Separating CDMA sources using Blind Source Separation Methods

Sung-Phil Kim
University of Florida, Dept. of Electrical & Computer Engineering
phil@cnel.ufl.edu

Abstract

Independent Component Analysis (ICA) is a useful extension of standard Principal Component Analysis (PCA). The ICA model is utilized mainly in blind separation of unknown source signals from their linear mixtures. In some applications, the mixture coefficients are totally unknown, while some knowledge about temporal model exists. CDMA (Code Division Multiple Access) is an example of such an application; only the code of the mobile phone user is known, while the codes of the interfering users are unknown. In this case, linear methods such as matched filter fail to estimate the parameters. In this work, we introduce two learning source separation methods for estimating the CDMA symbols. The first one is independent component analysis (ICA), which is the extension of the principle component analysis (PCA) and the second is maximizing entropy, which uses Renyi’s entropy theorem. We will simulate the performance of ICA and maximum entropy and compare the results.
1. INTRODUCTION

Code Division Multiple Access (CDMA) is a data transmission technique for the third generation of mobile phones, based on spread spectrum methods. The CDMA system enables users to transmit over the same frequency user is done by his unique code.

The signal processing in the downlink (e.g. mobile phone) differs from that of the uplink. First only the own code of the mobile phone is known. Second, the mobile phone doesn’t have such signal processing capability as the base station. Third, the mathematical model of channels differs slightly since users share the same channel in the downlink communication.

The independent component analysis (ICA), which is an extension of standard Principal Component Analysis (PCA) has been proposed as a method in blind source separation for CDMA. ICA can represent a set of random variables as a linear transformation of statistically independent component variables. The unknown linear mixing, as well as the generating independent components can be found by using ICA algorithms that utilize either higher-order moments or time structure of the observed variables. Another approach for the separation of the blind sources is the maximum entropy method due to Bell and Sejnowski (1995). We know that maximizing the entropy of the output is equivalent to maximizing the likelihood function of probability density function (pdf) of the source vector. Maximum entropy method can provide the separation of the signal sources of CDMA system.
The model considered in this paper consists of a linear transformation of both the independent variables, i.e., the transmitted signals and on time unit delayed version of them. Two methods for separating sources are used. The simulation results will be compared and analyzed.

2. Independent Component Analysis (ICA)

We approach the problem by using blind source separation, which is closely related to Independent Component Analysis (ICA), of a mixture of convolved sources. Independent component analysis (ICA) is a recently developed technique whose goal is to represent a set of random variables as a linear transformation of statistically independent component variables. The unknown linear mixing, as well as the generating independent components can be found by using ICA algorithms that utilize either higher-order moments or time structure of the observed variables. Temporal structure of the data is exploited. The essential assumption of the linear data is that their alphabet is known but the parametric form itself is unknown. No strict assumptions about correlation or independence properties are made.

In this application only the source signals which correspond to the coefficients of the ICA expansion are of interest. The basic ICA network usually consists of whitening, separation, and basis vector estimation layers.

In the following, we present the basic data model used in defining standard ICA.
denoted by

\[ r_m = [ r_{m(1)}, ..., r_{m(C)} ]^T \]

The \( C \) dimensional \( m \)th observed data vector. \( T \) denotes transpose. In the linear data model, it can be represented in the form

\[ r_m = G b_m + n_m \]

where \( C \times M \) mixing matrix

\[ G = [g_1, ..., g_M] \]

contains the ICA basis vectors, and \( M \)-vector \( b_m \) contains the source signals

\[ B_m = [b_{m(1)}, ..., b_{m(M)}]^T \]

In the standard ICA, the components of \( b_m \) are assumed to be independent and non-Gaussian. The vector \( n_m \) denotes corrupting additive independent, identically distributed(i.i.d.) noise. The noise term \( n_m \) is often omitted from the equation for analytical reasons, and because it is usually impossible to distinguish noise from the source signals. However, we don not drop noise out until otherwise stated.
3. SIGNAL MODEL

The signal model considered in this project is a multipath downlink model with slowly fading channel. The data has the form:

\[ r(t) = \sum_{m=1}^{N} \sum_{k=1}^{K} \sum_{l=1}^{L} b_{km} a_{kl} s_k(t - mT - d_{kl}) + n(t) \]  

(1)

where \( a_l \) is the fading factor of the \( l \)th transmission path, \( b_{km} \) is \( k \)th user \( m \)th symbol, \( s_k(\cdot) \) is \( k \)th user’s chip sequence, \( s_k(t) \in \{-1, +1\} \), \( t < [0, T) \) \( s_k(t) = 0 \), \( t \in [0, T), \) \( d_l \) is the delay corresponding to the \( l \)th path, which is assumed to be constant during the observation time, and \( n(t) \) is the noise. We also assume that there are \( L \) transmission paths, and that the fading coefficients corresponding to each path are constant for the given observation interval. The length of the chip sequence is \( C \) and \( N \) is the number of bits in the observation interval.

We collect \( C \)-length column vectors from subsequent discretized equispaced data samples \( r[n] \):

\[ r_m = [r[mC] \ r[mC + 1] \ ... \ r[m + 1]C - 1]^T \]  

(2)

Equation (2) can further be written with respect to the current symbol vector and the immediately preceding one:
\[ r(t) = \sum_{k=1}^{K} \sum_{l=1}^{L} a_i g_{kl} + b_{km} \sum_{l=1}^{L} a_i g_{kl} \] + n_m \tag{3} \]

where \( n_m \) denotes the noise vector, and the ‘early’ and ‘late’ parts of the code vectors are

\[ g_{kl} = [ s_k[C - d_l + 1] \ldots s_k[C] 0 \ldots 0]^T \tag{4} \]
\[ g_{kl} = [ 0 \ldots 0 s_k[1] \ldots s_k[C - d_l] ]^T \tag{5} \]

Here \( d_l \) is the discretized delay, \( d_l < \{0, \ldots, (C-1)/2\} \).

The matrix \( R = [r_1 \ldots r_N] \) can be represented in the compact form

\[ r_m = G_o b_m + G_1 b_{m-1} + n_m \tag{6} \]

where \( G_o, G_1 \) are \( C \times K \) mixing matrices corresponding to the original and the one time unit delayed symbols. The column vectors of \( G_o \) and \( G_1 \) are given by the ‘early’, respectively ‘late’ parts of the coding vectors:

\[ G_0 = [ \sum_{\ell=1}^{L} a_i g_{i\ell}, ..., \sum_{\ell=1}^{L} a_1 g_{i\ell} ] \tag{7} \]
4. Algorithms

We notice from equation (6) that our signal model can be regarded as a linear mixture of delayed and convolved source, in the particular case when the maximum delay is one time unit. If we consider that none of the mixing matrices (i.e. users codes) or symbol sequences is known, then we are in the framework of blind source separation (BSS). One method for solving this convolved mixtures BSS problem is by considering a feedback architecture. Since we suppose the users to be independent, then we may employ for the task of source separation the maximization of the entropy of a nonlinear function of the output, and optimize the weights of the network using the natural gradient algorithm.

A common preprocessing step in ICA and BSS is whitening of the input data \( r_m \), which removes as much as possible of the second order statistic effects and filters some additive noise. In the whitening step we also reduce the dimension to \( K \) (the number of users) since this is the actual number of independent components.

The new data is then of the form

\[
V = \Lambda^{1/2} U^T R
\]  

(9)
where $\Lambda_s$ and $U_s$ corresponds to the K eigenvalues and eigenvectors of the correlation matrix estimate $R R^T / (N - 1)$, respectively.

Then we may write the whitened version of equation (6):

$$v_m = A_0 b_m + A_1 b_{m-1}$$ (10)

where $v_m$ is the whitened input vector, and $A_{0,1}$ are the whitened square matrices of dimension K. Now we received whitened mixture $v_m$ and the previously estimated vector of symbols $b_{m-1}$

$$b_m = A_0^{-1} (v_m - A_1 b_{m-1})$$ (11)

where $v_m$ is the whitened input vector, and $A_0, A_1$ are the whitened square mixing matrices of dimension K.

Based on this feedback architecture we propose the following algorithm for blind symbol detection in a CDMA system:

1. Initialize randomly $A_{0,1}$
2. Use the following update rules for the mixing matrices

$$\Delta A_0 = -A_0 (I + y_m b_m^T)$$ (12)

$$\Delta A_1 = (I + A_1) y_m b_{m-1}^T$$ (13)

where $y_m = f(b_m)$ is applied elementwise and $f$ is an appropriate nonlinearity function.
3. Update the system using the following equations

\[ A_{0,1} = A_{0,1} + \mu \Delta A_{0,1} \]

4. Compute \( b_m \) as in equation (11)

5. If \( A_{0,1} \) have not converged, go to step 2,

6. Apply sign nonlinearity on the final estimated value of \( b_m \)

7. Identify the desired user's symbol sequence which best fits the training sequence.

The identity matrix of size \( K \) is denoted here by \( I \). If some information about the delays of transmission of the desired user is available, it may be used in the initialization of step 1. Since the system has feedback the choosing of the learning rate \( \mu \) in step 4 is essential for the convergence. We used a constant rate, but some iteration dependent strategy may improve the convergence time. Convergence in step 5 is verified by the mean-square-error matrix norm. Since the transmission system is binary differential, step 6 provides the most probable value for the estimated symbol. The detector estimates all users symbols upto a permutation, that is why the pilot sequence in step 7 is needed to identify the desired user.
5. Simulations

Here, we used the Walsh code of length 8. The number of users was $K = 4$ and the number of channel path was $L = 3$. We fixed the channel path gain as 1, 0.5 and 0.1 for every user assuming the downlink. The signal-to-noise ratio was 100dB. Only the real part of the data was used. The 20 bits of source signals were generated. We could generate the binary signals randomly, but for the comparison reason, we set the specific signals. After performing the separation, the permutation process was needed since ICA algorithm performs up to the permutation of the mixing matrix. The learning rate was 0.05. The algorithm converged about 20 iterations, but simulation was done for 50 iterations to avoid local minima.

The figure 1-1, 1-2 and 1-3 show the source, mixed and the separated signals for the ICA algorithm, respectively.
Figure 1-1. Source signals for ICA

Figure 1-2. Mixed signals after whitening for ICA
It can be shown that ICA algorithm produced about 29% bit error rate for detection the source signal

6. Conclusions

In this paper, we reviewed the proposed algorithm of the ICA with feedback architecture for symbol separation in a CDMA system. The sources were assumed to be mixed and convolved.

We can see that the ICA algorithm did not perform very well to separate the source signals from the mixed data compared to the other algorithm. It can be
supposed that the reason of the poor result was that we did not use the orthogonal sequence. Also, the initial values of the separation matrix can be another reason, although we tried to vary those values. Increasing the SNR did not help to improve the performance. Reducing the number of paths improved the results much or less. We used the different nonlinearity functions for two methods, tanh function for ICA and inverse of exponential function for maximum entropy. We do not conclude this difference could cause the different results but we did not verify it.

For the further study, using different whitening process may result in a good separation. Also, using different sequence code such as Gold sequence can provide the better performance. Training the algorithm using the pilot sequence can improve the separation, too.

References


Asynchronous Direct-Sequence Code Division Multiple Access Systems”, *IEEE Transactions on Communications*, vol.44, pp84-93 1996


Appendix

Common Matlab codes used in both algorithms

1. Walsh code

```matlab
function Wn=walsh(N,option);
%WALSH Returns walsh codes of length N

M = ceil(log(N)/log(2));  %find the power of 2 to match N, e.g. M=5
for N=32
    if (nargin ~= 2),
        option = '++';  %Set default to ones and zeros
    end
    if (option=='+-'),
        if 2^M == 1,
            Wn = [1];
        elseif 2^M == 2,
            Wn = [1 1; 1 -1];
        else
            Wn = [1 1 1 1; 1 -1 1 -1; 1 1 -1 -1; 1 -1 -1 1];
            for k = 1:M-2,
                Wn = [Wn Wn; Wn (-Wn)];
            end
        end
    else
        if 2^M == 1,
            Wn = [1];
        elseif 2^M == 2,
            Wn = [1 1; 1 0];
        else
            Wn = [1 1 1 1; 1 0 1 0; 1 1 0 0; 1 0 0 1];
            for k = 1:M-2,
                Wn = [Wn Wn; Wn (-Wn)];
            end
        end
    end
end
```
2. Whitening code

```matlab
function whitened_data = whiten1(data, NumUser);
%WHITEN - Whiten matrix.
%
% Whitens the data (matrix). Returns the whitened matrix.
%
% data Matrix to be whitened
%
% Determine the length of column of the matrix
[null, N]=size(data);

% Calculate the principal eigenvalues and eigenvectors
[E,D]=pcamat(data, 1, NumUser);

% Calculate the whitening matrix
whiteningMatrix = inv(sqrt(D)) * E';
if ~isreal(whiteningMatrix)
    whiteningMatrix=real(whiteningMatrix);
end

% Whiten the data
whitened_data = whiteningMatrix * data;
```

3. PCA code to calculate correlation

```matlab
function [E, D] = pcamat(vectors, firstEig, lastEig)
% Calculates the PCA matrices for given data (row) vectors. Returns
% the eigenvector (E) and diagonal eigenvalue (D) matrices containing
% the
% selected subspaces. Dimensionality reduction is controlled with
% the parameters 'firstEig' and 'lastEig'.
%
% Default values:
if nargin < 3, lastEig = size(vectors, 1); end
if nargin < 2, firstEig = 1; end

oldDimension = size(vectors, 1);

% Calculate PCA

% Calculate the covariance matrix.
covarianceMatrix = cov(vectors', 1);
```
maxLastEig = rank(covarianceMatrix, 1e-9);

% Calculate the eigenvalues and eigenvectors of covariance matrix.
[E, D] = eig(covarianceMatrix);

% Sort the eigenvalues - decending.
eigenvalues = flipud(sort(diag(D)));

% See if the user has reduced the dimension enough
if lastEig > maxLastEig
    lastEig = maxLastEig;
end

% Drop the smaller eigenvalues
if lastEig < oldDimension
    lowerLimitValue = (eigenvalues(lastEig) + eigenvalues(lastEig + 1)) / 2;
else
    lowerLimitValue = eigenvalues(oldDimension) - 1;
end
lowerColumns = diag(D) > lowerLimitValue;

% Drop the larger eigenvalues
if firstEig > 1
    higherLimitValue = (eigenvalues(firstEig - 1) +
                        eigenvalues(firstEig)) / 2;
else
    higherLimitValue = eigenvalues(1) + 1;
end
higherColumns = diag(D) < higherLimitValue;

% Combine the results from above
selectedColumns = lowerColumns & higherColumns;

% Select the columns which correspond to the desired range
% of eigenvalues.
E = selcol(E, selectedColumns);
D = selcol(selcol(D, selectedColumns)', selectedColumns);

defunction newMatrix = selcol(oldMatrix, maskVector);

% Selects the columns of the matrix that marked by one in the given
% vector.
% The maskVector is a column vector.

if size(maskVector, 1) ~= size(oldMatrix, 2),
    error ('The mask vector and matrix are of uncompatible size.');
end
numTaken = 0;

for i = 1 : size (maskVector, 1),
    if maskVector(i, 1) == 1,
takingMask(1, numTaken + 1) = i;
numTaken = numTaken + 1;
end
end

newMatrix = oldMatrix(:, takingMask);

**Matlab Codes for ICA algorithm**

1. Signal generating code

```matlab
function [data, G0, G1] = gen_data(bin_signal, SNR, ChipLength, PathGain)
    \%This models a CDMA data model with fading channel and noise. This returns
    \%the receiving signal.
    \%
    \% input
    \% bin_signal : M-by-N matrix, where M is Number of users and N is Number of bits
    \%   per user.
    \% SNR : Signal-to-Noise Ratio
    \% ChipLength : Chip sequence length
    \% PathGain : 1-by-(number of path) vector that contains the path gain of each
    \%
    \% output
    \% data : receiving signal with mixed transmitted singal plus noise
    \% NumUser : Number of User
    \% NumPath : Number of Paths
    [NumUser, NumBit] = size(bin_signal);
    \%
    \% L=NumPath; \%number of channel
    \% [none, NumPath] = size(PathGain);
    \% Generate Walsh Code
    s = zeros(NumUser, ChipLength); \%sequence of each (kth) user
    seq = randperm(ChipLength);
    for n = 1:NumUser,
        w = walsh(ChipLength, '+-');
        s(n,:) = w(seq(n),:);
    end \%set sequence of each user
    \% a=zeros(1,NumPath); \%path gain for downlink
    d = zeros(1, NumPath); \%delay
    for n = 1:NumPath
        d(n) = round(rand*(ChipLength-1)/2);
    end
```
G0=zeros(ChipLength,NumUser); %mixing matrix
G1=zeros(ChipLength,NumUser); %mixing matrix
ge=zeros(ChipLength,NumPath*NumUser); %early delayed vectors
gl=zeros(ChipLength,NumPath*NumUser); %late delayed vectors

%determine basis vectors
for u = 1:NumUser,
    for j = 1:NumPath,
        for i = 1:d(j),
            ge(i,j+(u-1)*NumPath)=s(u,ChipLength-d(j)+i);
        end
    end
end
for u = 1:NumUser,
    for j = 1:NumPath,
        for i = 0:ChipLength-d(j)-1,
            gl(ChipLength-i,j+(u-1)*NumPath)=s(u,ChipLength-d(j)-i);
        end
    end
end

%finding mixing matrix using basis vectors and fading terms
for i = 1:NumUser,
    for j = 1:NumPath,
        G0(:,i)=G0(:,i)+PathGain(j)*ge(:,j+(i-1)*NumPath);
        G1(:,i)=G1(:,i)+PathGain(j)*gl(:,j+(i-1)*NumPath);
    end
end

%mixed signals
R=zeros(ChipLength,NumBit);
for i = 1:NumBit,
    if i == 1,
        R(:,i)=G0*bin_signal(:,i);
    else
        R(:,i)=G0*bin_signal(:,i) + G1*bin_signal(:,i-1);
    end
end

%noise term
SNR=10;
NoiseFac=10^(-SNR/20);
noise=zeros(ChipLength,NumBit);
for i = 1:NumBit,
    SigPwr=std(R(:,i));
    noise(:,i)=randn(ChipLength,1)*SigPwr*NoiseFac;
end

%received signal
X=zeros(ChipLength,NumBit);
for i = 1:NumBit,
    X(:,i)=R(:,i)+noise(:,i);
end
data=X;
2. ICA code

function 
[data, BitErrorRate, result] = conv_ica(NumUser, NumBit, PathGain, SNR, ChipLength, myu, NumIteration)

% generating source signal
src(1,:) = ones(1, NumBit);
for i = 1:NumBit,
    for k = 2:NumUser,
        src(k, i) = sign(rand - 0.5);
    end
end

[R, G0, G1] = gen_data1(src, SNR, ChipLength, PathGain);
V = whiten1(R, NumUser);

% myu = constant step size
% initialization of mixing matrices and symbol
% A0 : early mixing matrix
% A1 : late mixing matrix
% b0 : estimation of delayed symbol
% b1 : estimation of current symbol

A0 = rand(NumUser) - 0.5*ones(NumUser);
A1 = rand(NumUser) - 0.5*ones(NumUser);
b = rand(NumUser, NumBit) - 0.5*ones(NumUser, NumBit);

% Updating the mixing matrices
for k = 1:10,
    for i = 1:NumBit,
        y = tanh(b(:, i));
        if i == 1,
            dA0 = -A0*(eye(NumUser) + y*b(:, i)');
            A0 = A0 + myu*dA0;
            if rank(A0) == NumUser
                A0 = A0;
            else
                b(:, i) = inv(A0)*V(:, i);
            end
        else
            dA0 = -A0*(eye(NumUser) + y*b(:, i)');
            dA1 = -(eye(NumUser) + A1)*y*b(:, i-1)';
            A0 = A0 + myu*dA0;
            A1 = A1 + myu*dA1;
            if rank(A0) == NumUser
                A0 = A0;
            else
                b(:, 3) = inv(A0)*(V(:, i) - A1*b(:, i-1));
            end
        end
    end
end
% finding the final estimation of symbols
result = sign(b);

% finding the best fitted sequence
match = zeros(1,NumUser);
BitError=NumBit;
for j=1:NumUser,
    min=NumBit;
    fit=0;
    for k=1:NumUser,
        dif=0;
        for i = 1:NumBit,
            if ~(result(j,i)+src(k,i))
                dif=dif+1;
            else
                dif=dif;
            end
        end
        if (dif < min )
            min=dif;
            fit=k;
        end
    end
    if (min < BitError)
        BitError = min;
    end
    match(1,j)=fit;
end
for i = 1:NumUser,
    new_src(i,:)=src(match(1,i),:);
end

% Plotting the graph of source and estimation
for j=1:NumUser,
    n=1:1:NumBit;
    subplot(NumUser,1,j);
    plot(n,new_src(j,:), '.' )
end
title('Binary Symbols of Source Signal for each User')
figure(2)
for j=1:NumUser,
    n=1:1:NumBit;
    subplot(NumUser,1,j);
    plot(n,result(j,:), '.' )
end
title('Binary Symbols of Estimated Signal for each User')

% Returning the result
data = result;
BitErrorRate = (BitError) / NumBit * 100;