به نام خدا

Blind Source Separation (BSS)

تکلیف شماره

9

محمدرضا آرانی

810100511

دانشگاه تهران

1402/03/14

جدول محتویات

بخش اول:

در این تمرین می خواهیم جداسازی کور منابع را با فرض استقلال منابع حل کنیم.

ماتریس مخلوط کننده A ، ماتریس منابع S و ماتریس Noise در فایل hw9.mat در اختیار شما قرار داده شده است. ابتدا ماتریس مشاهدات X را با رابطه ی X = A S + Noise به دست آورید. هم منابع و هم مشاهدات بدون نویز و هم مشاهدات نویزی را رسم کنید تا ظاهر آنها را ببینید. حال به دید جداسازی کور منابع به مساله نگاه کنید. در واقع فرض می کنیم فقط ماتریس X را داریم و تعداد منابع را هم می دانیم. استراتژی ما این خواهد بود که با ضرب یک ماتریس جدا کننده B در ماتریس X، خروجی هایی تولید کنیم که این خروجی ها تا حد ممکن از هم مستقل باشند.

۱- روش دومی که در کلاس مبتنی بر کمینه سازی D_{KL} ، بعد از سفید سازی داده ها بیان شد را پیاده سازی کنید (حالت deflation). توجه داشته باشید ماتریس جدا کننده ی نهایی شما حاصلضرب ماتریس orthonormal نهایی به دست آمده از الگوریتم، در ماتریس سفید کننده است.

توضیحات این قسمت متناسب با تکلیف قبل است!

قسمت اول:

۱-۱- ماتریس جدا کننده ای که در نهایت به دست آوردید را در ماتریس مخلوط کننده ی اصلی ضرب کنید و حاصل را گزارش کنید. ماتریس حاصل باید نزدیک به یک ماتریس permutation باشد به این معنی که در هر سطر و هر ستون فقط یک مقدار غیر صفر داشته باشد.

نتایج حاصل از کد زیر در ادامه خواهند آمد:

Step-1:

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

```
Z = W*U'*X; % Whitened Data


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

Step-2:

```

B = generate_orthonormal_matrix(size(A,1))  ; % Because B and A are in the same size due
to the fact that we have M=N!







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,:)';  %
Orthogonality
```

```
    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,'fro')<thresh_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")
```

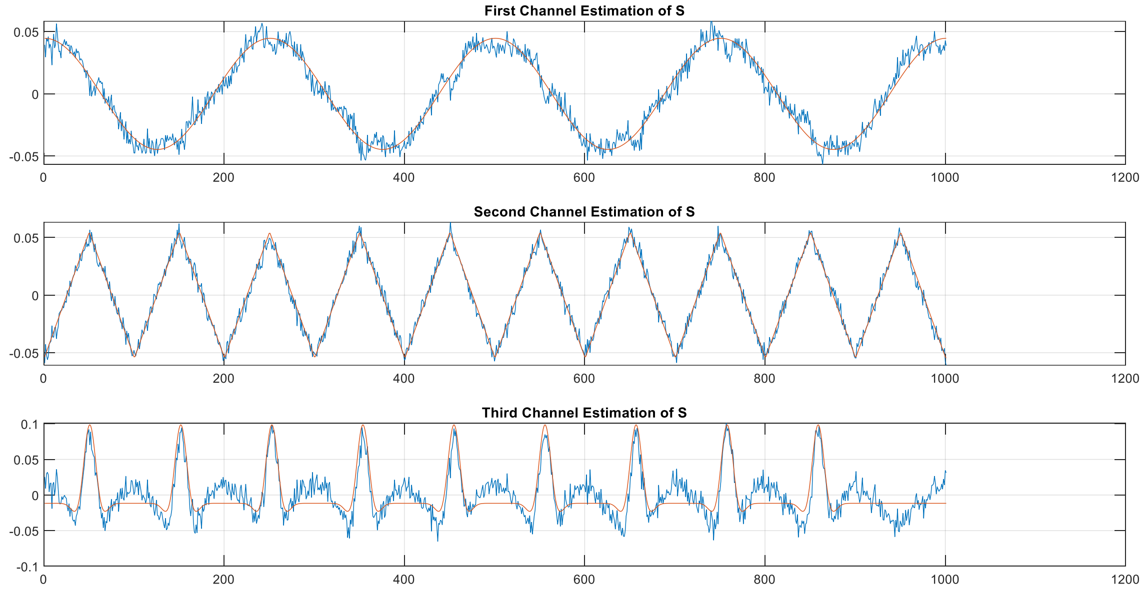
ماتریس Permutation مذکور به صورت زیر است:

```
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
```

برای مقایسه‌ی سیگنال‌های منابع داریم:

```
% 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")
```



شکل 1

در این مقایسه مشاهده می‌شود که به خوبی منابع حدس زده شده اند!

قسمت دوم:

۱-۲- ابهام ترتیب و همچنین ابهام scale منابع را برطرف کرده به این معنی که انرژی منابع تخمین زده شده را مساوی انرژی منابع اصلی کنید. منابع تخمین زده شده را روی منابع اصلی رسم کنید. و سپس مقدار خطای زیر را گزارش کنید.

$$E = \frac{\|\hat{S} - S\|_F^2}{\|S\|_F^2}$$

این کار توسط تابع زیر انجام می‌شود:

```
[Error_Perm,y_Hat_Chosen,B] = Perm_AMP_Disamb(B,S,Z);
```

در واقع این تابع به صورت زیر است:

```
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_matrix) ]);
    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
```

همانطور که در بالا آمده است این تابع با محاسبه‌ی تمامی Variations های ممکن برای ماتریس مربعی داده شده‌ی B، y متناسب را حساب کرده و بهترین را به عنوان خروجی بیرون می‌دهد.

محاسبه‌ی تمامی این Variation ها به این صورت است که:

$$Num_{of_{Vars}} = 2^N * (N)!$$

که در آن N تعداد ستون‌های ماتریس B خواهد بود.

برای مثال برای یک ماتریس 2*2 داریم:

```
% Example matrix
matrix = [1 2;3 4];

% Calculate permutations
% permutations = calculate_permutations(matrix);

Vars = calculate_permutations_and_Signs(matrix);
```

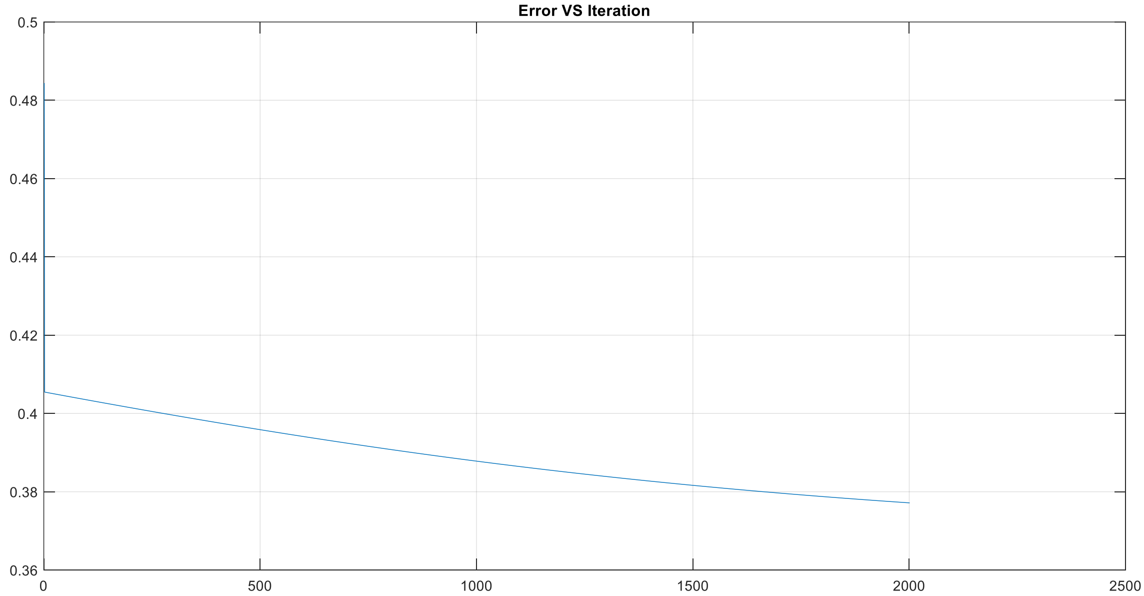
```
>> Vars

Vars(:,:,1) =    Vars(:,:,2) =    Vars(:,:,3) =    Vars(:,:,4) =

  -2   -1          -1   -2          -2    1           1   -2
  -4   -3          -3   -4          -4    3           3   -4

Vars(:,:,5) =    Vars(:,:,6) =    Vars(:,:,7) =    Vars(:,:,8) =

   2   -1          -1    2           2    1           1    2
   4   -3          -3    4           4    3           3    4
```

مشاهده می‌شود تمامی این حالت‌ها که 8 حالت اند یعنی 4 حالت علامت هر ستون و 2 حالت هم ترتیب آنها که مجموعا 8 حالت را می‌سازد در بالا قرار دارند.

قسمت سوم:

۱-۳- نمودار همگرایی (تابع هدف بر حسب شماره ی iteration) را رسم کنید.

نمودار خطا بر حسب تعداد تکرار به صورت زیر است:



شکل 2

بخش دوم:

۲- روش سومی که در کلاس ارائه شد و به ماتریس جداسازی وابسته نبود (equivariant) را پیاده سازی کنید. همه ی نتایج را مشابه قسمت ۱ گزارش کنید.

روش سومی که در کلاس مطرح شد و بر اساس این بود که به ماتریس جداسازی وابسته نباشد بر این اساس کار می‌کرد که ضرب B و A را بهینه‌کند!

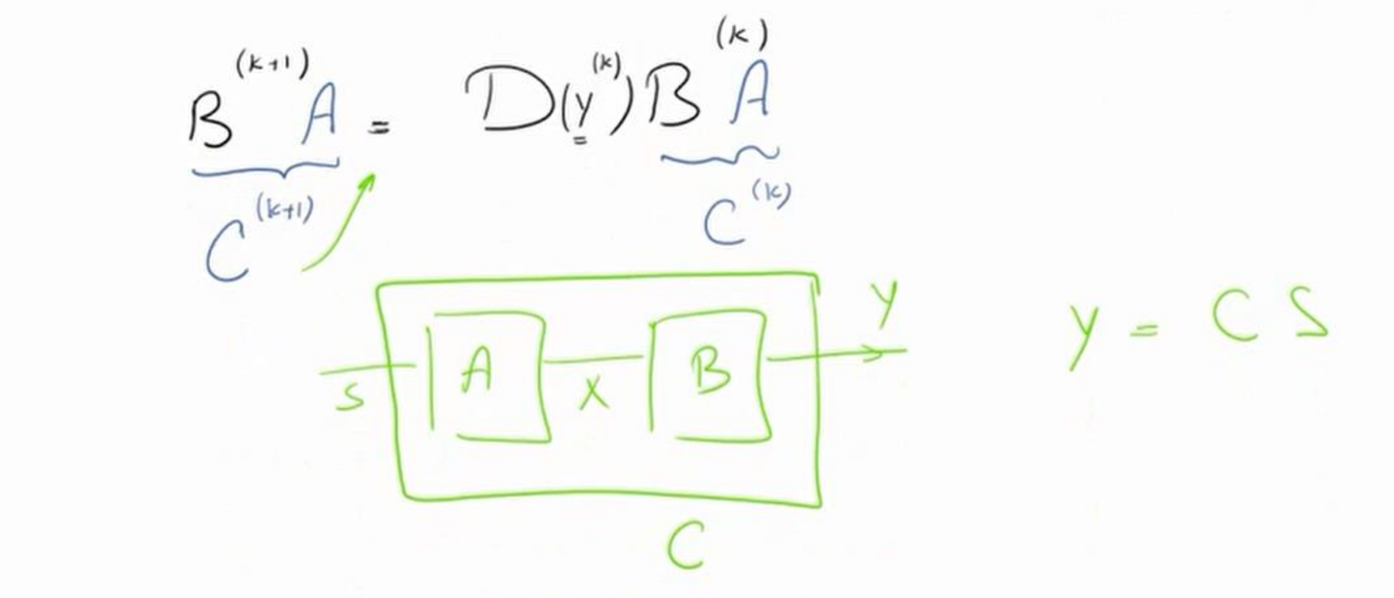
در واقع فرض کنید که چنین ماتریس *Mixture* ایی داشته باشیم:

$$A = \begin{bmatrix} 1 & 0.25 \\ 0.25 & 1 \end{bmatrix} \qquad A = \begin{bmatrix} 1 & 0.99 \\ 0.99 & 1 \end{bmatrix}$$

شکل 3

واضح است که جداسازی منابع با ماتریس سمت چپ راحت تر و بهتر از ماتریس سمت راست است! باید به دنبال روشی بود که این وابستگی را کمتر کند و یا از بین ببرد!

روش پیشنهادی آن است که ضرب B و A را که همان C است بهینه کنیم:

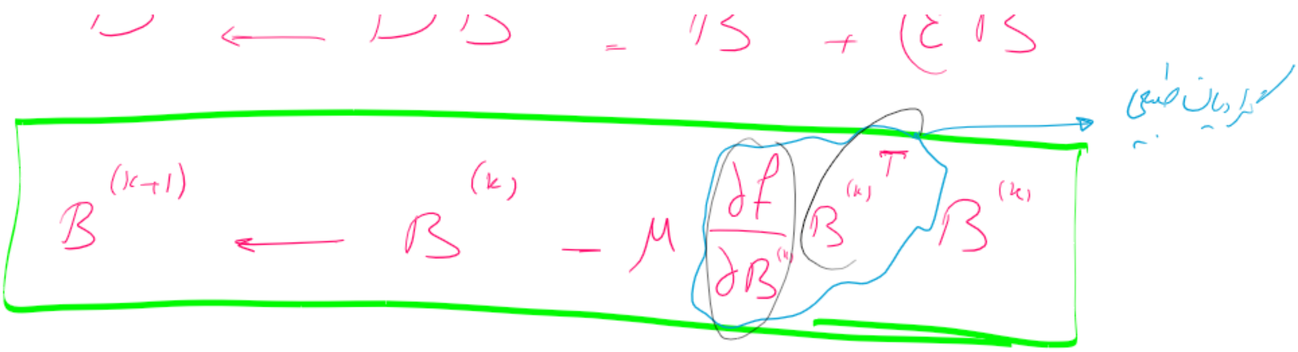


شکل 4

در این روش فرض می‌شود که ماتریس D به فرم $(I+\epsilon)$ است! با پیداکردن مقدار اپسیلون، ماتریس D پیدا می‌شود.

با استفاده از بسط تیلور برای $f(B)$ به این نتیجه می‌رسیم که بهترین مقدار برای اپسیلون به فرم زیر است:

$$\varepsilon = -\mu \frac{\partial f}{\partial B} B^T$$



شکل 5

پس داریم:

$$B^{k+1} = B^k - \mu \frac{\partial}{\partial B^k} f . (B^k)^T B^k$$

به این نوع آپدیت کردن و الگوریتم *Equivariant* می‌گویند!

با استفاده از دانسته‌های قبلی و صرفا تغییر نوع *update* کردن، به یک الگوریتم بهینه‌تر برای حل مسئله می‌رسیم. پس از پیاده‌سازی به صورت زیر داریم:

```
[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
```

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 fact that we have M=N!





mu        = 1e+02;
Max_Iter = 2e+3;
thresh_cntr = 1e-1;
thresh_B = 1e-8;


Error_Iter_deflate_EQ = 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,:))  ; % New Update Rule for
Equivariant!
         B(i,:)         =  B(i,:)/norm(B(i,:));   % Normalization
         B(i,:)         =  ( eye(size(B)) - B(1:i-1,:)'*B(1:i-1,:)   )* B(i,:)';  %
Orthogonality

    end

    [Error_Perm,y_Hat_Chosen,B] = Perm_AMP_Disamb(B,S,Z);
    Error_Iter_deflate_EQ(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_EQ(1,cntr))<thresh_cntr) || (cntr>Max_Iter) ) %|| (
norm(B_prev - B,'fro')<thresh_B  )
        break;
    end
    if ( Error_Iter_deflate_EQ(cntr)<MIN_ERR  )
        y_hat_best = y_Hat_Chosen;
        B_hat_best = B;
        Index_Best = cntr;
        MIN_ERR = Error_Iter_deflate_EQ(cntr);
    end
    cntr = cntr +1;


end


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

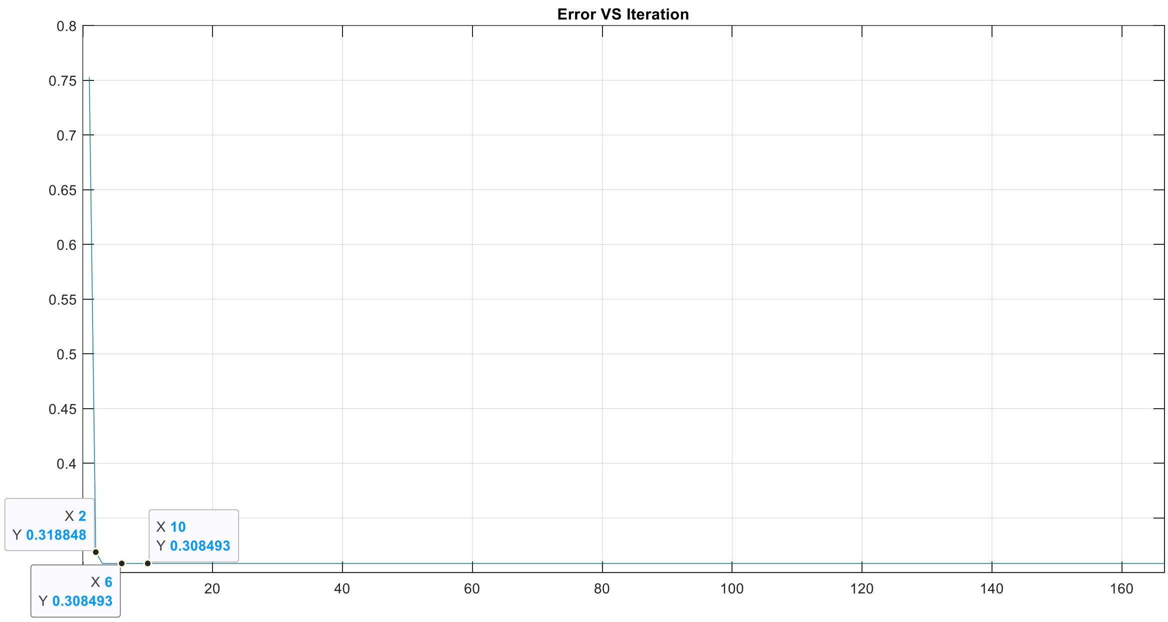
خروجی‌ها به صورت زیر اند:

Results:

```
disp("calculated Error Equals to: "+min(Error_Iter_deflate_EQ))
calculated Error Equals to: 0.30849
disp(abs(B*W*U'*A))
    0.9895    0.0259    0.0689
    0.0204    1.0942    0.2385
    0.0440    0.3701    1.1142
```

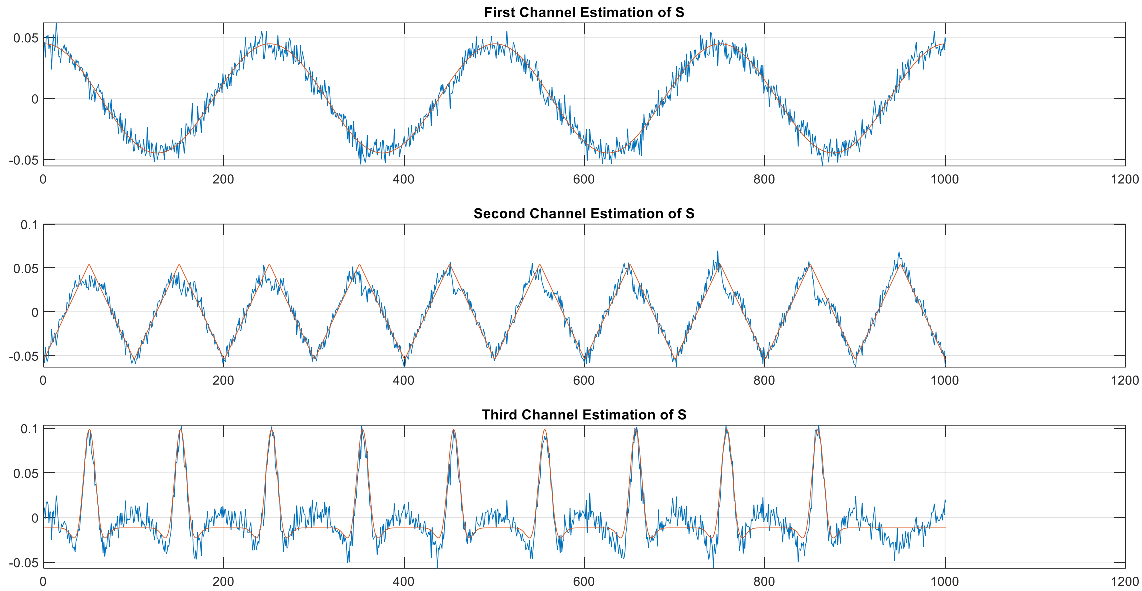
مقادیر فوق، ماتریس ما را به یک ماتریس *permutation* بسیار نزدیک می‌کند!

نودار خطا بر حسب تکرار به صورت زیر است:



6 شکل

همانطور که مشاهده می‌شود، خطای کمتر در تعداد iteration کمتر به دست آمده است.



شکل 7

همگرایی به خوبی صورت گرفته است و شکل موج منبع به درستی بازسازی شده است میزان خطای نهایی به *0.3* رسیده است که در مقایسه با الگوریتم قبلی حدود *0.07* بهتر شده است.

بخش سوم:

۳- به صورت تجربی و مقایسه ای بیان کنید کدام یک از سه روشی که در hw8 و hw9 پیاده کردید (دو روش بالا + روشی که در hw8 پیاده کردید) سریعتر همگرا شد و کدام یک کیفیت جداسازی (پارامتر E) بهتری داشت.

به صورت تجربی، می‌توان با استفاده از نمودارها و خروجی‌ها گفت که الگوریتم *Equivariant* بر روی نحوه‌ی بهینه‌سازی *ICA Deflation* بسیار موثر عمل کرده و موفق بوده است. از بین این 3 راه، مسیر آخر که ترکیب *Deflation* و *Equivariant* بودن است بهینه ترین عملکرد و سریعترین همگرایی را داشته است.

پایان