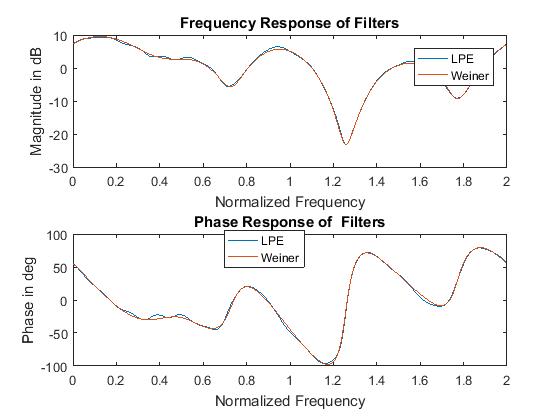
1



The Linear Prediction error filter uses only d(n) and tries to estimate x(n) whereas, inverse system identification problem calculates weiner filter coefficients using input and output. So weiner filter can be more accurate than LPE as it has both input and output. It can be seen from the figure that the filter coefficients match for few higher coefficients and start to deviate in decreasing filter coefficients for Linear Prediction Error Filter and Weiner filter. The frequency response is very similar with very little difference as bigger coefficients are similar.

MATLAB Code:

clear;

%close all

load('V:\EECS-844\Exam-3\P2.mat');

M=19; %Length of each snapshot

K=length(d);

N=K-M+1;

X=complex(zeros(M,N));

D=complex(zeros(M,N));

for k=1:N

X(:,k)=flipud(x(k:k+M-1)); %Snapshot matrix

D(:,k)=flipud(d(k:k+M-1)); %Snapshot matrix

end

%% Using Linear Prediction

R=1/N\*D\*D'; %Correlation Matrix

r=zeros(M,1); %Cross Correlation Matrix

for i=M:K-1

r=r+ flipud(d(i-M+1:i)).\*conj(d(i+1));

end

r=r/N;

wf=R\r; %Weiner filter

am=[1; -wf]; %Linear prediction error filter

%% Inverse system ID using d as input and x as output

M2=M+1;

N2=K-M2+1;

for k=1:N2

D2(:,k)=flipud(d(k:k+M2-1)); %Snapshot matrix

end

R2=1/N2\*D2\*D2'; %Correlation Matrix

P=zeros(M2,1); %Cross Correlation Matrix

for i=M2:N2

P=P+ flipud(d(i-M2+1:i)).\*conj(x(i));

end

P=P/N2;

w=R2\P; %Filter using Weiner Hopf Equation

[H\_lpf,ang\_lpf]=freqz(am,1,512,'whole');

[H\_w,ang\_w]=freqz(w,1,512,'whole');

%% Plotting Filter Properties

figure(1);

subplot(2,1,1); plot([1:M+1],20\*log10(abs(am)));

hold on; plot([1:M+1],20\*log10(abs(w))); legend('LPE ','Weiner')

xlabel('Filter Coefficients');ylabel('Magnitude in dB')

title('Magnitude of Filters')

subplot(2,1,2);plot([1:M+1],angle(am)\*180/pi); title(' Phase of LPE Filter');

hold on;plot([1:M+1],angle(w)\*180/pi,'r'); legend('LPE ','Weiner')

title(' Phase of Filters'); xlabel('Filter Coefficients');ylabel('Phase in deg')

figure(2); subplot(2,1,1); plot(ang\_lpf./pi,20\*log10(abs(H\_lpf)))

hold on; plot(ang\_w./pi,20\*log10(abs(H\_w))); legend('LPE ','Weiner')

legend('LPE','Weiner'); xlabel('Normalized Frequency');ylabel('Magnitude in dB')

title('Frequency Response of Filters')

subplot(2,1,2); plot(ang\_lpf./pi,angle(H\_lpf)\*180/pi);

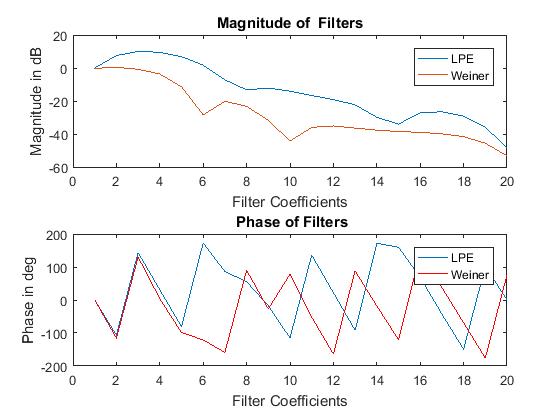
hold on; plot(ang\_w./pi,angle(H\_w)\*180/pi);

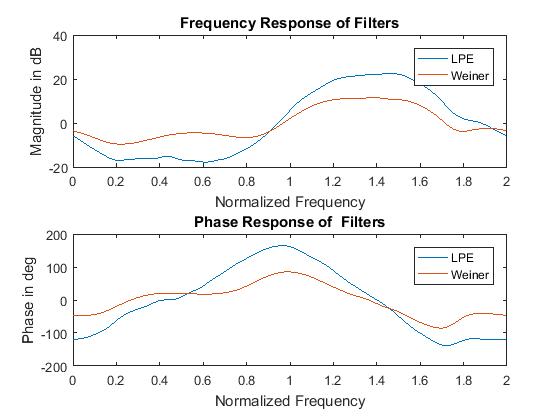
legend('LPE ','Weiner');xlabel('Normalized Frequency');ylabel('Phase in deg')

title('Phase Response of Filters')

y=am'\*D2; %Linear Prediction Filter output

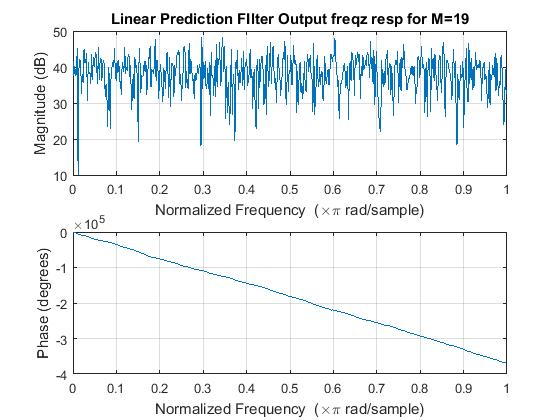
figure;freqz(y);title('Linear Prediction FIlter Output freqz resp for M=599')

2.



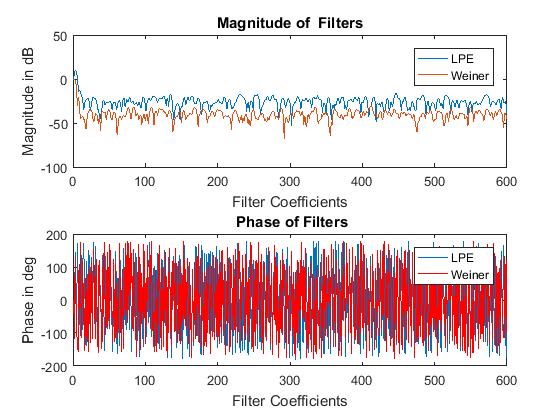
Like in P1.mat, the LPE filter coefficients doesn’t match with weiner filter coefficients. The filter coefficients are different and frequency response is also different. This may be due to the the process not being Auto Regressive or filter order not sufficient enough to remove the correlation between the inputs. Here the sprectral response of the LPE filter output

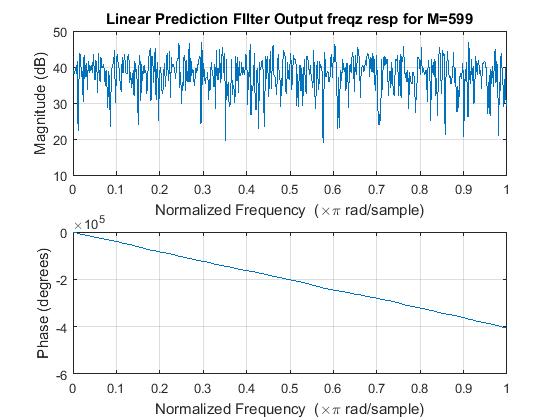
should be spectrally flat if the LPE filter is working properly and the response will not be flat if sufficient filter coefficients aren’t taken. So for M=19 the spectral response of y is:



As we increase the filter length then LPE is able to whiten the spectrum better.

For M=599 the filter and spectral response is shown as:

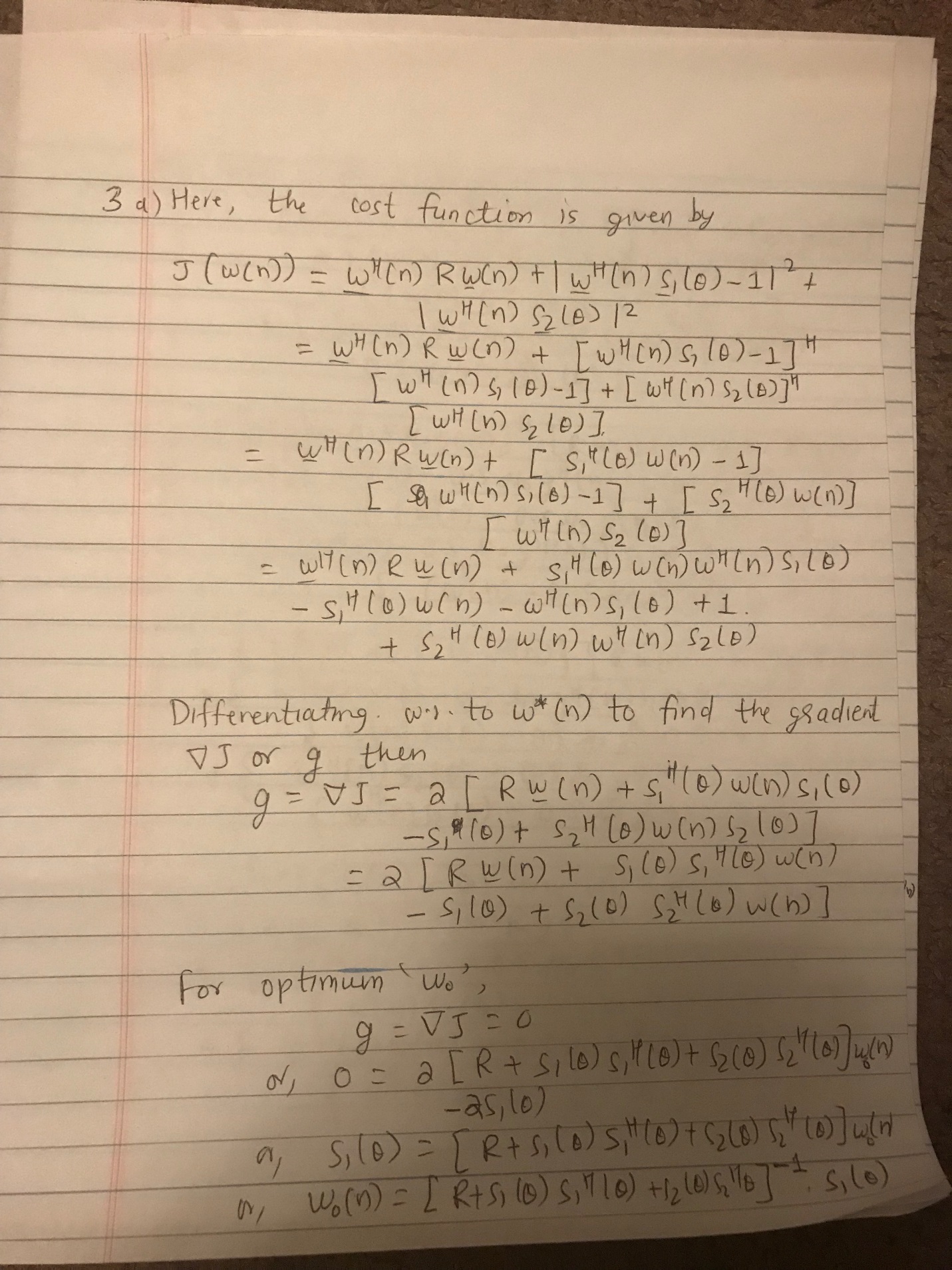


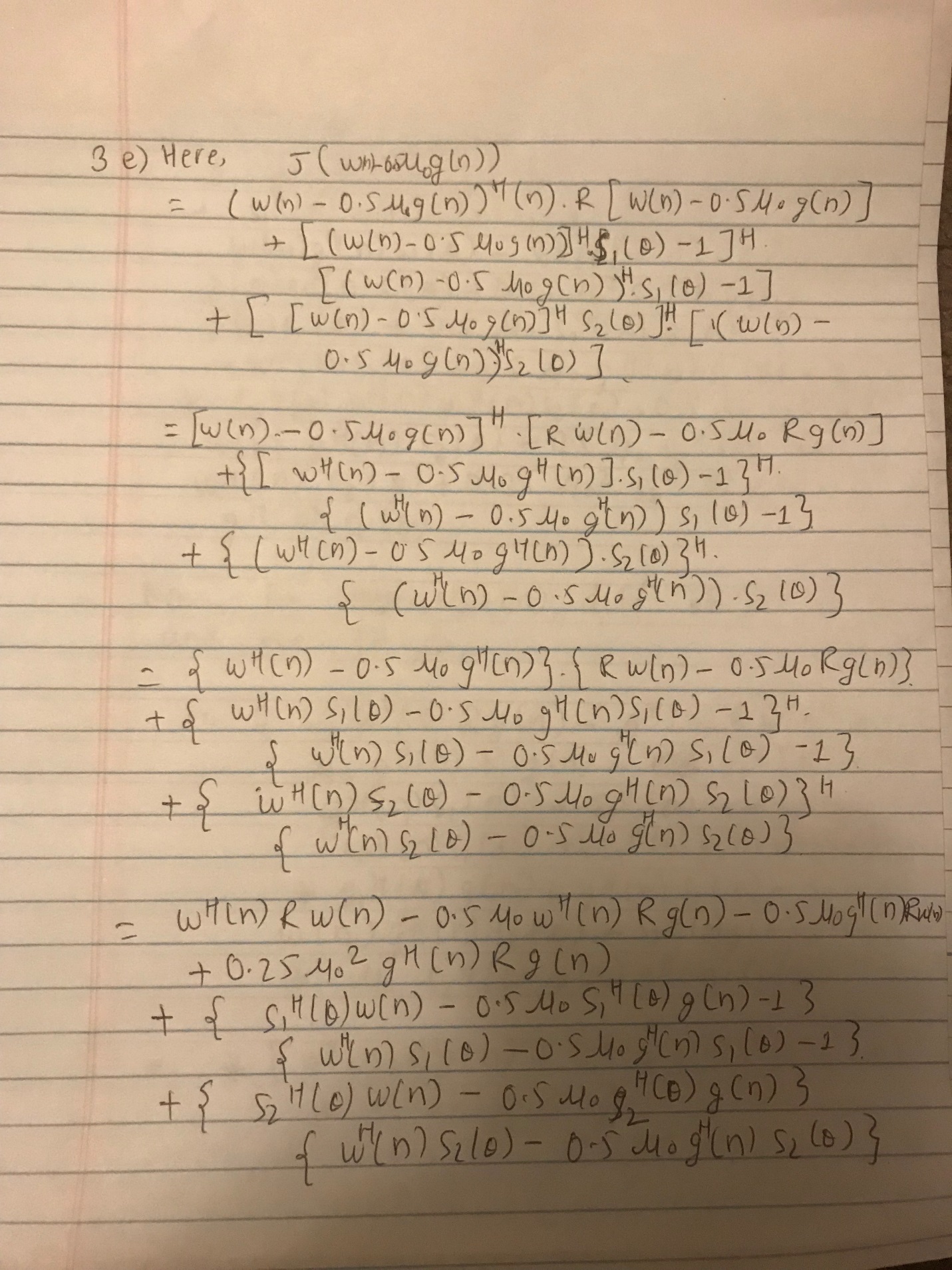


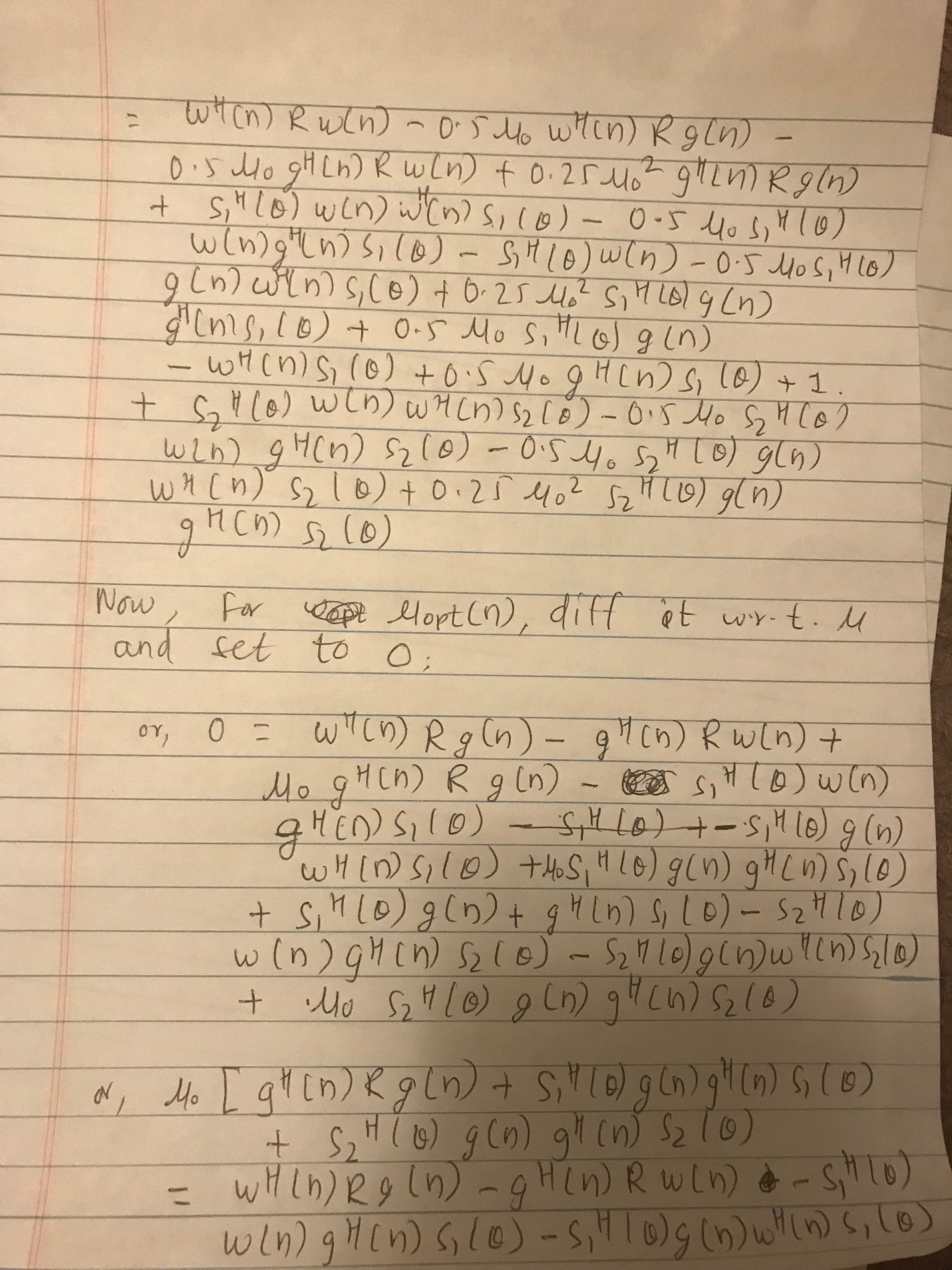
Here the frequency response of y is flatter than previous case hence the sufficient filter order is required for LPE filter to be able to whiten properly.

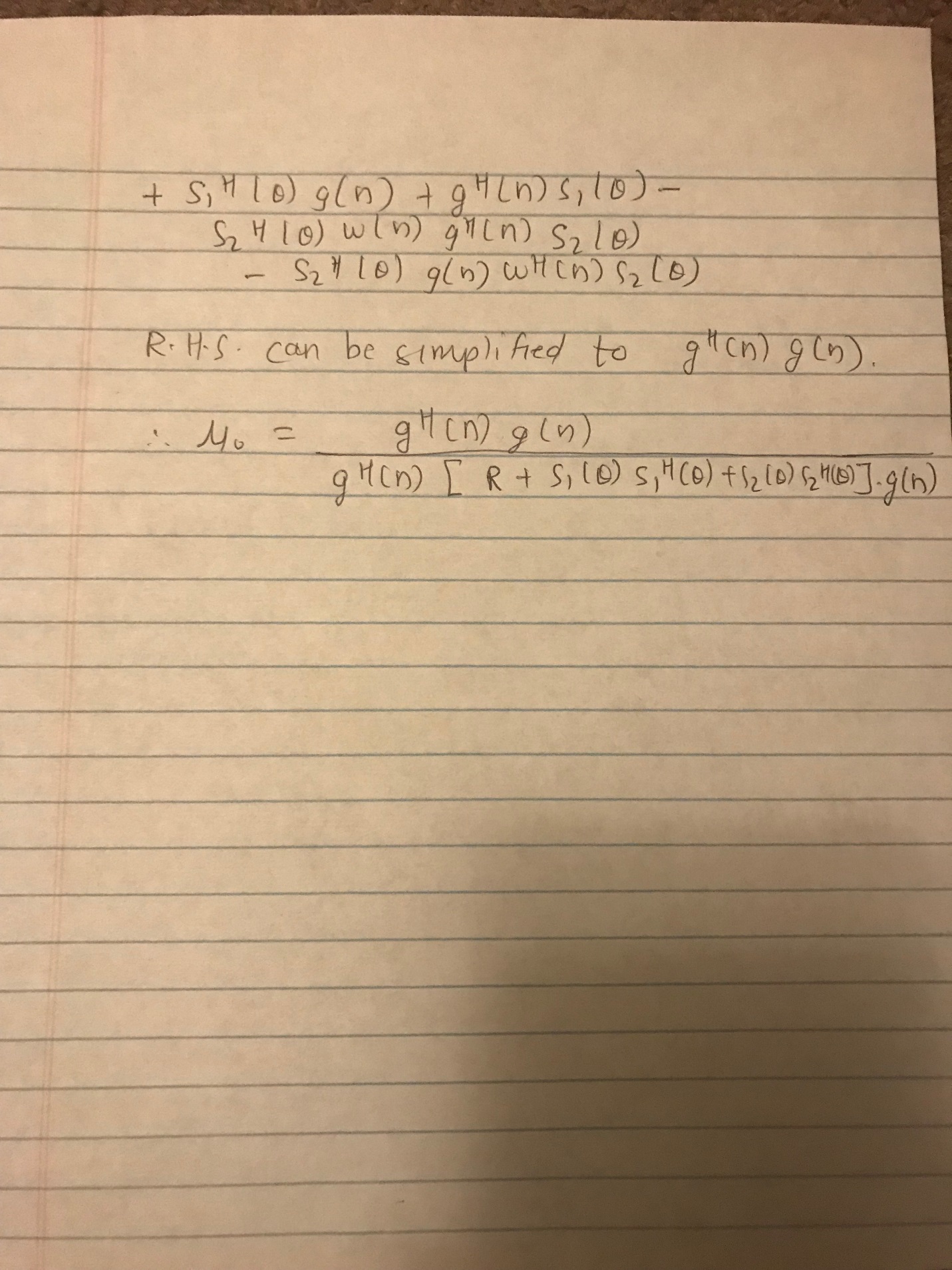
MATLAB Code:

Load P2.mat instead of P1.mat in 1

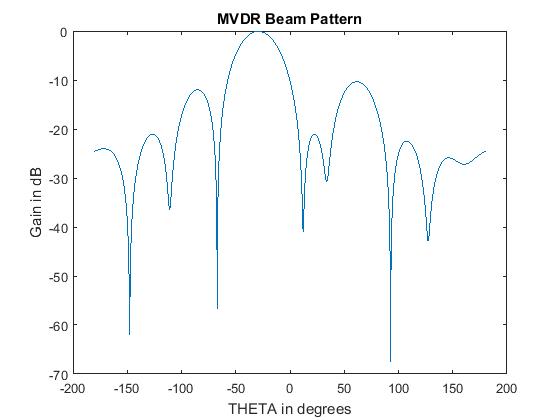
3. Here the cost function is given by 



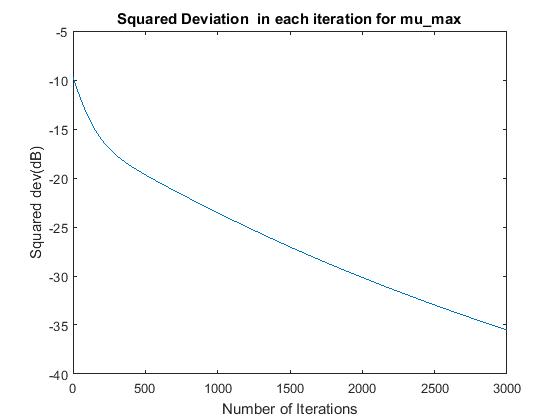




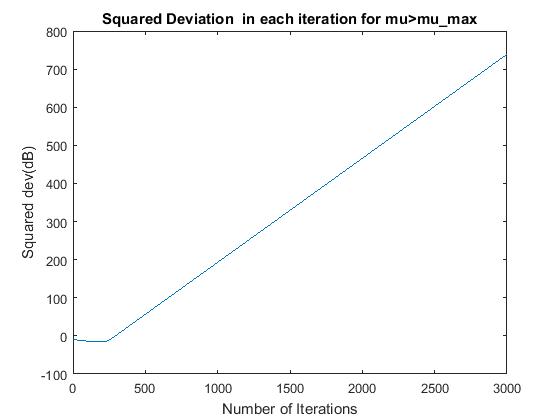
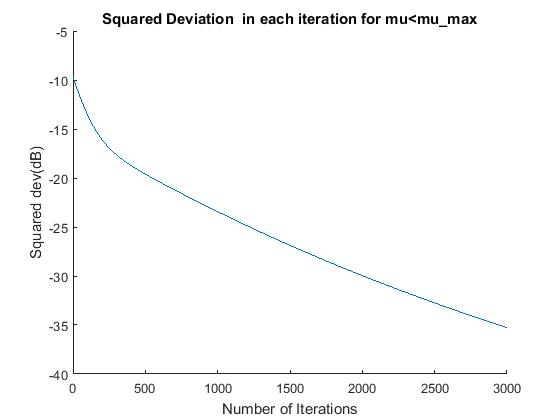
Here the beam pattern for optimum w0 is plotted as



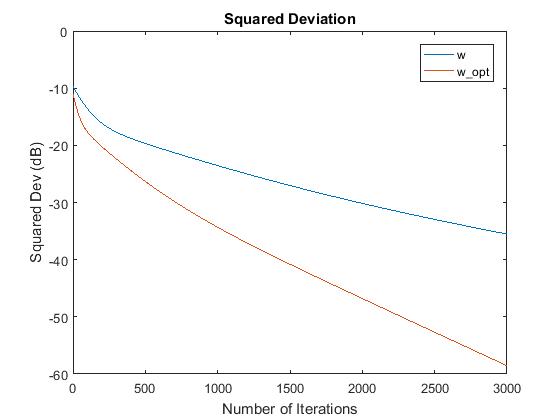
Using steepest descent, the squared deviation for mu\_max = 6.29146\*10^-4 can be plotted as



It can be clearly seen that it is converging. Any value greater than mu\_max makes the square deviation diverge and smaller than mu\_max to converge.



Similarly using optimized mu the steepest decent algorithm gives



Clearly using optimized mu than mu\_max convergence can be achieved faster.

Iterations For -30 dB squared deviation using mu\_max: 1975

Iterations For -30 dB squared deviation using mu\_opt: 713

MATLAB Code:

clear all

load('V:\EECS-844\Exam-3\P3.mat')

close all

N=10; %Number of Antenna Elements

s1=zeros(N,1); %Steering vector 1 at theta=-20

s2=zeros(N,1); %Steering vector 2 at theta=+20

for k=1:N

s1(k)=exp((-1i)\*(k-1)\*(-20\*pi/180));

s2(k)=exp((-1i)\*(k-1)\*(+20\*pi/180));

end

w0=inv(R+s1\*s1'+s2\*s2')\*s1; %Optimum filter

%% Beam Pattern Generation

phi=linspace(-pi/2,pi/2,14000); %phi

theta=pi\*sin(phi); %theta

sv=zeros(N,length(phi));

for i=1:length(phi)

for k=1:N

sv(k,i)=exp(-1i\*(k-1)\*theta(i)); %Steering vectors

end

end

beam\_pattern=sv'\*w0; %Beam Pattern

figure(1);plot(theta\*180/pi,20\*log10(abs(beam\_pattern)));

xlabel('THETA in degrees');ylabel('Gain in dB');

title('MVDR Beam Pattern')

%% Steepest Decent

eigen\_vals=eig(R);

eig\_max=max(eigen\_vals);

mu\_max=2/eig\_max;

mu=mu\_max; %Using mu\_max

g=zeros(N,1);

num\_iter=3000; %Number of Iterations

w=zeros(10,num\_iter+1);

for iter=1:num\_iter

g(:,iter)=2\*(R\*w(:,iter)+s1\*s1'\*w(:,iter)-s1+s2\*s2'\*w(:,iter));

w(:,iter+1)=w(:,iter)-1/2\*mu\*g(:,iter);

J(iter)=cost\_func(R,s1,s2,w(:,iter+1));

Dev(iter)=(w0-w(:,iter+1))'\*(w0-w(:,iter+1));

end

figure(2);plot([1:length(Dev)],10\*log10(Dev));

xlabel('Number of Iterations');ylabel('Squared dev(dB)')

title('Squared Deviation in each iteration for mu\\_max')

figure(5);plot([1:length(Dev)],10\*log10(Dev));

%mu value greater than mu\_max

mu=mu\_max+0.00001;

num\_iter=3000; %Number of Iterations

w=zeros(10,num\_iter+1);

for iter=1:num\_iter

g(:,iter)=2\*(R\*w(:,iter)+s1\*s1'\*w(:,iter)-s1+s2\*s2'\*w(:,iter));

w(:,iter+1)=w(:,iter)-1/2\*mu\*g(:,iter);

J(iter)=cost\_func(R,s1,s2,w(:,iter+1));

Dev2(iter)=(w0-w(:,iter+1))'\*(w0-w(:,iter+1));

end

figure(3);plot([1:length(Dev2)],10\*log10(Dev2));

xlabel('Number of Iterations');ylabel('Squared dev(dB)')

title('Squared Deviation in each iteration for mu>mu\\_max')

%mu value less than mu\_max

mu=mu\_max-0.00001;

num\_iter=3000; %Number of Iterations

w=zeros(10,num\_iter+1);

for iter=1:num\_iter

g(:,iter)=2\*(R\*w(:,iter)+s1\*s1'\*w(:,iter)-s1+s2\*s2'\*w(:,iter));

w(:,iter+1)=w(:,iter)-1/2\*mu\*g(:,iter);

J(iter)=cost\_func(R,s1,s2,w(:,iter+1));

Dev3(iter)=(w0-w(:,iter+1))'\*(w0-w(:,iter+1));

end

figure(4);hold on;plot([1:length(Dev3)],10\*log10(Dev3));

xlabel('Number of Iterations');ylabel('Squared dev(dB)')

title('Squared Deviation in each iteration for mu<mu\\_max')

%% Steepest decent using optimum mu

num\_iter=3000; %Number of Iterations

w2=zeros(10,1);

for iter=1:num\_iter

g2=2\*(R\*w2+s1\*s1'\*w2-s1+s2\*s2'\*w2);

mu\_opt=(g2'\*g2)/(g2'\*(R+s1\*s1'+s2\*s2')\*g2); %Optimum mu

w2=w2-1/2\*mu\_opt\*g2;

J2(iter)=cost\_func(R,s1,s2,w2);

Dev4(iter)=(w0-w2)'\*(w0-w2);

end

figure(5);hold on;plot([1:length(Dev4)],10\*log10(Dev4));

legend('w','w\\_opt');title('Squared Deviation')

xlabel('Number of Iterations');ylabel('Squared Dev (dB)')

dev1\_dB=10\*log10(Dev);

dev4\_dB=10\*log10(Dev4);

idx1=find(dev1\_dB<-30,1,'first');

idx2=find(dev4\_dB<-30,1,'first');

disp(sprintf('Iterations For -30 dB squared deviation using mu\_max: %d',idx1))

disp(sprintf('Iterations For -30 dB squared deviation using mu\_opt: %d',idx2))

function [J]=cost\_func(R,s1,s2,w)

J= w'\*R\*w+abs(w'\*s1-1)^2+abs(w'\*s2)^2;

return

4. Here the response of the matched and mismatch filter are plotted as below.

The mismatch loss for different filters are found as

mismatchedfilter1- Normal ==> 13.1506 dB

mismatchedfilter2- Rows Zero==> 11.1654 dB

mismatchedfilter3- Diagonal Loading==> 4.8352 dB

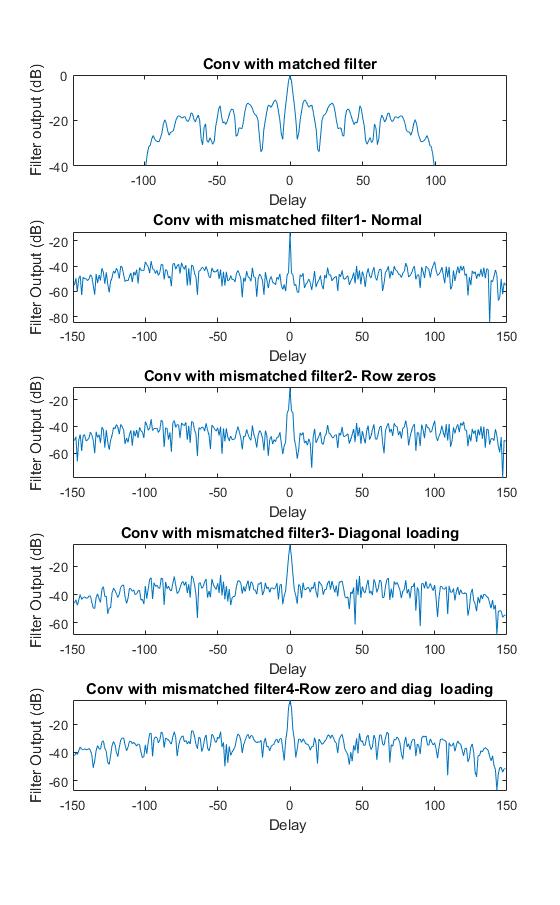
mismatchedfilter4-Row zero and diagonal loading==> 3.0370 dB

Here it can be seen that matched filter doesn’t have good sidelobe suppression which can be achieved by mismatch filter by losing some SNR.

For normal mismatch filter mismatch is very high but it gives better sidelobe suppression. At the cost of resolution, lesser mismatch and sidelobe suppressin can be achieved by setting some rows to zero.

With diagonal loading low mismatch can be obtained but sidelobe suppression decreases.

With both, some rows zero and diagonal loading, mismatch will be very less with increased sidelobe than previous. So using these mismatch filter techniques we can obtain trade off between resolution, sidelobe suppression and mismatch.



MATLAB Code:

load('V:\EECS-844\Exam-3\P4.mat')

M=length(x);

h\_nmf=conj(flipud(x))./(x'\*x); %Normalized matched filter

K=2\*M;

impulse\_pos=1.5\*M;

A=toeplitz([transpose(x) zeros(1,K-1)],[x(1) zeros(1,K-1)]);

%figure(1);imagesc(lp(A));

A2=A;

row\_zero\_idx=[impulse\_pos-2 impulse\_pos-1 impulse\_pos+1 impulse\_pos+2];

A2(row\_zero\_idx,:)=0; % A with some rowsrows set to 0

%figure(2);imagesc(lp(A2));

em=zeros(size(A,1),1);%Elementary vector

em(impulse\_pos)=1;

eig\_vals=eig(A'\*A);

eig\_max=max(eig\_vals);

h\_mmf1=(inv(A'\*A)\*A'\*em); %h\_mmf with no diag loading

h\_mmf2=(inv(A2'\*A2)\*A2'\*em); %h\_mmf with no diag loading and some rows zero

h\_mmf3=(inv(A'\*A+0.01\*eig\_max\*eye(size(A'\*A,1)))\*A'\*em); %h\_mmf with diag loading

h\_mmf4=(inv(A2'\*A2+0.01\*eig\_max\*eye(size(A'\*A,1)))\*A2'\*em); %h\_mmf with diag loading and some rows zero

h\_nmmf1=h\_mmf1/(sqrt(h\_mmf1'\*h\_mmf1)\*sqrt(x'\*x));

h\_nmmf2=h\_mmf2/(sqrt(h\_mmf2'\*h\_mmf2)\*sqrt(x'\*x));

h\_nmmf3=h\_mmf3/(sqrt(h\_mmf3'\*h\_mmf3)\*sqrt(x'\*x));

h\_nmmf4=h\_mmf4/(sqrt(h\_mmf4'\*h\_mmf4)\*sqrt(x'\*x));

conv\_with\_mf=conv(h\_nmf,x); %Convolution with norm matched filter

conv\_with\_mmf1=conv(h\_nmmf1,x); %Convolution with norm mismatched filter1

conv\_with\_mmf2=conv(h\_nmmf2,x); %Convolution with norm mismatched filter2

conv\_with\_mmf3=conv(h\_nmmf3,x); %Convolution with norm mismatched filter3

conv\_with\_mmf4=conv(h\_nmmf4,x); %Convolution with norm mismatched filter4

mism\_loss1=-20\*log10(max(abs(conv\_with\_mmf1))/max(abs(conv\_with\_mf))); %Mismatch loss for mmf1

mism\_loss2=-20\*log10(max(abs(conv\_with\_mmf2))/max(abs(conv\_with\_mf))); %Mismatch loss for mmf1

mism\_loss3=-20\*log10(max(abs(conv\_with\_mmf3))/max(abs(conv\_with\_mf))); %Mismatch loss for mmf1

mism\_loss4=-20\*log10(max(abs(conv\_with\_mmf4))/max(abs(conv\_with\_mf))); %Mismatch loss for mmf1

cmf=transpose(conv\_with\_mf);

conv\_mf=horzcat(zeros(1,50),cmf,zeros(1,50));

l=size(conv\_mf,2)+1;

l2=length(conv\_with\_mmf1)+1;

figure(3)

subplot(5,1,1);plot(-(l/2-1):(l/2-1),20\*log10(abs(conv\_mf)));title('Conv with matched filter')

xlim([-149 149]);ylim([min(20\*log10(abs(conv\_with\_mf))) max(20\*log10(abs(conv\_with\_mf)))])

xlabel('Delay ');ylabel('Filter output (dB)');

subplot(5,1,2);plot((-(l2/2-1):l2/2-1),20\*log10(abs(conv\_with\_mmf1)));title('Conv with mismatched filter1- Normal ')

xlabel('Delay ');ylabel('Filter Output (dB)');ylim([min(20\*log10(abs(conv\_with\_mmf1))) max(20\*log10(abs(conv\_with\_mmf1)))])

subplot(5,1,3);plot((-(l2/2-1):l2/2-1),20\*log10(abs(conv\_with\_mmf2)));title('Conv with mismatched filter2- Row zeros')

xlabel('Delay ');ylabel('Filter Output (dB)');ylim([min(20\*log10(abs(conv\_with\_mmf2))) max(20\*log10(abs(conv\_with\_mmf2)))])

subplot(5,1,4);plot((-(l2/2-1):l2/2-1),20\*log10(abs(conv\_with\_mmf3)));title('Conv with mismatched filter3- Diagonal loading')

xlabel('Delay ');ylabel('Filter Output (dB)');ylim([min(20\*log10(abs(conv\_with\_mmf3))) max(20\*log10(abs(conv\_with\_mmf3)))])

subplot(5,1,5);plot((-(l2/2-1):l2/2-1),20\*log10(abs(conv\_with\_mmf4)));title('Conv with mismatched filter4-Row zero and diag loading')

ylim([min(20\*log10(abs(conv\_with\_mmf4))) max(20\*log10(abs(conv\_with\_mmf4)))])

xlabel('Delay ');ylabel('Filter Output (dB)')

disp('============================')

disp(sprintf('mismatchedfilter1- Normal ==> %2.4f',(mism\_loss1)))

disp(sprintf('mismatchedfilter2- Rows Zero==> %2.4f',(mism\_loss2)))

disp(sprintf('mismatchedfilter3- Diagonal Loading==> %2.4f',(mism\_loss3)))

disp(sprintf('mismatchedfilter4-Row zero and diagonal loading==> %2.4f',(mism\_loss4)))

disp('============================')

%Deconvolution

unknwn\_sys\_mf=conv(y,h\_nmf);

unknwn\_sys\_mmf1=conv(y,h\_nmmf1);

unknwn\_sys\_mmf2=conv(y,h\_nmmf2);

unknwn\_sys\_mmf3=conv(y,h\_nmmf3);

unknwn\_sys\_mmf4=conv(y,h\_nmmf4);

cmf=transpose(unknwn\_sys\_mf);

conv\_mf=horzcat(zeros(1,50),cmf,zeros(1,50));

l=length(unknwn\_sys\_mf)+1;

l2=length(unknwn\_sys\_mmf1)+1;

figure(4)

subplot(5,1,1);plot(-(l/2-1):(l/2-1),20\*log10(abs(unknwn\_sys\_mf)));title('Deconv with matched filter')

xlim([-250 250]);ylim([min(20\*log10(abs(unknwn\_sys\_mf))) max(20\*log10(abs(unknwn\_sys\_mf)))])

xlabel('Delay ');ylabel('Deconvoluted Output (dB)');

subplot(5,1,2);plot((-(l2/2-1):l2/2-1),20\*log10(abs(unknwn\_sys\_mmf1)));title('Deconv with mismatched filter1- Normal ')

xlabel('Delay ');ylabel('Deconvoluted Output (dB)');%ylim([min(20\*log10(abs(unknwn\_sys\_mmf1))) max(20\*log10(abs(conv\_with\_mmf1)))])

subplot(5,1,3);plot((-(l2/2-1):l2/2-1),20\*log10(abs(unknwn\_sys\_mmf2)));title('Deconv with mismatched filter2- Row zeros')

xlabel('Delay ');ylabel('Deconvoluted Output (dB)');%ylim([min(20\*log10(abs(unknwn\_sys\_mmf2))) max(20\*log10(abs(conv\_with\_mmf1)))])

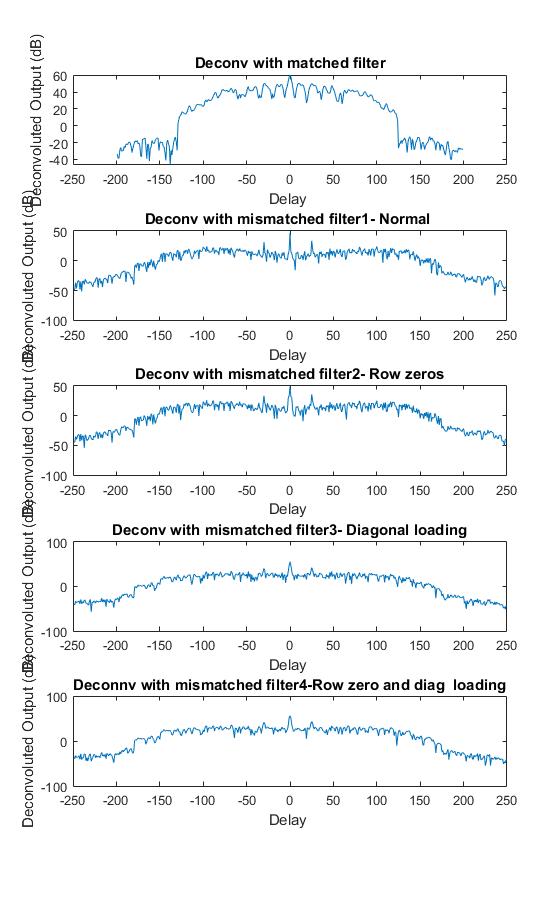
subplot(5,1,4);plot((-(l2/2-1):l2/2-1),20\*log10(abs(unknwn\_sys\_mmf3)));title('Deconv with mismatched filter3- Diagonal loading')

xlabel('Delay ');ylabel('Deconvoluted Output (dB)');%ylim([min(20\*log10(abs(unknwn\_sys\_mmf3))) max(20\*log10(abs(conv\_with\_mmf1)))])

subplot(5,1,5);plot((-(l2/2-1):l2/2-1),20\*log10(abs(unknwn\_sys\_mmf4)));title('Deconnv with mismatched filter4-Row zero and diag loading')

xlabel('Delay ');ylabel('Deconvoluted Output (dB)');%ylim([min(20\*log10(abs(unknwn\_sys\_mmf4))) max(20\*log10(abs(conv\_with\_mmf1)))])

5. The filters obtained were applied to deconvolve the output. The sidelobe suppression is better for mismatched filter than the matched filter but matched filter has high SNR.



For normal mismatch filter sidelobe suppression is high and resolution is better but bigger mismatch.

With settings rows zero, mismatch is reduced i.e. true response is see but at the cost of sidelobe suppression and resolution. By diagonal loading mismatch is further reduced i.e.moving towards true response but it starts to decrease SNR for the one target.

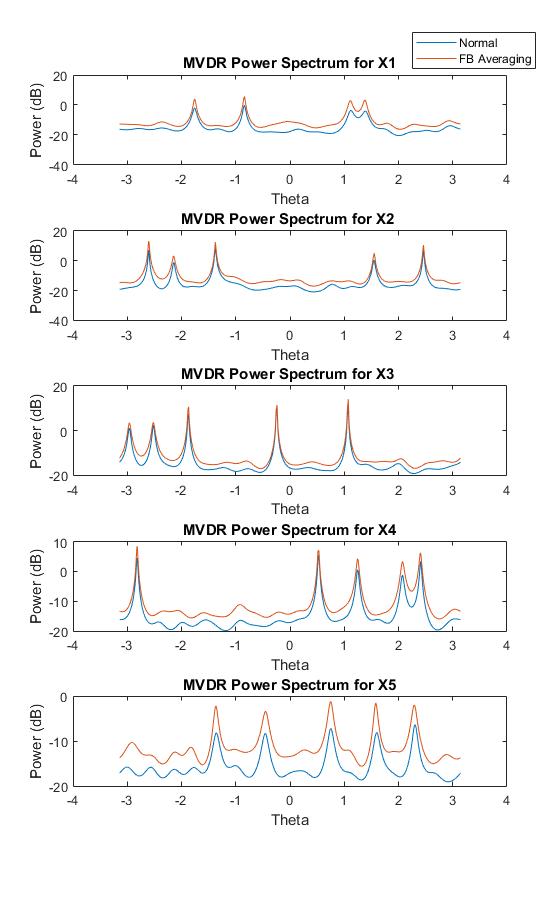
By diagonal loading and rows zero even better match is produced but resolution is less and sidelobe suppression is less.

MATLAB Code:

Derived from 4

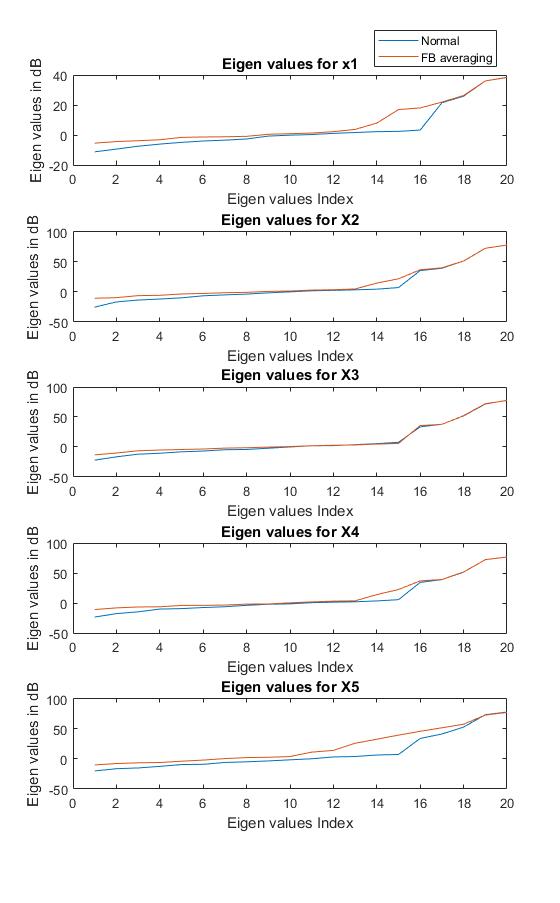
6. The mvdr response using normal correlation matrix and forward backward averaging estimate

of correlation is plotted as:



It can be observed that mvdr from normal estimate of correlation matrix gives better sidelobe suppression whereas, using forward backward averaging better resolution is obtained.

The Eigen values for normal correlation matrix and forward abackward averaging can be plotted as:



Here the higher eigen values match whereas there’s some mismatch in the smaller eigen values. But the condition number of normal correlation matrix is worse than the FB case so FB can be more robust than the normal case.

c. Based on the MVDR response it seems like there are five signals which can be better seen in X5. X1 is the worst since only four signals seem to be visible. For others as well the target seems to be seen at different locations rather than true ones due to calibration error. Hence it can be ranked as

**X1(most severe)> X2>X4>X3>X5(less error)**

MATLAB Code:

load('V:\EECS-844\Exam-3\P6.mat')

close all;

% for X1

[M,~]=size(X1);

[R\_normal1, R\_fb1]=corr\_matrix(X1);

eig\_normal1=sort(eig(R\_normal1));

eig\_fb1=sort(eig(R\_fb1));

[mvdr\_beam\_pattern\_normal1]=Mvdr\_power\_spectrum(M,R\_normal1);

[mvdr\_beam\_pattern\_fb1]=Mvdr\_power\_spectrum(M,R\_fb1);

phi=linspace(-pi/2,pi/2,600);

theta=pi\*sin(phi);

figure(1);

subplot(5,1,1); plot(theta,10\*log10(abs(mvdr\_beam\_pattern\_normal1)));title('MVDR Power Spectrum for X1')

hold on; plot(theta,10\*log10(abs(mvdr\_beam\_pattern\_fb1)));xlabel('Theta');ylabel('Power (dB)')

legend('Normal','FB Averaging')

figure(2);subplot(5,1,1);plot([1:M],10\*log10(abs(eig\_normal1))); title('Eigen values for x1')

hold on; plot([1:M],10\*log10(abs(eig\_fb1))); ylabel('Eigen values in dB');xlabel('Eigen values Index')

legend('Normal','FB averaging')

% for X2

[M,~]=size(X2);

[R\_normal2, R\_fb2]=corr\_matrix(X2);

eig\_normal2=sort(eig(R\_normal2));

eig\_fb2=sort(eig(R\_fb2));

[mvdr\_beam\_pattern\_normal2]=Mvdr\_power\_spectrum(M,R\_normal2);

[mvdr\_beam\_pattern\_fb2]=Mvdr\_power\_spectrum(M,R\_fb2);

figure(1);

subplot(5,1,2); plot(theta,10\*log10(abs(mvdr\_beam\_pattern\_normal2)));title('MVDR Power Spectrum for X2 ')

hold on; plot(theta,10\*log10(abs(mvdr\_beam\_pattern\_fb2)));xlabel('Theta');ylabel('Power (dB)')

legend('Normal','FB Averaging')

figure(2);subplot(5,1,2);plot([1:M],20\*log10(abs(eig\_normal2))); title('Eigen values for X2')

hold on;plot([1:M],20\*log10(abs(eig\_fb2))); ylabel('Eigen values in dB');xlabel('Eigen values Index')

% for X3

[M,~]=size(X3);

[R\_normal3, R\_fb3]=corr\_matrix(X3);

eig\_normal3=sort(eig(R\_normal3));

eig\_fb3=sort(eig(R\_fb3));

[mvdr\_beam\_pattern\_normal3]=Mvdr\_power\_spectrum(M,R\_normal3);

[mvdr\_beam\_pattern\_fb3]=Mvdr\_power\_spectrum(M,R\_fb3);

figure(1);

subplot(5,1,3); plot(theta,10\*log10(abs(mvdr\_beam\_pattern\_normal3)));title('MVDR Power Spectrum for X3 ')

hold on; plot(theta,10\*log10(abs(mvdr\_beam\_pattern\_fb3)));xlabel('Theta');ylabel('Power (dB)')

legend('Normal','FB Averaging')

figure(2);subplot(5,1,3);plot([1:M],20\*log10(abs(eig\_normal3))); title('Eigen values for X3')

hold on;plot([1:M],20\*log10(abs(eig\_fb3))); ylabel('Eigen values in dB');xlabel('Eigen values Index')

% for X4

[M,~]=size(X4);

[R\_normal4, R\_fb4]=corr\_matrix(X4);

eig\_normal4=sort(eig(R\_normal4));

eig\_fb4=sort(eig(R\_fb4));

[mvdr\_beam\_pattern\_normal4]=Mvdr\_power\_spectrum(M,R\_normal4);

[mvdr\_beam\_pattern\_fb4]=Mvdr\_power\_spectrum(M,R\_fb4);

figure(1);

subplot(5,1,4); plot(theta,10\*log10(abs(mvdr\_beam\_pattern\_normal4)));title('MVDR Power Spectrum for X4')

hold on; plot(theta,10\*log10(abs(mvdr\_beam\_pattern\_fb4)));xlabel('Theta');ylabel('Power (dB)')

legend('Normal','FB Averaging')

figure(2);subplot(5,1,4);plot([1:M],20\*log10(abs(eig\_normal4))); title('Eigen values for X4')

hold on;plot([1:M],20\*log10(abs(eig\_fb4))); ylabel('Eigen values in dB');xlabel('Eigen values Index')

% for X5

[M,~]=size(X5);

[R\_normal5, R\_fb5]=corr\_matrix(X5);

eig\_normal5=sort(eig(R\_normal5));

eig\_fb5=sort(eig(R\_fb5));

[mvdr\_beam\_pattern\_normal5]=Mvdr\_power\_spectrum(M,R\_normal5);

[mvdr\_beam\_pattern\_fb5]=Mvdr\_power\_spectrum(M,R\_fb5);

figure(1);

subplot(5,1,5); plot(theta,10\*log10(abs(mvdr\_beam\_pattern\_normal5)));title('MVDR Power Spectrum for X5')

hold on; plot(theta,10\*log10(abs(mvdr\_beam\_pattern\_fb5)));xlabel('Theta');ylabel('Power (dB)')

legend('Normal','FB Averaging')

figure(2);subplot(5,1,5);plot([1:M],20\*log10(abs(eig\_normal5))); title('Eigen values for X5')

hold on;plot([1:M],20\*log10(abs(eig\_fb5))); ylabel('Eigen values in dB');xlabel('Eigen values Index')

function [R\_normal, R\_fb]=corr\_matrix(X)

[M,L]=size(X);

R\_normal=1/L\*X\*X'; %Normal Correlation Matrix

J=flipud(eye(M));

R\_fb=1/(2\*L)\*(X\*X'+J\*conj(X)\*transpose(X)\*J); %Correlation Matrix using Forward Backward Averaging

return

function [mvdr\_beam\_pattern]=Mvdr\_power\_spectrum(M,R)

phi=linspace(-pi/2,pi/2,600);

theta=pi\*sin(phi);

for j=1:length(theta)

for k=1:M

sv(k,j)=transpose(exp((-1i)\*theta(j)\*(k-1)));

end

mvdr\_beam\_pattern(j)=1/((sv(:,j)'\*inv(R)\*sv(:,j)));

end

mvdr\_beam\_pattern\_dB=10\*log10( mvdr\_beam\_pattern);

end