Blind Source Separation

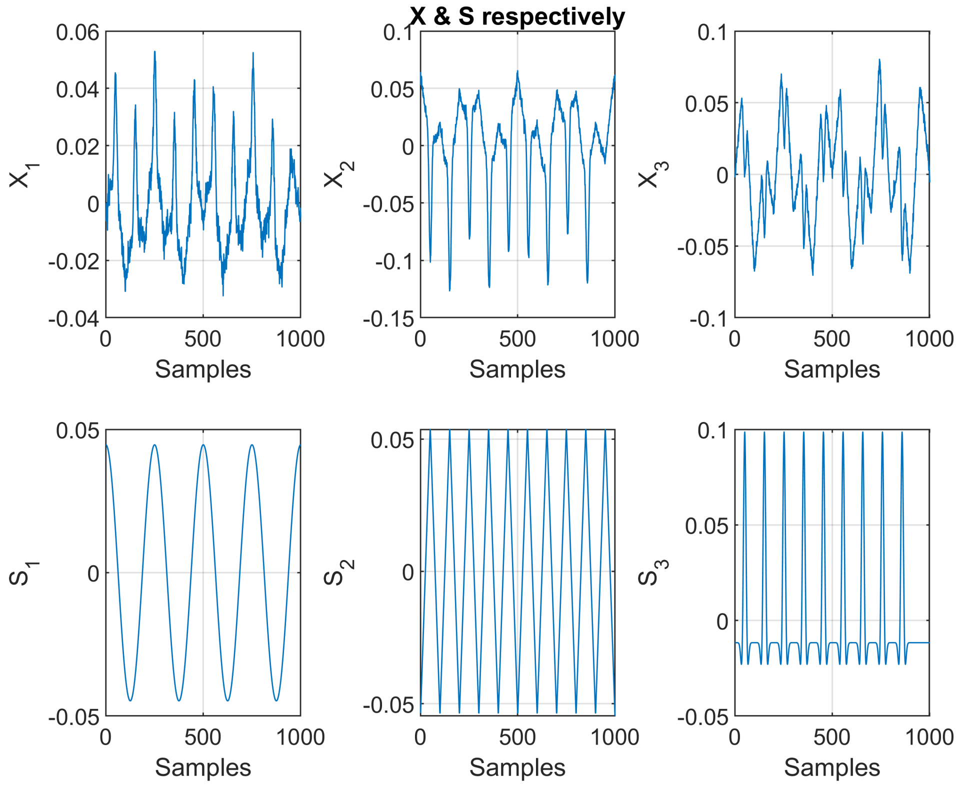
HW9-Section-1

Mohammadreza Arani :::::::::::: 810100511

1402/03/14

```
clear; clc; close all;

Data_hw9 = load("hw9.mat");

A = Data_hw9.A;
S = Data_hw9.S;
Noise = Data_hw9.Noise;

X = A*S+Noise;

figure()
subplot(2,3,1)
plot(X(1,:))
grid on
xlabel("Samples")
ylabel("X_1")


subplot(2,3,2)
plot(X(2,:))
grid on
xlabel("Samples")
ylabel("X_2")
title("X & S respectively")

subplot(2,3,3)
plot(X(3,:))
grid on
xlabel("Samples")
ylabel("X_3")


subplot(2,3,4)
plot(S(1,:))
grid on
xlabel("Samples")
ylabel("S_1")

subplot(2,3,5)
plot(S(2,:))
grid on
xlabel("Samples")
```

```
ylabel("S_2")

subplot(2,3,6)
plot(S(3,:))
grid on
xlabel("Samples")
ylabel("S_3")
```



Deflation Mode:

In this Mode, we don't have to calculate the second term of cost function and also the stepsize! This happens when matrix "B" is orthonormal!

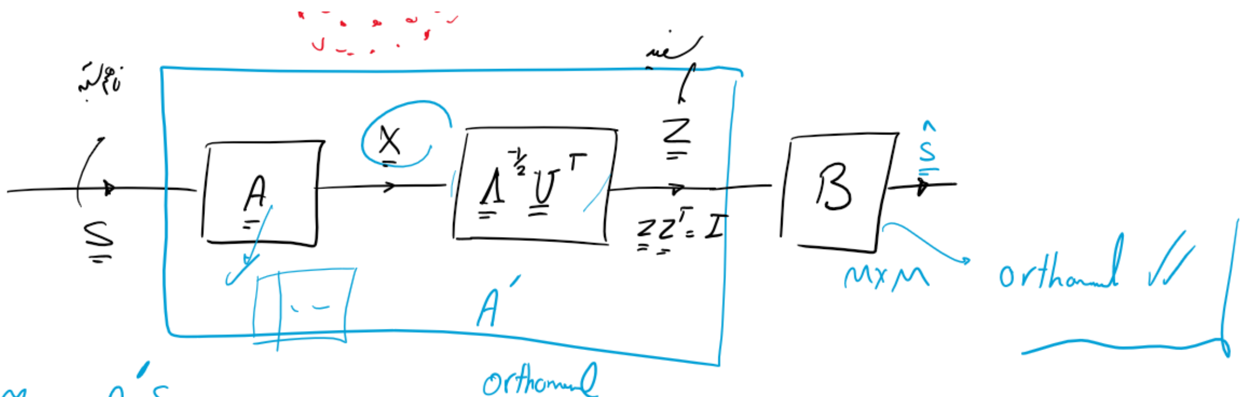
It's been proven many times that "B" must be orthonormal!

$$\underline{\underline{X}}_{M\times T} \longrightarrow \{\underline{u}_1', \dots, \underline{u}_M\} \longrightarrow \underline{\underline{R}}_{x\, M\times M} = \underline{\underline{X}}\,\underline{\underline{X}}^T$$

$$[\underline{U}, \underline{\underline{\Lambda}}] = eig(\underline{\underline{R}}_x) \; ; \quad \underline{\underline{R}}_x = \underline{U}\,\underline{\underline{\Lambda}}\,\underline{U}^T = \sum_{m=1}^{M} \lambda_m \underline{u}_m \underline{u}_m^T$$

$$\boxed{\underline{\underline{Z}}_{M\times T} = \underline{U}^T_{M\times M}\, \underline{\underline{X}}_{M\times T}} \; ; \quad \underline{\underline{Z}}\,\underline{\underline{Z}}^T = R_z = (U^T \underbrace{X X^T}_{R_x} U) =$$

And proof of "B" being orthonormal is:



$$\underline{Z} = \underline{A}'\underline{S}$$

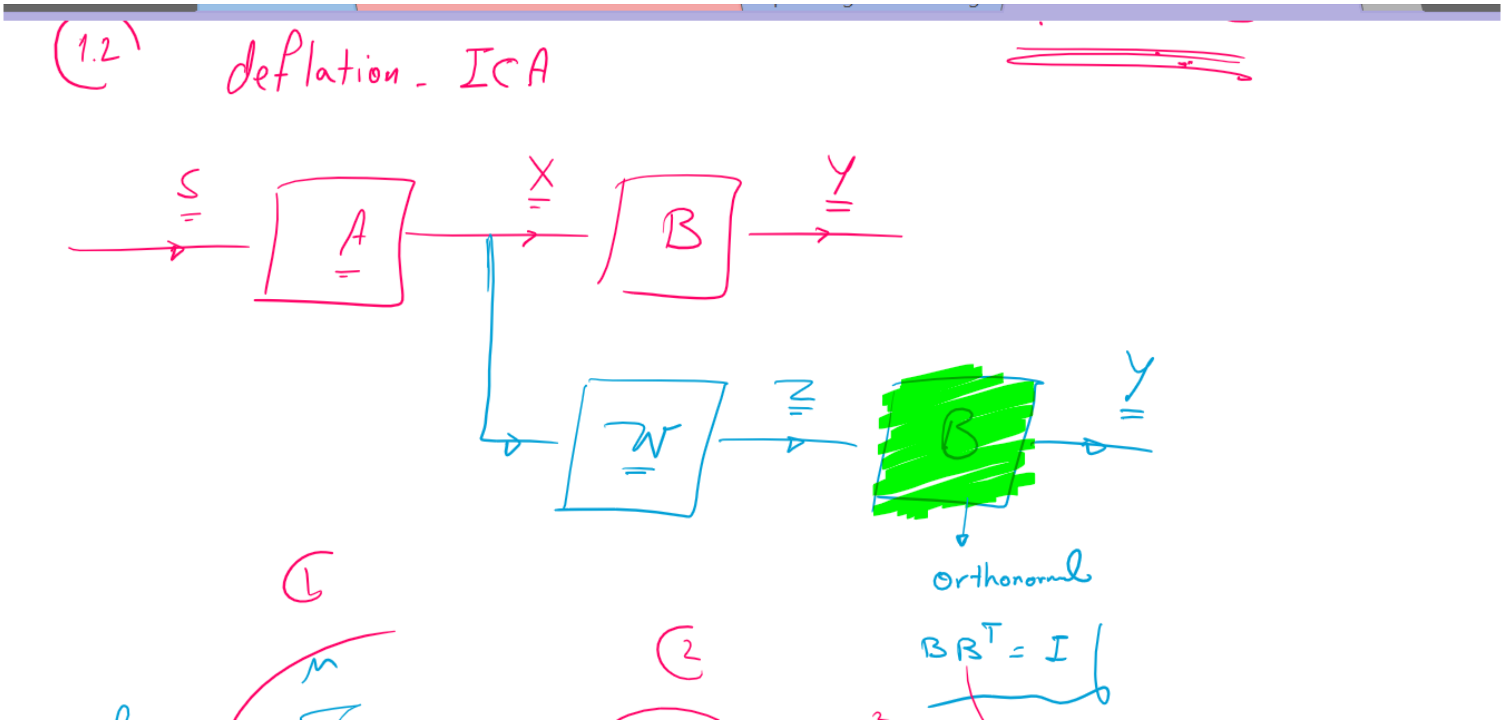
$$\underline{Z}\,\underline{Z}^T = \underline{I} \longrightarrow (A'S)(A'S)^T = I$$

$$\longrightarrow A' \underbrace{\boxed{\underline{S}\,\underline{S}^T}}_{I} A'^T = I \longrightarrow \underline{A}'\underline{A}'^T = \underline{I}$$

Step-1: Whitening:

```
[U , Gamma] = eig(X*X');
W = Gamma^(-0.5);
Z = W*U'*X; % Whitened Data

R_z = Z*Z';
disp(R_z);
```

```
    1.0000    0.0000   -0.0000
    0.0000    1.0000    0.0000
   -0.0000    0.0000    1.0000
```



$$f(B) = \sum_{m=1 \text{ to } M} H(y_m) - H(y); \quad \text{where } H(y) = H(z) + \log(\det(B)); \quad \text{Having "B" to be Orthonormal we have } \log(\det(B)) = \text{cte};$$

$$\frac{\partial}{\partial B} f(B) = \frac{\partial}{\partial B} \sum_{m=1 \text{ to } M} H(y_m) - 0;$$

① $\min H(y_1)$ s.t. $b_1^T b_1 = 1$

② $\min H(y_2)$ s.t. $b_2^T b_2 = 1$, $b_2^T b_1 = 0$

$M-1$

M $\min H(y_M)$ s.t. $b_M^T b_M = 1$, $b_M^T b_1 = 0$, $b_M^T b_2 = 0$, $\vdots$, $b_M^T b_{M-1} = 0$

$$BZ = Y$$

It is called deflation because we are performing minimization over each row of B independently!

$$f(B) = \sum_{m=1}^{M} H(y_m)$$

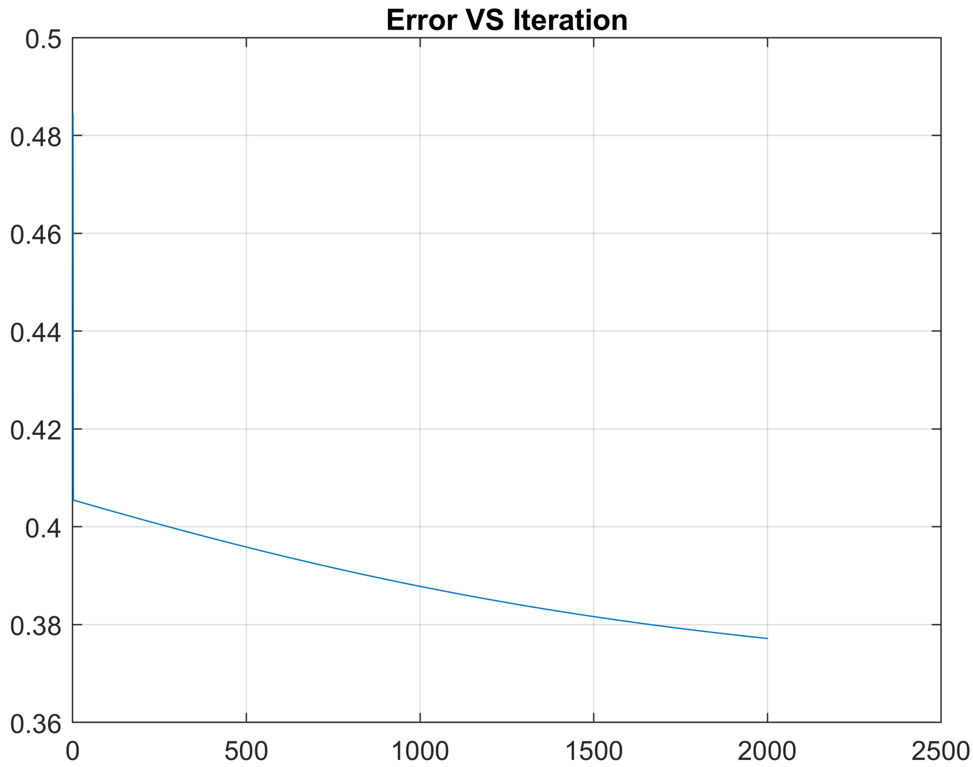
b_m^T

$$\begin{bmatrix} b_1^T \\ b_2^T \\ \vdots \\ b_M^T \end{bmatrix}$$

Step-2: Perform ALternation Minimization for each column of B!

```
B = generate_orthonormal_matrix(size(A,1))  ; % Because B and A are in the same size due to the



mu       = 1e+02;
Max_Iter = 2e+3;
thresh_cntr = 1e-1;
thresh_B = 1e-6;

Error_Iter_deflate = zeros(1,Max_Iter)+inf;
cntr = 1;

y_hat = B*Z;
MIN_ERR = inf;
while(true)
    B_prev = B;

    % Update B:
```

```
    for i=1:length(B) % Each Row
         % Estimation of Score Function:
         [Theta_hat , Score_Func_y ] = Theta_Calc_Kernel(y_hat(i,:));
         StepSize = ( Score_Func_y*Z' )/length(Z) ;
        %  StepSize = normalize(StepSize,2,"norm");
         B(i,:)         =  B(i,:) - mu*StepSize  ;
         B(i,:)         =  B(i,:)/norm(B(i,:));   % Normalization
         B(i,:)         =  ( eye(size(B)) - B(1:i-1,:)'*B(1:i-1,:)   )* B(i,:)';  % Orthogonalit

    end

    [Error_Perm,y_Hat_Chosen,B] = Perm_AMP_Disamb(B,S,Z);
    Error_Iter_deflate(cntr)     = min(Error_Perm);
    y_hat = B*Z;
%     Errors_ICA = norm(y-S)/norm(S);
%     Error_Iter_deflate(p) = min(Errors_ICA);




    % Check Convergence:
    if( (abs(Error_Iter_deflate(1,cntr))<thresh_cntr) || (cntr>Max_Iter) || ( norm(B_prev - B,
        break;
    end
    if ( Error_Iter_deflate(cntr)<MIN_ERR  )
        y_hat_best = y_Hat_Chosen;
        B_hat_best = B;
        Index_Best = cntr;
        MIN_ERR = Error_Iter_deflate(cntr);
    end
    cntr = cntr +1;

end

figure()
plot(Error_Iter_deflate)
grid on
title("Error VS Iteration")
```




```
disp("calculated Error Equals to: "+min(Error_Iter_deflate))
```

```
calculated Error Equals to: 0.37716
```

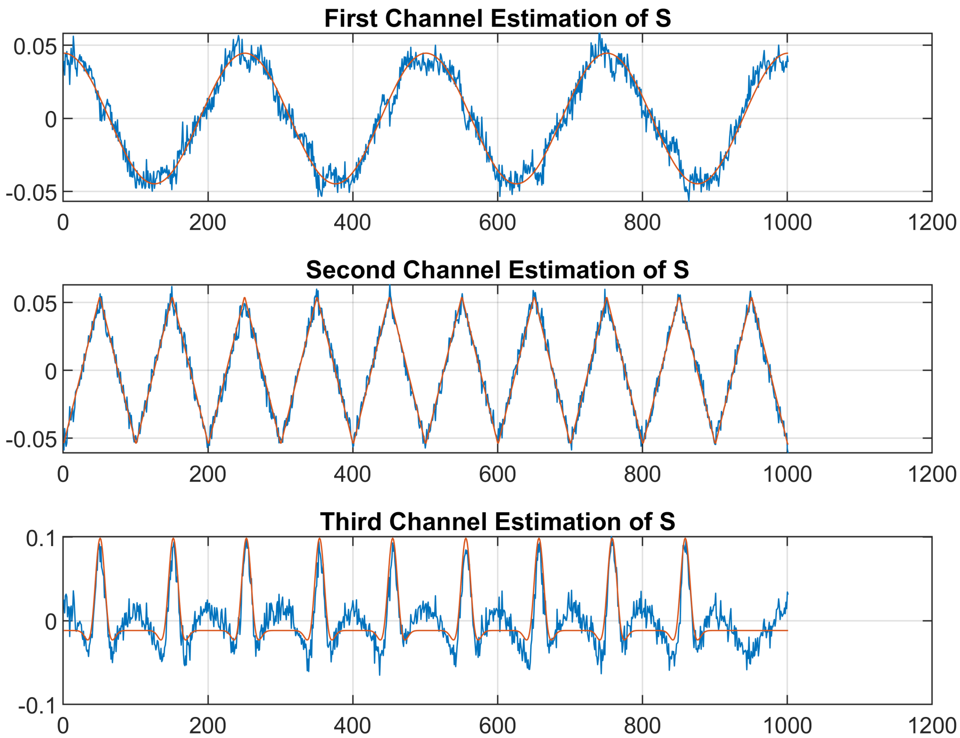
```
disp(abs(B*W*U'*A))
```

```
    0.9827    0.1515    0.1570
    0.0523    0.9949    0.0097
    0.1149    0.5675    1.1306
```

Signal Illustration:

```
% Recovered S_hat:
y = y_Hat_Chosen;


figure()
subplot(3,1,1)
plot(y(1,:))% Permutaion
hold on
plot(S(1,:))
hold off
grid on
title("First Channel Estimation of S")
```

```
subplot(3,1,2)
plot(y(2,:))
hold on
plot(S(2,:))
hold off
grid on
title("Second Channel Estimation of S")


subplot(3,1,3)
plot(y(3,:))
hold on
plot(S(3,:))
hold off
grid on
title("Third Channel Estimation of S")
```




```

```

Functions:

```
function matrix = generate_orthonormal_matrix(size)
    % Step 1: Generate a random matrix
```

```
    matrix = randn(size, size);

    % Step 2: Apply the Gram-Schmidt process
    for i = 1:size
        for j = 1:i-1
            matrix(:, i) = matrix(:, i) - dot(matrix(:, j), matrix(:, i)) * matrix(:, j);
        end
    end

    % Step 3: Normalize each column
    norms = vecnorm(matrix);
    matrix = matrix ./ norms;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Theta_hat, PSI_hat] = Theta_Calc_Kernel(y)
% Theta Hat Calculation Function for Kernel Method+MSE:

Coeff = [0,1,2,3,4,5];
N = length(Coeff);
Num_of_Channels = size(y(:,1));

Theta_hat        = zeros(Num_of_Channels(1,1),N);
PSI_hat = zeros(size(y));


    for n=1:Num_of_Channels(1,1)
        y_temp = y(n,:);
        ky = [ones(size(y_temp)) ; y_temp; y_temp.^2; y_temp.^3; y_temp.^4; y_temp.^5   ];
        ky_prime = [zeros(size(y_temp))  ; ones(size(y_temp)); 2*y_temp; 3*y_temp.^2; 4*y_temp.

        Theta_hat(n,:) = pinv(ky*ky')/length(y_temp)*mean(ky_prime,2);
        PSI_hat(n,:)   = Theta_hat(n,:)*ky;
    end

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Error_Perm,S_Hat_Chosen,B]  = Perm_AMP_Disamb(B,S,Z) %% Perm_AMP_Disamb

    Final_Result_S_hat = calculate_permutations_and_Signs(B);
    % Calc the Error:
    L3 = size(Final_Result_S_hat,3);
    Error_Perm = zeros(1,L3);
    for i=1:L3
        S_hat_temp =  Final_Result_S_hat(:,:,i)*Z;
        Error_Perm(1,i) = norm(S_hat_temp-S,"fro")/norm(S,"fro");
    end
    [~ , idx ] = min(Error_Perm);
    S_Hat_Chosen = Final_Result_S_hat(:,:,idx)*Z;
    B = Final_Result_S_hat(:,:,idx);
end
```

```
function    Final_Result = calculate_permutations_and_Signs(matrix)
    num_of_columns_matrix = size(matrix,2);
    variations = calculate_variations(matrix);
    cntr =1;
    %Temp = zeros(size(variations(:,:,1)));
    Final_Result = zeros([size(matrix),  (2^num_of_columns_matrix)*factorial(num_of_columns_mat
    for j=1:size(variations,3)
        Temp = variations(:,:,j);

         num_of_columns = size(Temp,2);
         Different_Col_Arranges = perms(1:num_of_columns);
        for i=1:size(Different_Col_Arranges,1)
            Final_Result(:,:,cntr) =  Temp(:,Different_Col_Arranges(i,:))  ;
            cntr = cntr +1;
        end
    end


end


function variations = calculate_variations(matrix)
    % Get the size of the matrix
    [num_rows, num_cols] = size(matrix);

    % Generate all possible combinations of signs
    sign_combinations = cell(1, num_rows);
    [sign_combinations{:}] = ndgrid([-1, 1]);
    sign_combinations = cellfun(@(x) x(:), sign_combinations, 'UniformOutput', false);
    sign_combinations = cat(2, sign_combinations{:});

    % Calculate the number of variations
    num_variations = size(sign_combinations, 1);

    % Initialize the variations array
    variations = zeros(num_rows, num_cols, num_variations);

    % Generate the variations
    for i = 1:num_variations
        % Apply the sign variations to each row
        variations( :, :, i) = matrix .* reshape(sign_combinations(i, :), 1, num_rows, 1);
    end
end
```