**Implementation and Optimization of Fast ICA Algorithm**

Ashok Rangaswamy,  
Associate Professor

Department of Electrical and Electronics Engineering.

Sri Shakthi Institute of Engineering and Technology  
Coimbatore, Tamilnadu, India.

Independent component analysis (ICA) is a recently proposed method as a solution to the blind source separation problem. The objective is to recover the unobserved source signals from the observed mixtures without the knowledge of the mixing coefficients. It has the potential for a wide range of applications in industrial, medical, security, and military fields because it reduces the complex problem of dealing with high-dimensional statistical descriptions to products of one-dimensional density functions[1,2]. For some applications, off-line ICA analysis on a workstation could be adequate, but for a vast majority it is desirable to have the optimized implementation for real-time analysis. I would like to implement and optimize the FAST ICA algorithm with pipeline parallelism using CilkPlus, OpenMP or Intel Threading Building Blocks.

## Motivation for ICA

Imagine that in a room, two people are speaking simultaneously and that there are two microphones which produce time signals denoted by *x1(t)* and *x2(t)*. Each of these received signals is a weighted sum of the speech signals emitted by the two speakers denoted by s1(t) and s2(t). Then we can express the received signals in terms of the original signals as

(1)

where *a11, a12, a21,* and *a22* are certain parameters that depend on the microphone characteristics and their distances from the speakers. Clearly, it would be very useful to recover the original speech signals from the received signals.

More generally, if there are n different signals and n received mixed signals, then the relationship can be expressed as

. . . . . .

(2)

or in matrix-vector notation .

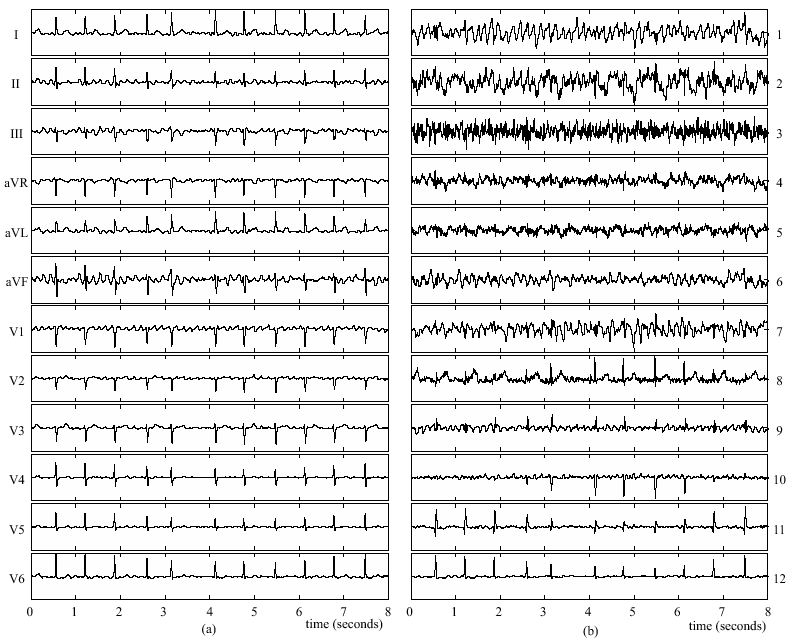
Here, for example s1 and s2 could be speech signals, s3 could be the sound produced by a motor vehicle, etc. In a biomedical environment s1(t), … could represent a set of EEG signals, ECG signals, etc. The recently developed technique called ICA, can be used to estimate A or its inverse W = A-1 based on the information of their statistical independence, which then allows blind separation of the original signals from their mixtures.

This algorithm has a wide range of applications in industrial and medical fields. For some specific application, ICA analysis on a workstation is adequate, but for a vast majority it is desirable to have a high performance Independent component analysis (ICA) in real-time.

## ICA APPLICATIONS

### Example 1: Hidden Information extraction from ECG signals

(a) 12–lead ECG from a patient (b) ICA estimated sources from lower to higher kurtosis.



### Example 2: Image Processing

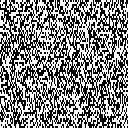
Mixed image

x1 x2 x3

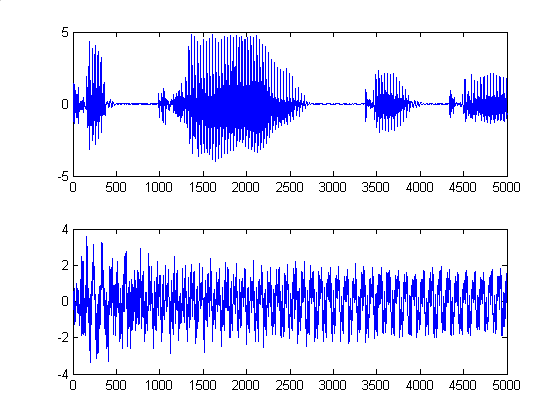
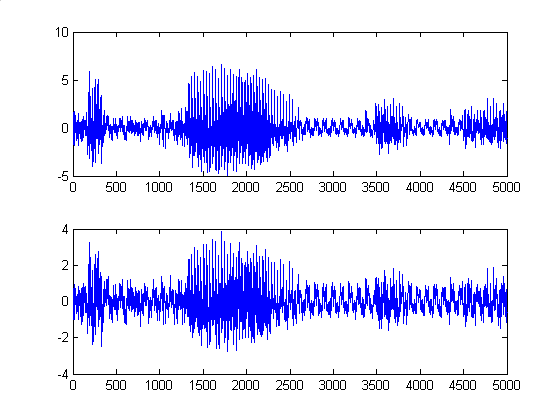
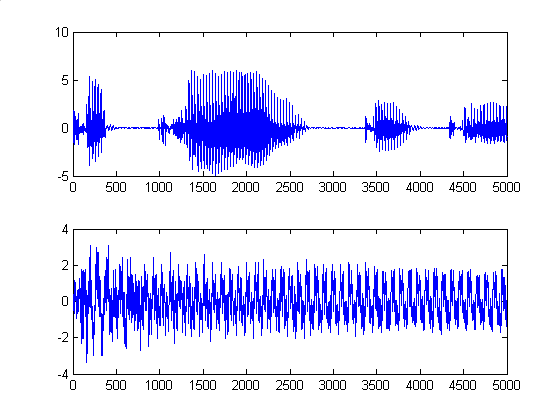
  

Separated image

s1 s2 s3

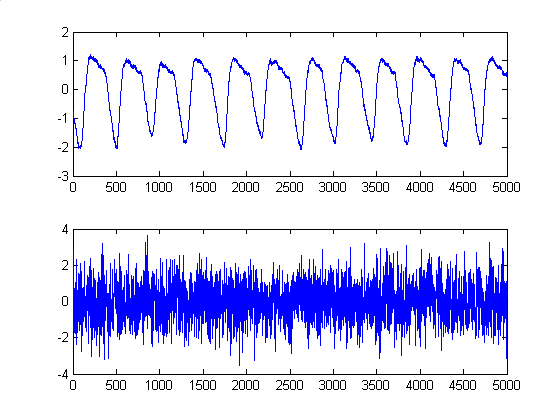
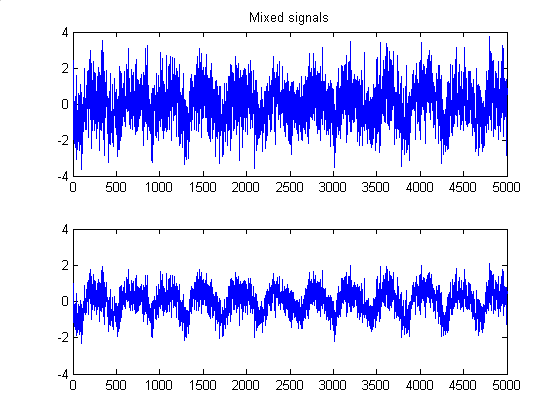
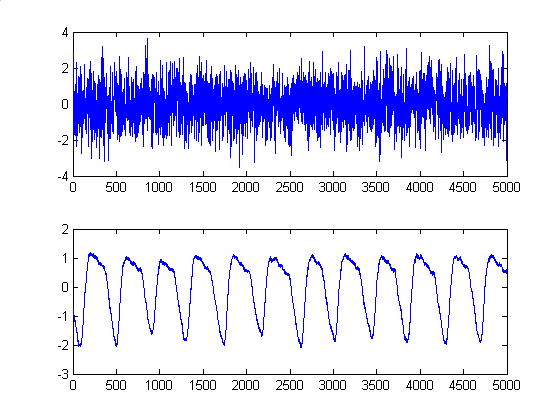
  

### *Example 4: Speech processing*



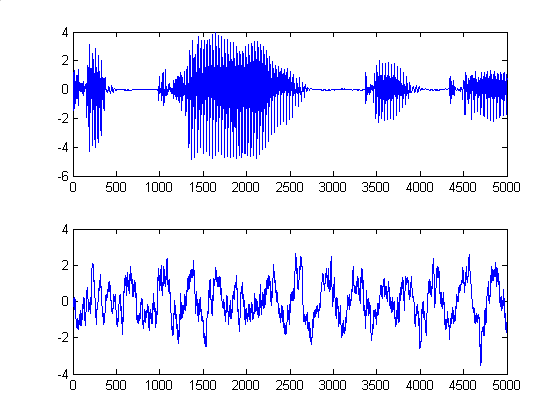
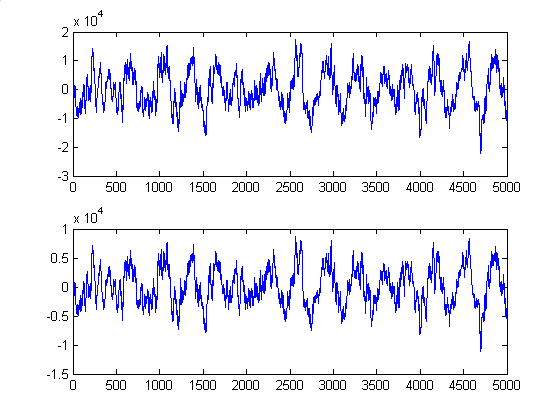
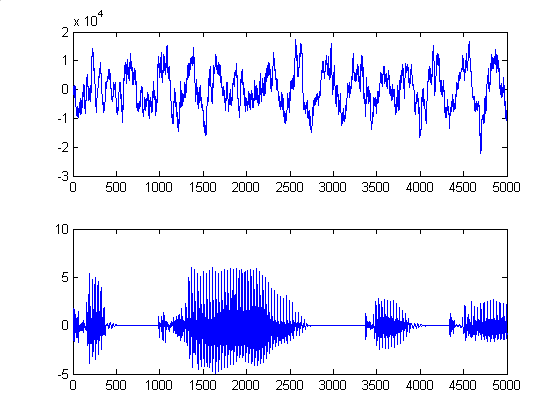
a. Source speech signals b. Mixed Signals c. Estimated signals by matlab code

### Example 5: Bio signals separation



**a**. Source bio signals (5000 samples) **b**. Mixed Signals **c**. Estimated signals by matlab code

### Example 6: Military Application (encryption)



a. Source: military truck + speech b. Mixed Signals c. Estimated signals

# **COMPUTATIONS IN ICA ALGORITHM**

Independent components

Eigen Decomposition

and

Whitening

Decorrelation

and

Fast ICA for

one unit

Checking for convergence

Formation of

De mixing

matrix

Observed data

Separate

the data

ICA Computation Sequence

## Preprocessing for ICA

*Centering*: Subtract mean vector *m* from *x* to make *x* a zero mean variable.

*Whitening*: Whitening reduces number of parameters to be estimated. Whitened data has its components uncorrelated and their variances equal unity. In other words, the covariance matrix of  is identity matrix. Whitening can be done using eigen value decomposition (EVD) of the covariance matrix of mixed signals *, C*. Let *V* is the matrix of eigenvectors of *C* and *Λ* is the Diagonal matrix of eigen values of *C* after EVD. Whitening is done by



Whitening transforms mixing matrix *A* into  where .

*Application dependent further preprocessing*: If data consists of time-signals, band pass filtering may be useful.

## Iterative Computation

The fast ICA finds a direction, i.e. a unit vector w such that the projection *wTx* maximizes non-gaussianity. Non gaussianity is measured by negentropy J(*wTx*) where .

*z* is Gaussian variable of zero mean and unit variance (i.e. standardized). The variance of *wTx* has to be unity for the measure to be valid. For whitened data, this is equivalent to constraining the norm of w to be unity. To prevent different vectors from converging to the same maxima, we must decorrelate them after each iteration. For this, when we have estimated *p* vectors, *w*1, *w*2, …, *wp*, we run the algorithm for *wp+*1, and after every iteration step subtract from *wp*+1 the projection matrix *B*, whose columns are *w*1 ,*w*2, …, *wp*.

The algorithm has the following steps:

*Step 1***:** *Initialization* : Choose initial random weight vector  with norm 1. Let *B* be the null matrix of the size of number of independent components.

*Step 2* **:** *Iteration* : Let the non-linear function be . Then



The update of the nth row of *W* is given by .



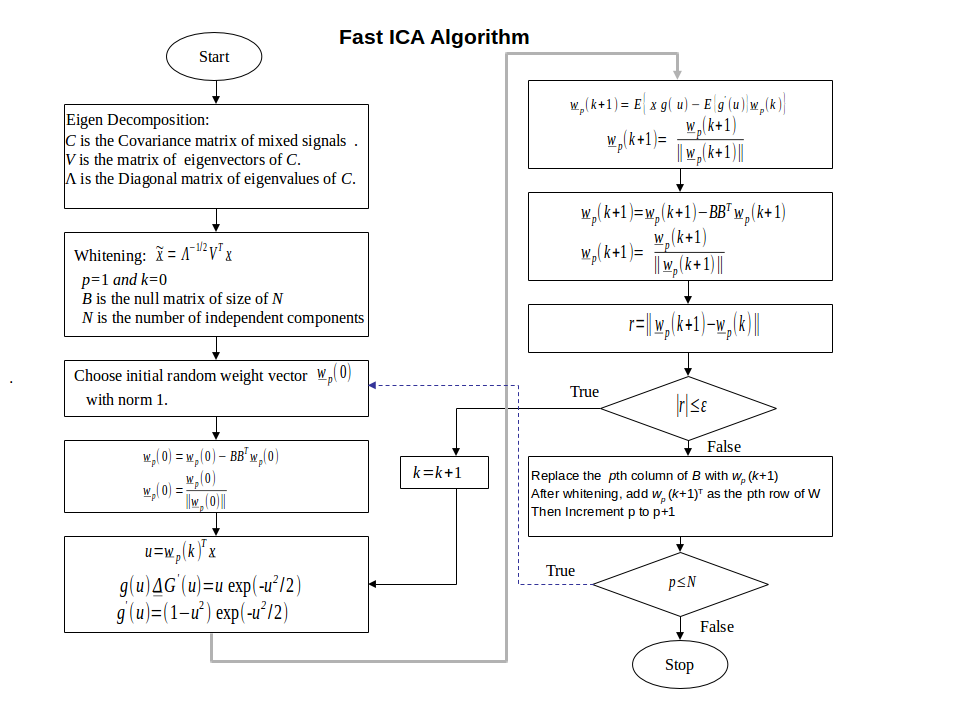


*Step 3* **:** *Decorrelation* : To prevent the different vectors from converging to the same maxima, it needs to be decorrelated.



*Step 4* **:** If  and  have converged, then goto step 5, else increment *k* to *k* + 1 and goto step 2.

*Step 5* : Replace the nth column of *B* with . After whitening, add  as the nth row of *W*. Increment *n* to *n* + 1 and set *k* = 0. If  number of independent components, then goto step 2 else stop.



# **PRELIMINARY IMPLEMENTATION IN MATLAB**

% **MATLAB CODE FOR FAST ICA ALGORITHM**

%epsilon -> convergence factor

epsilon=0.01;

Niter= 100;

%load input signals data X

load X;

[M N] =size(X);

B = zeros(M);

C = cov (X');

[V, D] = eig(C);

% V matrix of eigenvectors of C

% D Diagonal matrix of eigenvalues of C

Q = inv(sqrt(D))\*V'; % Whitening Matrix

Qinv = V\*sqrt (D);

At=Q\*A;

Xt=Q\*X;

%------FAST ICA ITERATION UNIT---------------------

for m=1:M;

%initial Vector

w = randn(M,1); w=w-B\*B'\*w; w=w/norm(w);

w\_old = zeros(size(w));

for iter=1:Niter,

Xg=[]; g\_der\_vec=[]; Gvec=[];

w=w-B\*B'\*w; w=w/norm(w);

% Test for convergence

if norm(w - w\_old)<epsilon | norm(w + w\_old)<epsilon,

B(:,m)=w; Wt(m,:)=w'\*Q; Wtinv(:,m)=Qinv\*w;

break

end

for j=1:N;

x=Xt(:,j);

u=x'\*w; G= -exp(-u^2/2); g=u\*exp(-u^2/2);

g\_der=(1-u^2)\*exp(-u^2/2); Xg=[Xg x\*g];

g\_der\_vec=[g\_der\_vec g\_der]; Gvec= [Gvec G];

end;

w\_old=w;

w=mean(Xg')'-(mean(g\_der\_vec)\*w); GG(iter)=mean(Gvec);Gsave=Gvec;

w=w/norm(w);

end

end

disp('Wtinv\*Wt');Wtinv\*Wt

Shat=Wt\*X;

Shat1=Shat(1,:); Shat2=Shat(2,:);if M==3, Shat3=Shat(3,:); end;

% shat -> estimated source signal

% -------------------------------------------------------------

# **References:**

[1] A. Hyvärinen, J. Karhunen and E. Oja, *Independent Component Analysis,* NY: John Wiley and Sons, 2001.

[2] A. Hyvärinen and E. Oja, “Independent Component Analysis: algorithms and applications”, *Neural Networks*, Vol. 13, pp. 411-430, 2000.

[3] M. Fleury, R.P. Self, and A. C. Downton, *Development of a Fine-grained Parallel Karhunen-Loève Transform*, University of Essex