Source Separation of Astrophysical Images Using Particle Filters: Matlab Codes - Version 1.0

Mauro Costagli
ISTI-CNR Via Moruzzi,1 - 56124 Pisa

October 2003

1 Introduction

This document describes the Matlab codes used to perform the simulations presented in [2]. Our aim is to separate a superposition of astrophysical images: most of the previous work on the problem perform a blind separation, assume noiseless models, and in the few cases when noise is taken into account it is generally assumed to be Gaussian and space-invariant. The code presented here has been implemented for the non-blind solution of the source separation problem using the approach named *particle filtering*. This method is an advanced Bayesian estimation technique which can deal with non-Gaussian and nonlinear models, and additive space-varying noise, in the sense that it is a generalization of the Kalman Filter. In this work, particle filters are utilized with objectives of both noise filtering and separation of signals: this approach is extremely flexible, as it is possible to exploit the available a-priori information about the statistical properties of the sources through the Bayesian theory.

The codes presented here have been developed by the author himself, as an extension of a more general-purpose code written by Alijah Ahmed, in order to deal with the astrophysical context presented in [2]. Especially in case of low SNR, the simulations show that the output quality of the separated signals is better than that of ICA, which is one of the most widespread methods for source separation. On the other hand, since a wide set of parameters, which can take from a large range of values, has to be initialized, the use of this approach needs extensive experimentation and testing.

2 Structure of the Algorithm

2.1 particle_resR_1_0.m

In order to perform the source separation, the user has to run the function called "particle_resR_1_0" presented in Section 3.1. This function accepts the following input data:

- *The observed signals*: the observed images need to be re-parametrized into vectors. The input variable y is a matrix of n rows (where n is the number of observed signals) and T columns (where T is the number of pixels of each observed mixture). If an image is re-parametrized

into a vector by a simple column-by-column scanning scheme, detrimental sudden variations of the observed signal may occur, because of the fact that the last pixel of a column can be very different from the first pixel of the next column. In order to avoid this problem, a simple code (named "buildvector", presented in Section 3.3) has been implemented, in order to re-parametrize the images into vectors, and keep the correlation between adjacent pixels.

- *Number of Gaussian components*: q is number of Gaussian components for each source.
- *Number of the sources*: m is the number of sources to be separated.
- *Number of particles*: N is the number of particles that have to be generated in order to approximate each distribution of interest, at each step t, $1 \leq t \leq T$.
- *Total number of pixels*: T is the total number of pixels of each image.
- *Starting values for the mean values and standard deviation values of the Gaussian components*: these values have to be contained in the $m \times q$ matrices called *mus* and *sigmas* respectively. For example, $mus(2,3)$ contains the mean value of the third Gaussian component of the second source.
- *RMS noise values*: the RMS noise matrices (which are supposed to be known) need to be re-parametrized like the observed images, with the same strategy, in order to obtain a matrix (named *rms*) of n rows and T columns.
- *Starting values for the mixing matrix*: the starting mixing matrix must have constant unity diagonal, and then re-parametrized into the vector H, in accordance with [2], page 47.

The output provided by the code consists of four variables:

- *MMSE estimates of the sources*: this variable is called *mmse_alpha*, and it is a matrix of m rows and T columns. The m^{th} row contains the estimate of the m^{th} source. If the observations y have been created using "buildvector", it is possible to re-parametrize each row of *mmse_alpha* into the corresponding image by means of the code named "buildmatrix", presented in Section 3.4.
- *Track of the mixing matrix*: this $nm \times T$ variable is called *htrack*, and contains the evolution of the mixing matrix coefficients.
- *Track of the mean values and standard deviation values of the Gaussian components*: the $m \times q \times T$ variables named *meantrack* and *lsigtrack* contain the evolution of the statistical parameters of each Gaussian component.

This code implements the algorithm presented in [2], Chapter 4, which can be divided into four main steps.

1. *Initialization*: before starting the on-line estimation of the sources, the algorithm initializes the set of needed variables.

2. *Sampling*: "particle_resR_1_0" calls the function "sample_prior_1_0", which generates N particles which will be used to approximate the densities of all the parameters of interest, exploiting the a-priori information on the statistical properties of the sources. This function will be described in the next subsection.

3. *Kalman Filtering*: this step has to be performed in order to estimate the mixing matrix and the importance weights.

4. *Selection / Resampling*: this section has been implemented by Alijah Ahmed, and performs the Selection step by means of the Residual Resampling scheme.

2.2 sample_prior_1_0.m

This function is called by "particle_resR_1_0", and performs the sampling step. The importance function is chosen to be the prior distribution [2], which can be factorized into:

$$\begin{aligned} p(\tilde{\boldsymbol{\theta}}_t|\tilde{\boldsymbol{\theta}}_{t-1}) &= p(\boldsymbol{\alpha}_{1:m,t}, \mathbf{z}_{1:m,t}, \boldsymbol{\pi}_t, \{\mu_{i,j,t}\}, \{\sigma^2_{i,j,t}\}|\tilde{\boldsymbol{\theta}}_{t-1}) \\ &= p(\boldsymbol{\alpha}_{1:m,t}|\mathbf{z}_{1:m,t}, \{\mu_{i,j,t}\}, \{\sigma^2_{i,j,t}\}) \times \\ &\quad p(\{\mu_{i,j,t}\}|\{\mu_{i,j,t-1}\}, \mathbf{z}_{i,t}) \times \\ &\quad p(\{\sigma^2_{i,j,t}\}|\{\sigma^2_{i,j,t-1}\}, \mathbf{z}_{i,t}) \times \\ &\quad p(\mathbf{z}_{1:m,t}|\mathbf{z}_{1:m,t-1}, \boldsymbol{\pi}_t) \times \\ &\quad p(\boldsymbol{\pi}_t|\boldsymbol{\pi}_{t-1}). \end{aligned}$$

This hierarchical structure allows us to obtain an approximation of the source distributions exploiting the particles generated from the distributions of the other parameters, sampling subsequently from $p(\boldsymbol{\pi}_t|\boldsymbol{\pi}_{t-1})$, $p(\mathbf{z}_{1:m,t}|\mathbf{z}_{1:m,t-1}, \boldsymbol{\pi}_t)$, $p(\{\sigma^2_{i,j,t}\}|\{\sigma^2_{i,j,t-1}\}, \mathbf{z}_{i,t})$, $p(\{\mu_{i,j,t}\}|\{\mu_{i,j,t-1}\}, \mathbf{z}_{i,t})$, and finally obtain the particles that approximate the source distributions $p(\boldsymbol{\alpha}_{1:m,t}|\mathbf{z}_{1:m,t}, \{\mu_{i,j,t}\}, \{\sigma^2_{i,j,t}\})$. The inputs of the function "sample_prior_1_0.m" are:

- The particles generated at the previous step: they are called $p1$ (for the transition matrix), $p2$ (for the indicator matrix), $p3$ (for the mean values), $p4$ (for the standard deviations) and $p7$ (for the sources).
- Other variables: this function needs the parameters n (number of observations), m (number of sources), q (number of Gaussian components), N (number of particles) and t (index of the current pixel).

The output variables contain the particles related to the transition matrix ($pp1$), indicator matrix ($pp2$), mean values ($pp3$), standard deviations ($pp4$) and sources ($pp7$). Actually, $pp4$ contains the logarithm of the standard deviation, according to [1], page 165 (Jeffreys' prior).

3 Matlab Source Files

3.1 particle_resR_1_0.m

```
function[mmse_alpha,htrack,meantrack,lsigtrack]=particle_resR_1_0(y,q,m,N,mus,sigmas,rms,H)

% INPUTS:

% n : number of observed signals
% y : matrix of mixed signals i.e. observations (n rows, T columns)
% q : number of Gaussian components used to model source densities
% m : number of sources
% N : number of particles
% T : total number of pixels
% mus : this matrix contains the means of all the Gaussian components (m rows, q columns)
% sigmas : this matrix contains the standard deviations (m rows, q columns)
% rms : RMS noise vectors (n rows, T columns)
% H: vector (q*m rows) containing the starting values for the mixing matrix with constant
%                                       unity diagonal, in accordance with [2], page 47

% OUTPUTS:

% mmse_alpha : mmse estimate of the sources
% htrack : track of mixing matrix (re-parametrized into a column vector of n*m rows)
% meantrack : track of the mean values of the Gaussian components
% lsigtrack : track of the standard deviations of the Gaussian components

n=size(y,1);
T=size(y,2);
htrack=ones(n*m,T);     % Vector which stores the mixing coefficients ([2], page 46-47)
meantrack=zeros(m,q,T); % Matrix which stores the mean values of the Gaussian components
lsigtrack=zeros(m,q,T); % Matrix which stores the variances of the Gaussian components
wt=(1/N)*ones(N,1);     % weights of the particles

% -------------------------------------------
% index out of n*m which is fixed due to constant diagonal constraint
for k=1:m,
    idxc(k)=(k-1)*(m+1)+1;
end
idxall=[1:n*m];
idx_c=idxall;
for k=1:m,
    idx_c=idx_c(find((idx_c)~=idxc(k)));
end
% -------------------------------------------

%initialising the sets of particles
pi_mat=zeros(m,q,N);   % particles for the pi_matrix
z=ceil(rand(m,N)*q);   % particles for the indicator matrix
mu_ij=randn(m,q,N);    % particles for the mean values of the Gaussian components
l_sig_ij=randn(m,q,N); % particles for the standard deviations of the Gaussian components
l_sig_v=ones(n*m,N);   % particles for the state noise variance
alpha=randn(m,N);      % particles for the sources
```

```
mmse_alpha=zeros(m,T); % minimum mean-squared error estimate of sources

% Initialising the Gaussian components with their input values
for k1=1:N,
    mu_ij(:,:,k1)= mus(:,:);
    l_sig_ij(:,:,k1)=log(sigmas(:,:)); % in order to generate Gamma-distributed particles
%                                      (non-informative Jeffreys' prior, [1], page 165)
    for k2=1:m,
     pi_mat(k2,:,k1)=ones(1,q)*1/q;               % uniform distribution
     std_v=exp(l_sig_ij(k2,z(k2,k1),k1));       % parameter to generate alpha
     mean_v=mu_ij(k2,z(k2,k1),k1);              % parameter to generate alpha
     alpha(k2,k1)=randn(1)*std_v+mean_v;  % source alpha
    end
end

%=========================================================================================
% Initialising Kalman filter parameters (see [1], page 235)
tj_mkk=zeros(n*m,1,N,T);
mkk_hold=zeros(n*m,1,N);
m00=ones(n*m,1,N);
mkk_1=zeros(n*m,1,N);
mkk=zeros(n*m,1,N);
P00=zeros(n*m,n*m,N);
Pkk_1=zeros(n*m,n*m,N);
Pkk=zeros(n*m,n*m,N);
ykk_1=zeros(n,1,N);
Sk=zeros(n,n,N);
for k1=1:N,
    m00(:,1,k1)=H;   % in order to use the input values
    for k2=1:length(idxc),
        m00(idxc(k2),1,k1)=1;
        for k3=1:length(idx_c),
            P00(idx_c(k3),idx_c(k3),k1)=0.001;
        end
    end
end

% Matrices of the model (see [1], page 162)
A_th=zeros(n*m,n*m,N);
B_th=zeros(n*m,n*m,N);
C_th=zeros(n,n*m,N);
D_th=zeros(n,n,N);
for k1=1:N,
    A_th(:,:,k1)=eye(n*m);
    for k2=1:length(idx_c),
        B_th(idx_c(k2),idx_c(k2),k1)=exp(0.5*l_sig_v(idx_c(k2),k1));
    end
    C_th(:,:,k1)=kron(alpha(:,k1)',eye(n));
end
% End of Initialising Kalman filter parameters
% -------------------------------------------------------------------------

% =========================================================================
% START ITERATIONS
```

```
% ======================================================================
for t=1:T,

    if rem(t,256)==0,
        disp(t);       % to display t every 256 steps.
    end

% Propose new values
    p1=pi_mat; p2=z;
    p3=mu_ij; p4=l_sig_ij;
    p7=alpha;
 % propose from prior importance function
    [pp1,pp2,pp3,pp4,pp7]=sample_prior_1_0(p1,p2,p3,p4,p7,n,m,q,N,t);
    pi_mat=pp1; z=pp2;
    mu_ij=pp3; l_sig_ij=pp4;
    l_sig_v=-1000*ones(n*m,N);
    alpha=pp7;

% proposing for matrices A, B, C.
    A_th_prev=A_th;
    for k1=1:N,
        for k2=1:length(idx_c),
            A_th(idx_c(k2),:,k1)=A_th_prev(idx_c(k2),:,k1)+0.001/t*randn(1,n*m);
        end
        for k2=1:length(idx_c),
            B_th(idx_c(k2),idx_c(k2),k1)=exp(0.5*l_sig_v(idx_c(k2),k1));
        end
        C_th(:,:,k1)=kron(alpha(:,k1)',eye(n));
        for k2=1:n,
            D_th(k2,k2,k1)=rms(k2,t); % uses the RMS noise input
        end
    end
%=========================================================================
    if t==1,% first step of the Kalman Filter ([1], page 235)
        for l=1:N,
            mkk_1(:,:,l)=A_th(:,:,l)*m00(:,:,l);
            Pkk_1(:,:,l)=A_th(:,:,l)*P00(:,:,l)*A_th(:,:,l)'+B_th(:,:,l)*B_th(:,:,l)';
            ykk_1(:,:,l)=C_th(:,:,l)*mkk_1(:,:,l);
            Sk(:,:,l)=C_th(:,:,l)*Pkk_1(:,:,l)*C_th(:,:,l)'+D_th(:,:,l)*D_th(:,:,l)';
            mkk(:,:,l)=mkk_1(:,:,l)+Pkk_1(:,:,l)*C_th(:,:,l)'* pinv(Sk(:,:,l))*(y(:,t)-ykk_1(:,:,l));
            Pkk(:,:,l)=Pkk_1(:,:,l)-Pkk_1(:,:,l)*C_th(:,:,l)'*pinv(Sk(:,:,l))*C_th(:,:,l)*Pkk_1(:,:,l);
        end
        else      % Kalman Filter for t > 1 ([1], page 235)
        for l=1:N,
             mkk_1(:,:,l)=A_th(:,:,l)*mkk(:,:,l);
             Pkk_1(:,:,l)=A_th(:,:,l)*Pkk(:,:,l)*A_th(:,:,l)'+B_th(:,:,l)*B_th(:,:,l)';
             ykk_1(:,:,l)=C_th(:,:,l)*mkk_1(:,:,l);
             Sk(:,:,l)=C_th(:,:,l)*Pkk_1(:,:,l)*C_th(:,:,l)'+D_th(:,:,l)*D_th(:,:,l)';
             mkk(:,:,l)=mkk_1(:,:,l)+Pkk_1(:,:,l)*C_th(:,:,l)'* pinv(Sk(:,:,l))*(y(:,t)-ykk_1(:,:,l));
             Pkk(:,:,l)=Pkk_1(:,:,l)-Pkk_1(:,:,l)*C_th(:,:,l)'*pinv(Sk(:,:,l))*C_th(:,:,l)*Pkk_1(:,:,l);
        end
    end
%=========================================================================
```

```
% variable to store trajectories
    tj_mkk(:,:,1:N,t)=mkk;  % tj_mkk is a variable used in the RESAMPLING STEP

%=========================================================================
    %Calculating the weights
%=========================================================================
    for l=1:N, %tmp_scal is used to calculate the weights wt
        tmp_scal=(y(:,t)-ykk_1(:,:,l))'*inv(Sk(:,:,l))*(y(:,t)-ykk_1(:,:,l));
        wt(l)=1/sqrt(det(Sk(:,:,l)))*exp(-0.5*tmp_scal);
    end
    wt=wt./sum(wt); %normalized weights
%=========================================================================
    if sum(wt)>0.5,   % ELSE: there is a problem with the weights (their sum is not equal to 1)

        meantrack(:,:,t)=zeros(m,q,1);
        lsigtrack(:,:,t)=zeros(m,q,1);
        htrack(:,t)=zeros(n*m,1);
        % calculating the trajectories of the parameters of interest:
        for l=1:N,
            htrack(:,t)=htrack(:,t)+wt(l)*mkk(:,:,l);
            for k1=1:m,
                for k2=1:q,
                    meantrack(k1,k2,t)=meantrack(k1,k2,t)+wt(l)*mu_ij(k1,k2,l);
                    lsigtrack(k1,k2,t)=lsigtrack(k1,k2,t)+wt(l)*exp(l_sig_ij(k1,k2,l));
                end
            end
        end
        for k1=1:N,
            mmse_alpha(:,t)=mmse_alpha(:,t)+wt(k1)*alpha(:,k1); % MMSE estimate of the sources
        end

    %=====================================================================
    % START OF THE SELECTION / RESAMPLING STEP (implemented by A. Ahmed)
    %=====================================================================
    % Selection via Systematic sampling
        pi_mat_tmp=pi_mat;
        z_tmp=z;
        mu_ij_tmp=mu_ij;
        l_sig_ij_tmp=l_sig_ij;
        l_sig_v_tmp=l_sig_v;
        alpha_tmp=alpha;
        mkk_tmp=mkk;
        Pkk_tmp=Pkk;
        tmp_tj=tj_mkk;
        tmp_A_th=A_th;
    % RESIDUAL RESAMPLING PROCEDURE
    % first integer part
        wn_res=(N.*wt)';
        N_sons=fix(wn_res);
    % residual number of particles to sample
        N_res=N-sum(N_sons);
        if (N_res~=0)
            wn_res=(wn_res-N_sons)/N_res;
    % generate the cumulative distribution
```

```
            dist=cumsum(wn_res);
    % generate N_res ordered random variables uniformly distributed in [0,1]
            u = fliplr(cumprod(rand(1,N_res).^(1./(N_res:-1:1))));
            j=1;
            for i=1:N_res
                while (u(1,i)>dist(1,j))
                    j=j+1;
                end
                N_sons(1,j)=N_sons(1,j)+1;
            end
        end
%=============================================================================
% Replicating particles with strong weights
        ind=1;
        for i=1:N
        % if copy then keep it here
             if (N_sons(1,i)>0)
                 tp1=pi_mat_tmp(:,:,i);
                 tp2=z_tmp(:,i);
                 tp3=mu_ij_tmp(:,:,i);
                 tp4=l_sig_ij_tmp(:,:,i);
                 tp6=l_sig_v_tmp(:,i);
                 tp7=alpha_tmp(:,i);
                 tpH=mkk_tmp(:,:,i);
                 tpP=Pkk_tmp(:,:,i);
                 tpH_t=tmp_tj(:,:,i,1:t);
                 tpA_th=tmp_A_th(:,:,i);

                 for j=ind:ind+N_sons(1,i)-1
                     pi_mat(:,:,j)=tp1;
                     z(:,j)=tp2;
                     mu_ij(:,:,j)=tp3;
                     l_sig_ij(:,:,j)=tp4;
                     l_sig_v(:,j)=tp6;
                     alpha(:,j)=tp7;
                     mkk(:,:,j)=tpH;
                     Pkk(:,:,j)=tpP;
                     tj_mkk(:,:,j,1:t)=tpH_t;
                     A_th(:,:,j)=tpA_th;
                 end
              end
              ind=ind+N_sons(1,i);
         end
%=============================================================================================
%           END OF THE SELECTION / RESAMPLING STEP (implemented by A. Ahmed)
%=============================================================================================

    else              % if the algorithm enters here, it means there is a problem with the weights
        mmse_alpha(2:m,t)=y(2:m,t); % estimate of the foreground source = observation
        mmse_alpha(1,t)=mmse_alpha(1,t-1); % estimate of the CMB
        t
    end
end                   % END ITERATIONS
```

3.2 sample_prior_1_0.m

```
function[pp1,pp2,pp3,pp4,pp7]=sample_prior_1_0(p1,p2,p3,p4,p7,n,m,q,N,t)

% this function is called by particle_resR_1_0.m

% INPUTS and OUTPUTS:
%
% p1 : particles for pi_mat, for the pixel t-1 (output for the pixel t: pp1)
% p2 : particles for the z, for the pixel t-1 (output for the pixel t: pp2)
% p3 : particles for the mean values, for the pixel t-1 (output for the pixel t: pp3)
% p4 : particles for the standard deviations, for the pixel t-1 (output for the pixel t: pp4)
% p7 : particles for the source, for the pixel t-1 (output for the pixel t: pp7)
% n  : number of observations
% m  : number of sources
% q  : number of Gaussian components of each source
% N  : number of particles
% t  : pixel to be examined

% initialisation
pp1=zeros(m,q,N);
pp2=zeros(m,N);
pp3=zeros(m,q,N);
pp4=zeros(m,q,N);
pp7=zeros(m,N);

%**************************************************************************
    % Sampling sub-optimally from p(theta_t|theta_t-1)

for l=1:N,

    % Sampling from p(pi_mat|pi_mat_prev)
    for k1=1:m,
        tmp_vect=p1(k1,:,l);
        tmp_vect2=0*tmp_vect;
        inc_val=0.01;
        for k3=1:q,
            tmp_value=tmp_vect(k3)+2*(rand(1)-0.5)*inc_val;
            if tmp_value>0
                tmp_vect2(k3)=tmp_value;
            else
                tmp_vect2(k3)=tmp_vect(k3);
            end
        end
        tmp_vect2=tmp_vect2/sum(tmp_vect2);
        pp1(k1,:,l)=tmp_vect2;
    end

    % Sampling from p(z_t|pi_mat) : Dirichlet prior
    for k1=1:m,
        tmp_mat=pp1(k1,:,l);
        p_vect=tmp_mat;
        cum_p_vect=cumsum(p_vect);
        u=rand(1);
```

```
        g=min(find(cum_p_vect>u));
        pp2(k1,l)=g;
        if k1==1,
            pp2(k1,l)=1;      % The first source is the CMB (only one Gaussian component)
        end
    end

    % Sampling from p(mu_ij_t|mu_ij_t-1) and p(l_sig_ij_t|l_sig_ij_t-1)
    for k1=1:m,
        for k2=1:q,
            inc_val=0.0000001;
            pp3(k1,k2,l)=randn(1)*inc_val+p3(k1,k2,l); % Gaussian prior
            pp4(k1,k2,l)=randn(1)*inc_val+p4(k1,k2,l); % Jeffreys' prior: log(sigma)
        end
    end

    % Sampling source alpha
    for k1=1:m,
        std_v=sqrt(exp(pp4(k1,pp2(k1,l),l)));
        mean_v=pp3(k1,pp2(k1,l),l);
        pp7(k1,l)=randn(1)*std_v+mean_v;
    end

end
%************************************************************************
```

3.3 buildvector.m

```
function[source_vec]=buildvector(source_matrix,dim);

% INPUT :

% source_matrix : square source image
% dim : dimension of the source image

% OUTPUT :

% source_vec : the source image is re-parametrized into a column vector
%              which keeps the correlation between pixels - the source
%              image is scanned column by column, one column from the
%              top to the bottom, and the following one is scanned from
%              the bottom to the top, and so on...

i=1;
j=1;
k=1;
for cicle = 1:dim/2,
    while i<dim+1
        source_vec(k) = source_matrix(i,j);
        i=i+1;
        k=k+1;
    end
    i=i-1;
    j=j+1;
    while i>0
        source_vec(k)=source_matrix(i,j);
        i=i-1;
        k=k+1;
    end
    i=i+1;
    j=j+1;
end
```

3.4 buildmatrix.m

```
function[matrix]=buildmatrix(vector, dim)

% notation : see buildvector.m

i=1;
j=1;
k=1;
for cicle = 1:dim/2,
    while i<dim+1
        matrix(i,j) = vector(k);
        i=i+1;
        k=k+1;
    end
    i=i-1;
    j=j+1;
    while i>0
        matrix(i,j) = vector(k);
        i=i-1;
        k=k+1;
    end
    i=i+1;
    j=j+1;
end
imagesc(matrix);
```

References

[1] Ahmed A.: "Signal Separation",
PhD Thesis, Signal Processing Group, Department Of Engineering,
University Of Cambridge, U.K. (2000)
http://www-sigproc.eng.cam.ac.uk/publications/archive/AlijahAhmedThesis.zip

[2] Costagli M.: "Bayesian Source Separation of Astrophysical Images Using Particle Filters",
Laurea Master Thesis, Dipartimento Di Ingegneria Della Informazione,
Università Di Pisa, Italy (2003)
http://etd.adm.unipi.it/theses/available/etd-09242003-174956/