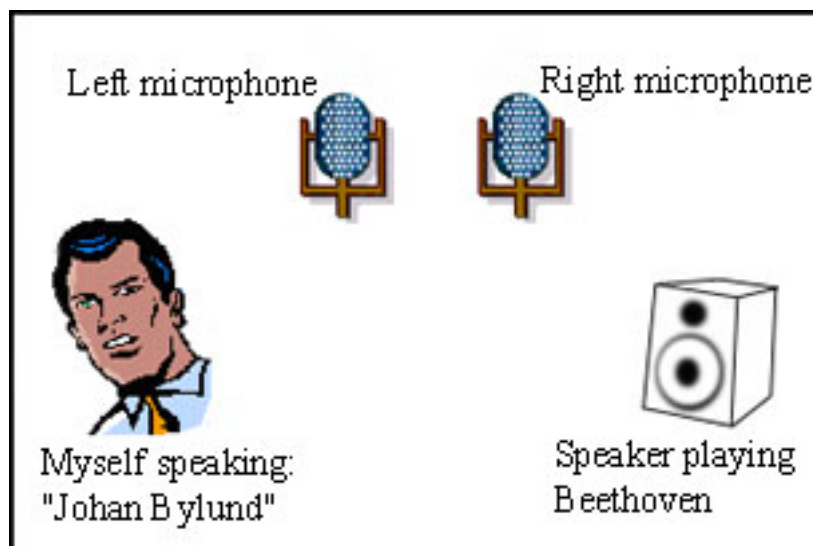


Figure 1. Recording configuration

That is, I being closer to the left microphone, and the speaker playing Beethoven being closer to the right one. The difference in distance results in that the amplitude of my voice becomes higher on the recording of the left microphone, than on the recording of the right microphone. Obviously likewise, but the other way around, concerning the speaker and the right microphone.

The two signals were recorded in WAVE-format, as leftmic.wav and rightmic.wav.
I choose to implement the algorithm in Matlab.

Matlab implementation

The first step was to read the two WAVE-files into a matrix. The matrix, x in equation (3) above, was defined to include two column-vectors, \bar{x}_1 and \bar{x}_2 , who contains the amplitude, in the time domain, of each sampled point. The plot of each vector looks as follows (*Figure 2*).

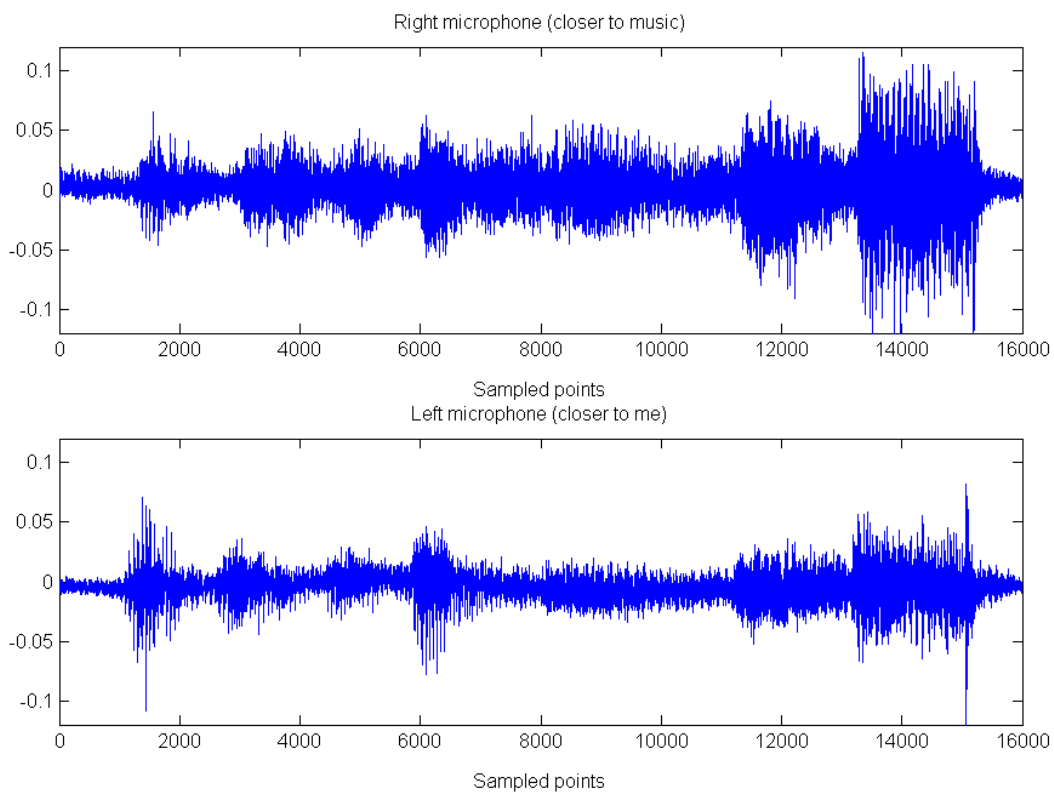


Figure 2. Plot of initial signals.

Then the Fourier transform of both signals was calculated. Showing considerable differences between the signals, but also that they cover about the same frequencies. The frequency spectra of the two signals are displayed below (*Figure 3*).

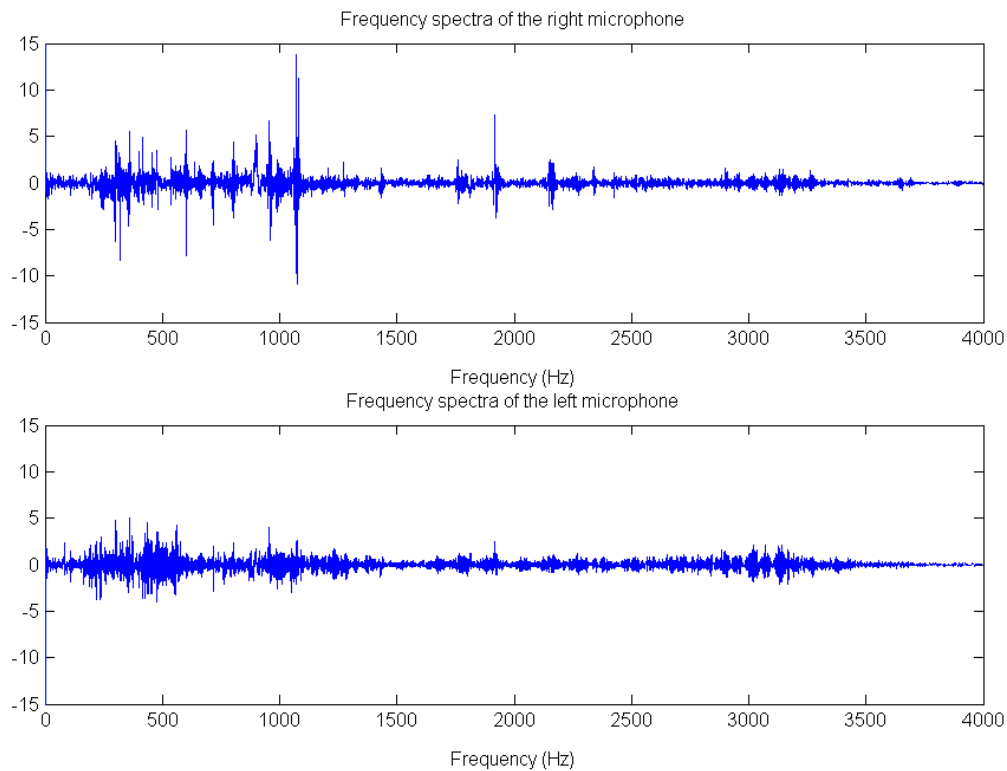


Figure 3. Plot of the frequency spectra of the two signals.

It is easy to identify the upper of the two signals plotted (*Figure 3.*) as the one corresponding to the microphone closest to the speaker playing music, due to its clear and distinct sinusoidal components.

The fact that the noticeable differences in the frequency spectra are bigger than in the time domain justified my choice of implementing the algorithm in the frequency domain.

To make the algorithm work properly the data at first had to be made uncorrelated. In order to make each signal-vector uncorrelated you want to make the sampled points statistically independent. That is achieved by randomly permutating the order of the sampled points. But doing the same operations on both the vectors, so that each point in the vector from the right microphone corresponds to the same point in the vector from the left microphone as before the permutation.

The data was then sphered, or normalised, by subtracting the mean from the amplitude in each point, and by then multiplying the statistically independent vectors with the inverse of the square root of the covariance matrix. The variance of the two signals should then be as equal to one another as possible, and the covariance between them close to zero. Subsequently the two signals

were divided into batch blocks, B , of a small number of points. After that the actual separation of the sources could begin.

Separation of sources

The separation algorithm goes through the two scrambled signals in batch blocks, B , and adjusts the weights, W , after each block. The amplitude of each point of each block is simply multiplied with the value of the corresponding weight. If the weight-factor is big the amplitude of that corresponding point in the output-vector, \bar{y}_i , will be high. This procedure is then looped several times, leading to that the dominant points will be amplified further.

Weights

The weights are calculated basically by taking the previous weight and adding it to the product of the learning rate and the function, f , which satisfies the conditions of equation (6) and (7) above. Unfortunately I cannot explain how to acquire the function f , more than that it is the gradient of a loss function, l , whose expectation is minimised when $W = A^{-1}$. However the function, f , can also be obtained by heuristic arguments, and the whole family of f satisfying equation (6) was tabulated by Amari and Cardoso in 1997 (Amari, 1997). By then using functions that I could find in literature and examples in an empirical fashion, changing constants, initial values etc finally resulted in a fairly well working algorithm.

Results

After looping the separation algorithm a number of times the output in the frequency domain came to look as follows (*Figure 4*).

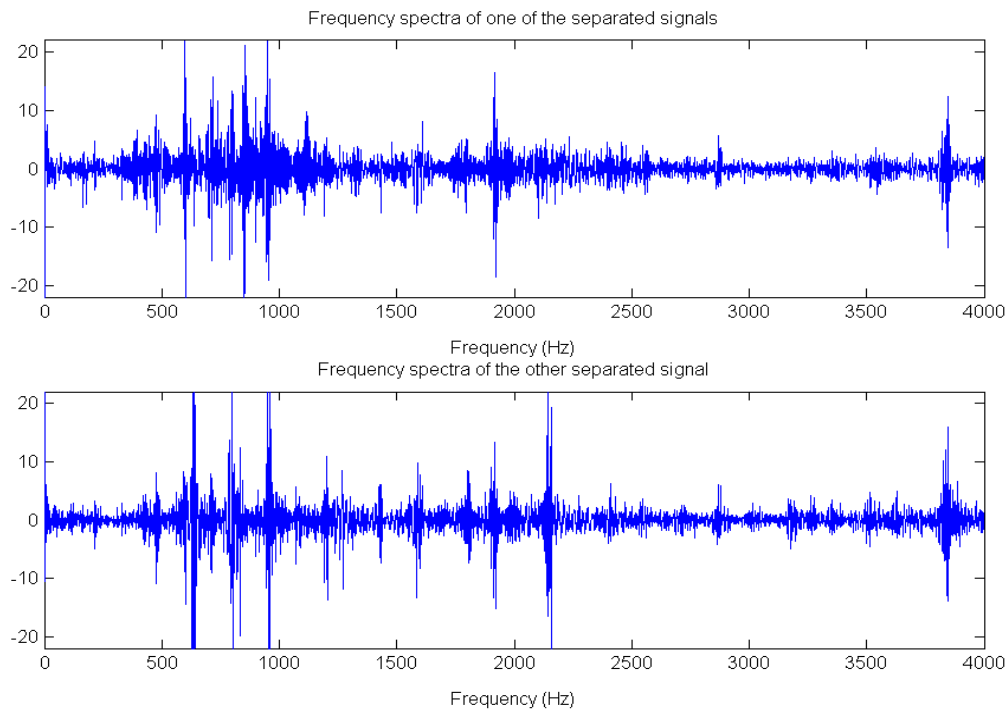


Figure 4. Plot of the frequency spectra of the separated signals.

Showing rather big similarities with the mixed signal of the right microphone, but with the peaks of the sinusoidal components of the music amplified effectively. Noticing also that the algorithm has increased the over-all amplitude distinctly.

By taking the inverse Fourier transform of the resulting vectors the result in the time domain was calculated. These are the two signals that we are able to listen to as sounds, and they are plotted below (*Figure 5.*).

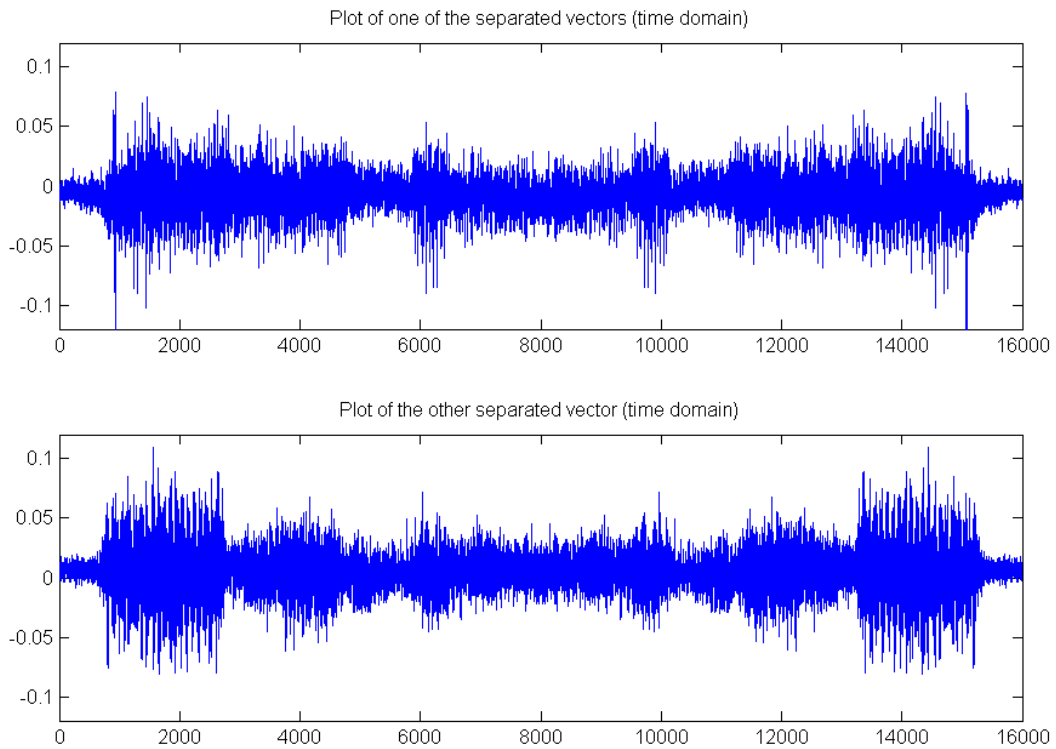


Figure 5. Showing the resulting vectors in the time domain.

As plots these signals are harder to compare with the initial data than the two frequency spectras. However, by listening to the upper of the previously plotted, it is evident that the music has been effectively extracted, and my voice is no longer hearable.

Acknowledgements

I would like to thank Kim Hui Yap, on Electrical Engineering, University of Sydney, for useful discussions and good ideas.

Bibliography

Amari S. Cichocki A. and Yang H.H. (1996) "A new learning algorithm for blind signal separation." *Advances in Neural Information Processing Systems 8*, MIT press.

Amari, S (1997) "Natural Gradient Works Efficiently in Learning."

Lee T-W. (1998) "Independent Component Analysis: Theory and Applications." Kluwer Academic Publishers.

Paraga N. Nadal J-P. (1996) "Blind Source Separation with Time Dependent Mixtures."

Råde L. Westergren B. (1998) *Mathematics Handbook for Science and Engineering*, 4th edn, Studentlitteratur, Lund, Sweden.

Rowe D.B. (1999) "Bayesian Blind Source Separation"

Internet:

"Tony Bells homepage" <http://www.cnl.salk.edu/~tony/>

"Paris Samaragdis' homepage" <http://sound.media.mit.edu/~paris/>

Appendix

fftproject.m

```
% fftproject.m
% By: Johan Bylund

clear, clf, format compact;

% Makes a matrix out of wav files in project directory.
% The arrays will all be in the same length as the shortest one.
% Files longer than the shortest one will be truncated.

disp('Reading .wav files from project directory');
mic_1=wavread('rightmic.wav'); %Reading file from right microphone.
mic_2=wavread('leftmic.wav'); %Reading file from left microphone.
size(mic_1)
size(mic_2)

% The below operation makes them 1xN:
mic_1=mic_1';
mic_2=mic_2';

if length(mic_1)>length(mic_2)
mic_1=mic_1(1:length(mic_2));
else
mic_2=mic_2(1:length(mic_1));
end

size(mic_1)
size(mic_2)

disp('Playing the recording of the right microphone (closer to music)');
soundsc(mic_1)
disp('Playing the recording of the left microphone (closer to me)');
soundsc(mic_2)

subplot(2,1,1)
plot(mic_1, axis([0 16000 -0.12 0.12]));
title('Right microphone (closer to music)')
xlabel('Sampled points');
subplot(2,1,2)
plot(mic_2, axis([0 16000 -0.12 0.12]));
title('Left microphone (closer to me)')
xlabel('Sampled points');

% I also choose to look at the frequency spectra of the signal:
Fs=8000; % Sampling frequency

% Matrix with the frequency spectra from the two microphones:
fftsounds=[real(fft(mic_1,Fs));real(fft(mic_2,Fs))];
f=[1:Fs/2];

figure(2)
subplot(2,1,1)
plot(f,fftsounds(1,f)), axis([0 4000 -15 15]);
title('Frequency spectra of the right microphone')
xlabel('Frequency (Hz)');
```

```
subplot(2,1,2)
plot(f,fftsounds(2,f)), axis([0 4000 -15 15]);
title('Frequency spectra of the left microphone')
xlabel('Frequency (Hz)');

% At first I tried the algorithm in the time domain, and it didn't
% manage to separate the two sources very well.
% After that I used the frequency spectra of the two microphone signals
% in the algorithm and it worked out much better.

sounds=[real(fft(mic_1));real(fft(mic_2))];

% N="number of microphones"
% P="number of points"
[N,P]=size(sounds) % P=16000, N=2, in this case.
permute=randperm(P); % Generate a permutation vector.
s=sounds(:,permute); % Time-scrambled inputs for stationarity.

x=s;
mixes=sounds;

% Spheres the data (normalisation).
mx=mean(mixes');
c=cov(mixes');
x=x-mx'*ones(1,P); % Subtract means from mixes.
wz=2*inv(sqrtm(c)); % Get decorrelating matrix.
x=wz*x; % Decorrelate mixes so cov(x')=4*eye(N);

w=pi^2*rand(N); % Initialise unmixing matrix.
M=size(w,2); % M=N usually
sweep=0; oldw=w; olddelta=ones(1,N*N);
Id=eye(M);

% L="learning rate, B="points per block"
% Both are used in sep.m, which goes through the mixed signals
% in batch blocks of size B, adjusting weights, w, at the end
% of each block.

%L=0.01; B=30; sep
%L=0.001; B=30; sep % Annealing will improve solution.
%L=0.0001; B=30; sep % ...and so on

%for multiple sweeps:
L=0.0001; B=30;
for I=1:100
    sep; % For details see sep.m
end;

uu=w*wz*mixes; % make unmixed sources

% Plot the two separated vectors in the frequency domain.
figure(3)
subplot(2,1,1)
plot(f,uu(1,f)), axis([0 4000 -22 22]);
title('Frequency spectra of one of the separated signals')
xlabel('Frequency (Hz)');
subplot(2,1,2)
plot(f,uu(2,f)), axis([0 4000 -22 22]);
title('Frequency spectra of the other separated signal')
xlabel('Frequency (Hz)');
```

```
    % Transform signals back to time domain.
uu(2,:)=real(iff(uu(2,:)));
uu(1,:)=real(iff(uu(1,:)));

disp('Playing the first of the separated vectors');
soundsc(uu(1,:))

    % Plot the vector that is played above.
figure(4);
subplot(2,1,1)
plot(uu(1,:), axis([0 16000 -0.12 0.12]));
title('Plot of one of the separated vectors (time domain)')

disp('Playing the second of the separated vectors');
soundsc(uu(2,:))

    % Plot the vector that is played above.
subplot(2,1,2)
plot(uu(2,:), axis([0 16000 -0.12 0.12]));
title('Plot of the other separated vector (time domain)')
```

sep.m

```
% SEP goes once through the scrambled mixed signals, x
% (which is of length P), in batch blocks of size B, adjusting weights,
% w, at the end of each block.

sweep=sweep+1; t=1;
noblocks=fix(P/B);
BI=B*Id;
for t=t:B:t-1+noblocks*B,
    u=w*x(:,t:t+B-1);
    w=w+L*(BI+(1-2*(1./(1+exp(-u))))*u')*w;
end;
sepout
```

sepout.m

```
% Called after each pass through the data.

[change,olddelta,angle]=wchange(oldw,w,olddelta);
oldw=w;
fprintf(' sweep=%d, change=%.4f angle=%.1f deg., [N%d,M%d,P%d,B%d,L%.5f]
\n',...
    sweep,change,180*angle/pi,N,M,P,B,L);
```

wchange.m

```
function [change,delta,angle]=wchange(w,oldw,olddelta)
[M,N]=size(w); delta=reshape(oldw-w,1,M*N);
change=delta*delta';
angle=acos((delta*olddelta')/sqrt((delta*delta')*(olddelta*olddelta')));
```