EECS 844

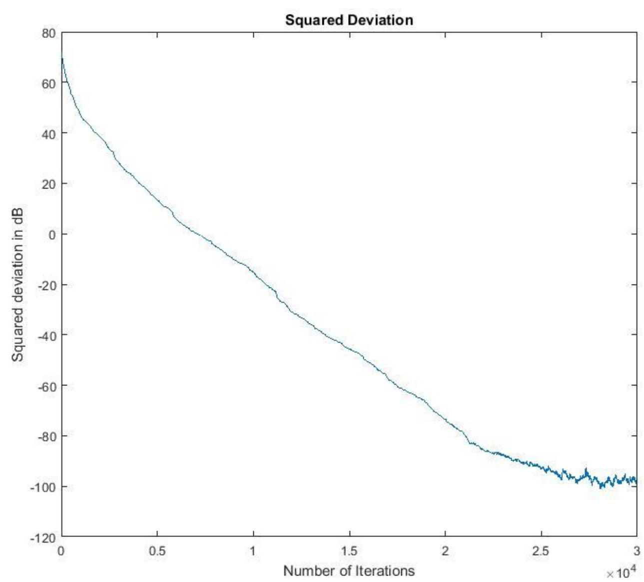
Homework 4

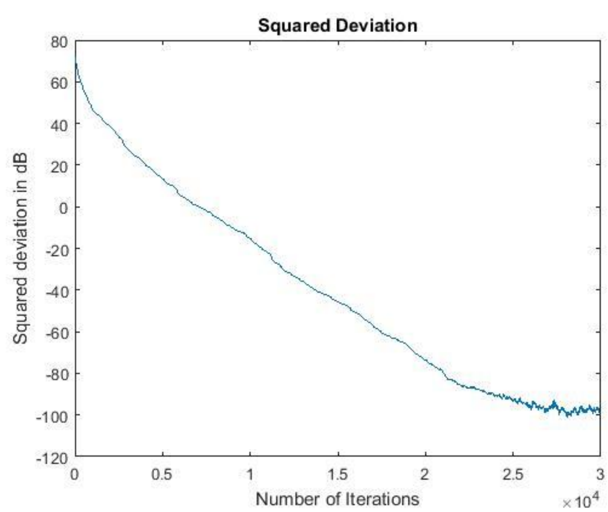
Q1. For P1.mat dataset, NLMS algorithm was applied and the resultant plot can be seen in figure 1.

With the increase in number of iterations, it is converging but the rate of convergence is slow. For NLMS convergence speed is dependant on the eigen value spread. Finally near the convergence, there is oscillations or misadjustments near weiner filter solution which is dependant on the step size.

Similarly the error seems to be fluctuating in it's path towards convergence or minimum error and keeps on fluctuating near the valley which is also dictated by the step size.

Here if stepsize is big, convergence would be fast but misadjustments would be more and vice versa.



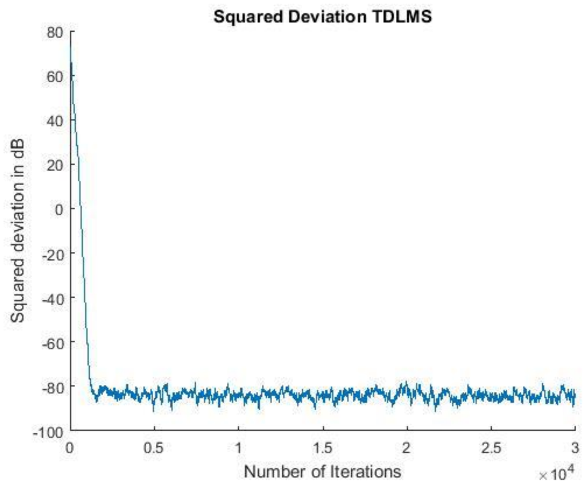


MATLAB Code

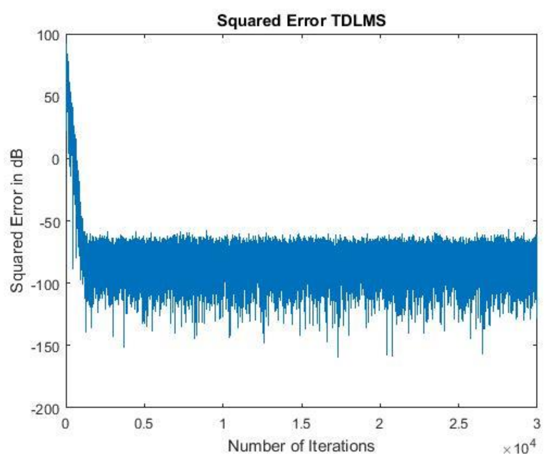
```
clear all;
close all
load('V:\EECS-844\Exam-4\P1.mat');

M=60;          %Length of each snapshot
K=length(x);   %Length of data
N=K-M+1;       %Number of Snapshots
X=complex(zeros(M,N));

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

R=1/N*(X*X');        %Correlation Matrix

P=zeros(M,1);      %Cross Correlation Matrix
for i=M:K
  P=P+ flipud(x(i-M+1:i)).*conj(d(i));
end
P=P/N;
w_opt=inv(R)*P;       %Optimum Weiner filter

%% NLMS

mu=0.5;             %Step Size
del=0.02;           %Leakage Factor

w=zeros(M,1);
dev=zeros(K,1);
squared_error=zeros(K,1);
for n=M:K
  error=d(n)-w'*flipud(x(n-M+1:n));     %Error
  mu_normalized=mu/(del+ctranspose(x(n-M+1:n))*x(n-M+1:n));
  w=w+mu_normalized*conj(error)*flipud(x(n-M+1:n));
  dev(n)=(w-w_opt)'*(w-w_opt);
  squared_error(n)=abs(error)^2;
end

 figure(1);plot(20*log10(dev(M:K)));title('Squared Deviation')
 xlabel('Number of Iterations');
 ylabel('Squared deviation in dB')
 figure(2);plot(20*log10(squared_error(M:K)));title('Squared Error')
  xlabel('Number of Iterations');
 ylabel('Squared Error in dB')
```

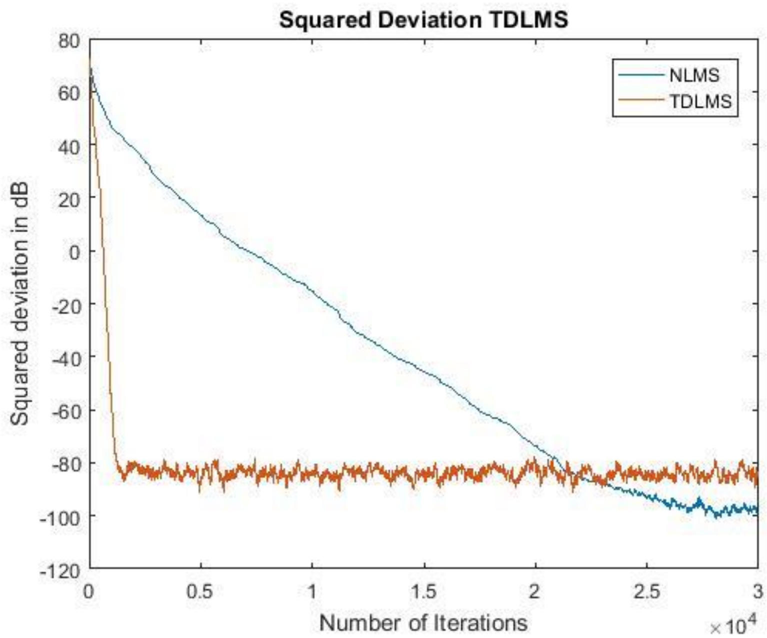
Q2. Here using P1.mat Transform domain LMS was implemented. KLT was applied to input and used for TDLMS algorithm. The squared error and squared deviation is plotted in figure 2.

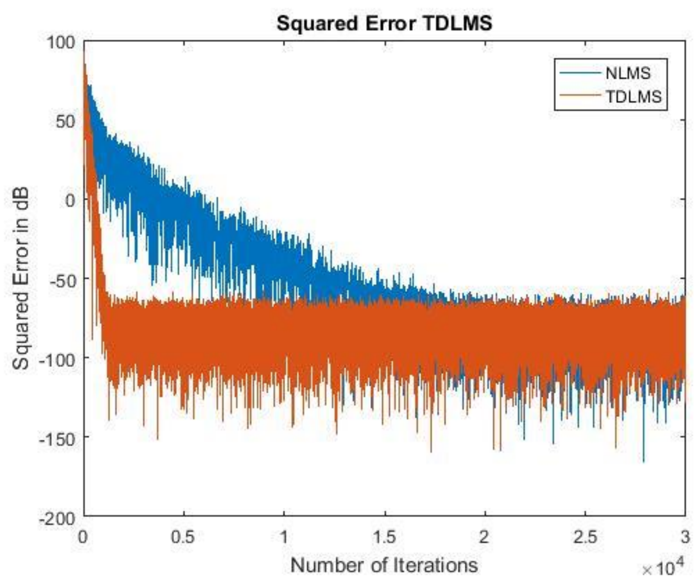
Compared to NLMS the convergence speed is faster but at the cost of more error than NLMS. Because TDLMS is operated in transform domain where the signal is decorrelated, convergence is faster than



NLMS.




Squared Deviation TDLMS
NLMS
TDLMS
Squared deviation in dB
Number of Iterations
$\times 10^4$
80
60
40
20
0
-20
-40
-60
-80
-100
-120
0
0.5
1
1.5
2
2.5
3


Squared Error TDLMS
NLMS
TDLMS
Squared Error in dB
Number of Iterations
$\times 10^4$
100
50
0
-50
-100
-150
-200
0
0.5
1
1.5
2
2.5
3

MATLAB Code

```
clear;
load('V:\EECS-844\Exam-4\P1.mat');
M=60;      %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

R=1/N*(X*X');        %Correlation Matrix

P=zeros(M,1);      %Cross Correlation Matrix
for i=M:K
  P=P+ flipud(x(i-M+1:i)).*conj(d(i));
end
P=P/N;
w_opt=R\P;       % Optimum Weiner filter

%% TDLMS
mu=0.5/M;
del=0.02;

[Q,D]=eig(R);
eig_vals=eig(R);
D=D+del*eye(size(D));
Z=Q'*X;              %Transformed input
w_optT=Q'*w_opt;     %Transformed Optimum weight vector

w_norm=zeros(M,1);
w_td=zeros(M,1);
for n=M:K
  e=d(n)-(w_norm'*flipud(x(n-M+1:n)));     %Error in normal domain
  z=Q'*flipud(x(n-M+1:n));
  w_td=w_td+mu*inv(D)*z*conj(e);
  w_norm=Q*w_td;          %Normal domain filter coeff
  squared_error(n)=(abs(e).^2);    %Error in normal domain
  dev(n)=(w_norm-w_opt)'*(w_norm-w_opt);    %Error in normal domain
end
figure(1);hold on;plot(20*log10(dev(M:K)))
title('Squared Deviation TDLMS');
xlabel('Number of Iterations');
ylabel('Squared deviation in dB')
figure(2);hold on; plot(20*log10(squared_error(M:K)));
title('Squared Error TDLMS')
xlabel('Number of Iterations');
ylabel('Squared Error in dB')
```

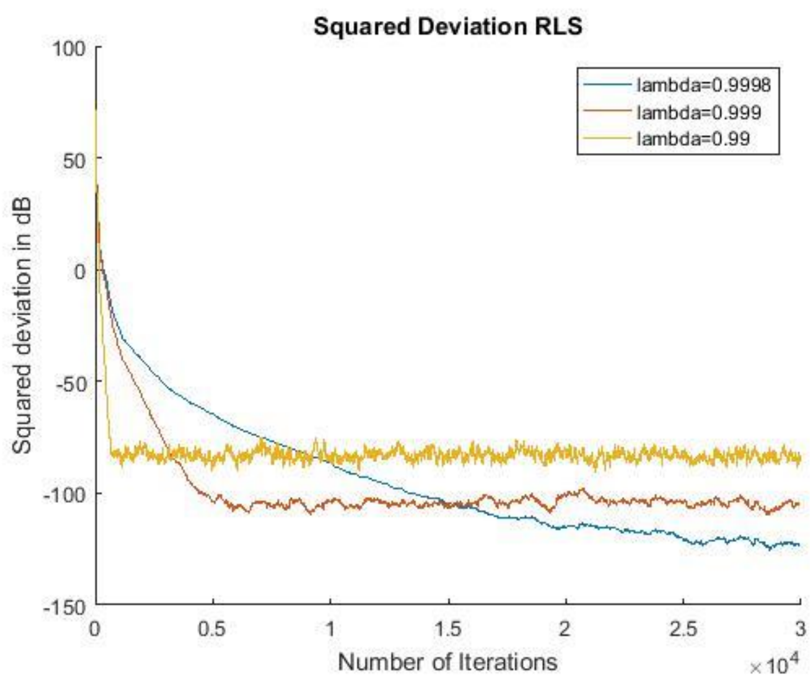
Q3. For P1.mat RLS was implemented and the squared error and squared deviation was plotted. Here RLS with lowest forgetting factor (lambda=0.99) has the fastest convergence speed and highest forgetting factor(lambda=0.9998) has the slowest convergence. But the final squared deviation is maximum for lowest forgetting factor (lambda=0.99) and minimum for highest forgetting factor (lambda=0.9998). Since lowest forgetting factor takes more past inputs into account it reaches to convergence faster.

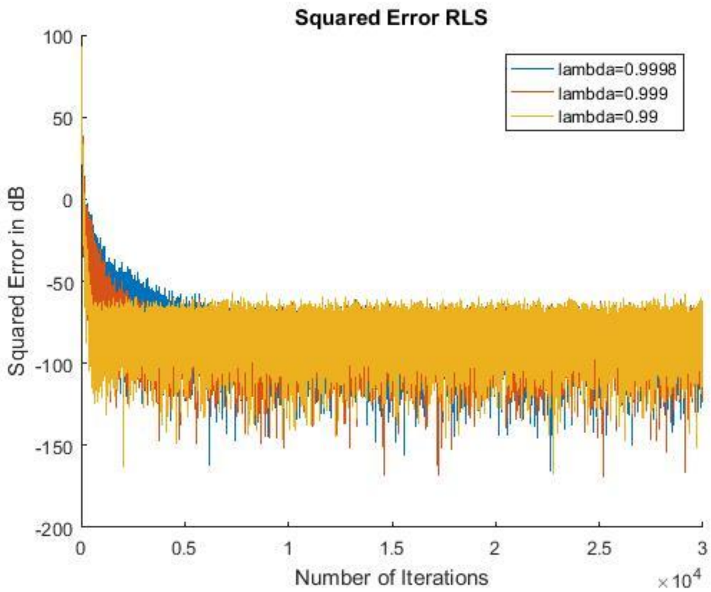
Compared to NLMS convergence is very fast.

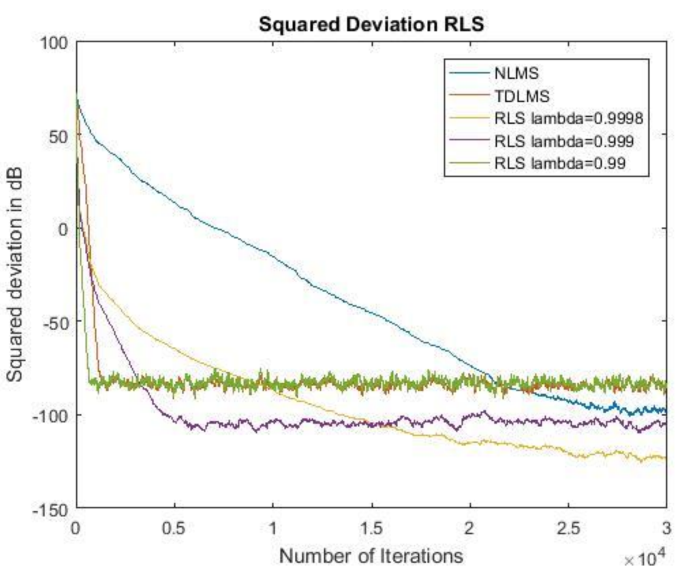
CONVERGENCE (Iterations): NLMS(~26880) vs RLS (~740, lambda=0.99) vs TDLMS(~1566)

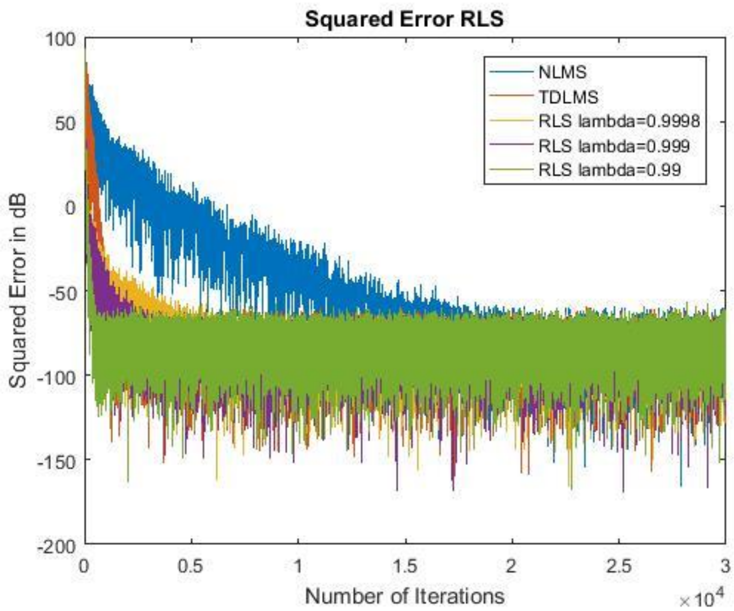
Squared deviation is less for RLS (lambda=0.9998) followed by RLS(lambda=0.999) , NLMS, TDLMS, RLS(lambda=0.99).

For squared error after convergence, error remains almost comparable for all algorithms.




Squared Error RLS
lambda=0.9998
lambda=0.999
lambda=0.99
Squared Error in dB
Number of Iterations
100
50
0
-50
-100
-150
-200
0
0.5
1
1.5
2
2.5
3
×10^4


Squared Deviation RLS
NLMS
TDLMS
RLS lambda=0.9998
RLS lambda=0.999
RLS lambda=0.99
Squared deviation in dB
Number of Iterations
100
50
0
-50
-100
-150
0
0.5
1
1.5
2
2.5
3
×10^4



MATLAB Code:

```
clear;
%close all
load('V:\EECS-844\Exam-4\P1.mat');

M=60;      %Length of each snapshot
K=length(x);
N=K-M+1;

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

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

R=1/N*X*X';        %Correlation Matrix

P=zeros(M,1);      %Cross Correlation Matrix
for i=M:K
  P=P+ flipud(x(i-M+1:i)).*conj(d(i));
end
P=P/N;
w_opt=inv(R)*P;       %Optimum Weiner filter

%% RLS

lambda_vect=[0.9998 0.999 0.99];

for lidx=1:length(lambda_vect)
  lambda=lambda_vect(lidx);
  P=eye(M);
w=zeros(M,1);
for n=M:K
```

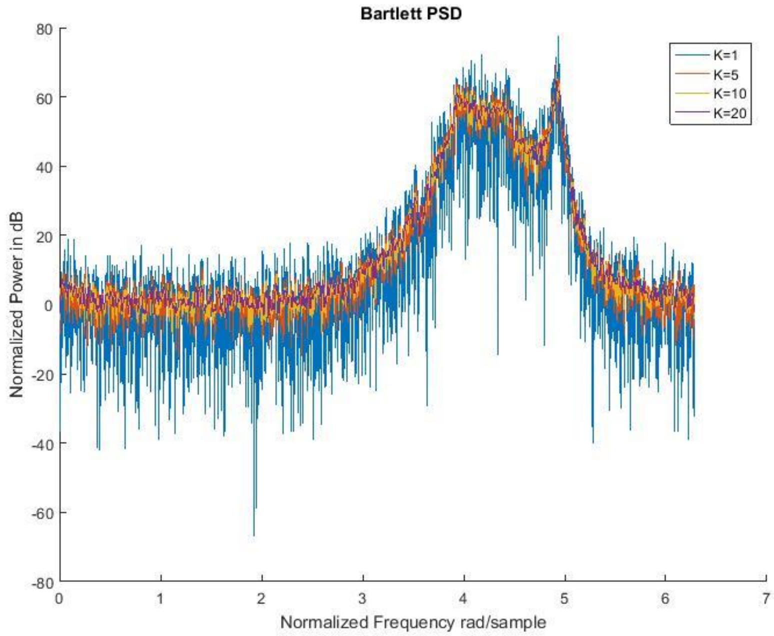
```
  PI=P*(flipud(x(n-M+1:n)));
  k=PI/(lambda+(flipud(x(n-M+1:n))'*PI));
  error=d(n)-w'*(flipud(x(n-M+1:n)));
  w=w+k*conj(error);
  P=P/lambda-1/lambda*(k*(flipud(x(n-M+1:n))'*P));
  squared_error(n)=(abs(error)^2);
  dev(n)=(w-w_opt)'*(w-w_opt);
end

figure(1);hold on;plot(20*log10(dev(M:K)));
figure(2);hold on;plot(20*log10(squared_error(M:K)));
end
figure(1);title('Squared Deviation RLS');
 xlabel('Number of Iterations');
 ylabel('Squared deviation in dB')
figure(2);title('Squared Error RLS');
 xlabel('Number of Iterations');
 ylabel('Squared Error in dB')

figure(1);legend('lambda=0.9998','lambda=0.999','lambda=0.99')
figure(2);legend('lambda=0.9998','lambda=0.999','lambda=0.99')
```

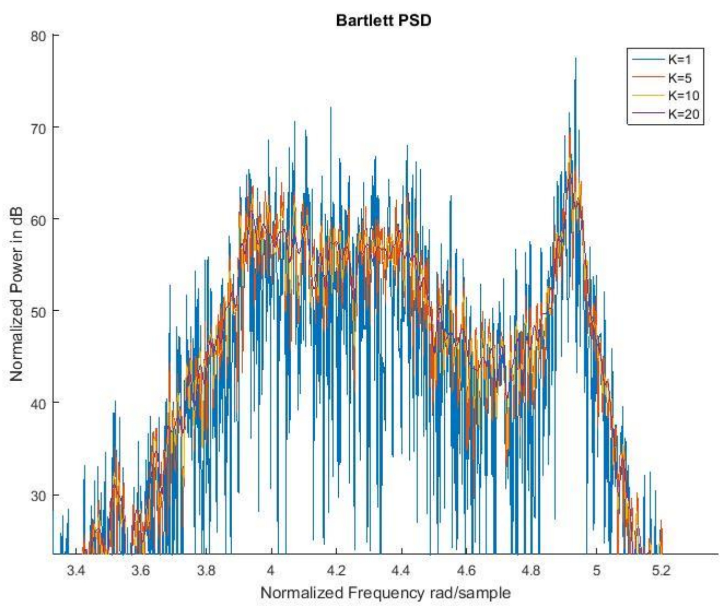
Q 4. Here for P4.mat, Bartlett PSD estimation was implemented for different segment lengths.

For K=1, it can be seen that the resolution is very high but the spectrum variance is the largest.

As we go on increasing the number of segments 'K', the number of samples in a segment decrease which degrades the resolution but it decreases the variance as averaging is done.

For K=20 the variance is least but the resolution is the worst. Also with averaging the peaks become more smooth and distinct at 3.9 rad , 4.3 rad and 4.9 rad.


Bartlett PSD
K=1
K=5
K=10
K=20
Normalized Power in dB
Normalized Frequency rad/sample
80
60
40
20
0
-20
-40
-60
-80
0
1
2
3
4
5
6
7


Bartlett PSD
K=1
K=5
K=10
K=20
Normalized Power in dB
Normalized Frequency rad/sample
80
70
60
50
40
30
3.4
3.6
3.8
4
4.2
4.4
4.6
4.8
5
5.2

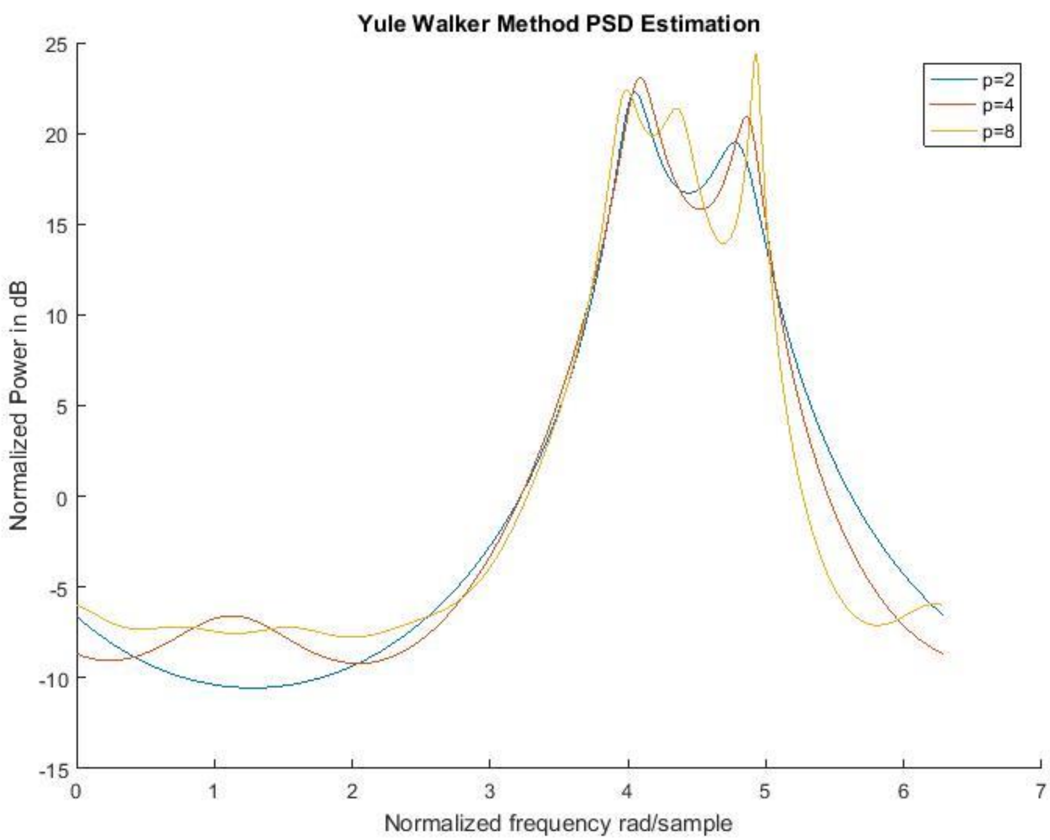
MATLAB Code:

```
%Barttlett window
clear all;
close all;
load('V:\EECS-844\Exam-4\P4.mat');
K=[1 5 10 20];

for iter=1:4
  M=length(x)/K(iter);  %Numner of samples for PSD calc
  Xi=reshape(x,M,K(iter)); %Division into chunks

  for div=1:size(Xi,2)
    Xii=Xi(:,div);          %Samples for PSD calculation
    %  f=-1/2:0.001:1/2;
    f=linspace(0,1,size(x,1)/2);        %Frequency
    for idx=1:length(f)
      temp=0;
      for n=0:M-1
        temp=temp+Xii(n+1)*exp((-1i)*2*pi*f(idx)*n);
      end
      P(idx)=(abs(temp)^2)/length(Xii);  %Power SPectrum for each chunk
    end
    P_all(:,div)=P;     %Power Spectrum for all chunks
  end
  P_final=1/K(iter)*sum(P_all,2);  %Normalizing

  w=2*pi*f;
  figure(1);hold on;plot(w,20*log10(abs(P_final)))
end
figure(1); title('Bartlett PSD ')
xlabel('Normalized Frequency rad/sample')
ylabel('Normalized Power in dB');
legend('K=1','K=5','K=10','K=20')
```

Q5. For P4.mat PSD was calculated by first calculating AR model coefficients and seeing the frequency response for the filter coefficients which is a parametric method. Here SCM was calculated by using snapshot matrix.

The data was modelled using different AR models of order 2, 4 and 8. AR model of 8 showed better frequency resolution as clearly 3 distinct peaks can be visible or differentiated. For p=2 it can only reveal two signals and p=4 has better resolution than p=4.

Compared to Bartlett method, the frequency response is smooth and peaks are distinct. So for properly modelled AR model, it has better PSD estimation than Bartlett. But Bartlett is better than lower order AR model.



MATLAB Code:

```
%Yule Walker Method for PSD Estimation

clear all;
% close all;
load('V:\EECS-844\Exam-4\P4.mat');
pn=[2 4 8];   %AR model order

for idx=1:length(pn)
 clearvars -except pn x idx
  p=pn(idx);
  M=p+1;    %No of AR coefficients
  K=length(x);
  N=K-M+1;

  for k=1:N
    X(:,k)=flipud(x(k:k+M-1));  %Snapshot matrix
  end
  R=1/N*X*X';    %Sample Correlation matrix from Data
  rxx=R(2:M,1);   %Autocorrelation
  Rxx=R(1:p,1:p);
  a=-inv(Rxx)*rxx;
  ar=[1;conj(a)];   %AR(p) model filter coefficients
  w=linspace(0,2*pi,1000);
  [H,W]=freqz(1,ar,1000,'whole');
  figure(1);hold on;plot(w,20*log10(abs(H)));
end
title('Yule Walker Method PSD Estimation');
xlabel('Normalized frequency rad/sample');
ylabel('Normalized Power in dB');
legend('p=2','p=4','p=8')
```

Q6. For different data sets X1, X2, X3 and X4, Bayesian Information Criteria was implemented using normal Sample Covariance Matrix (SCM). Which revealed following number of signals for the datasets.

For sample X1 no of signals is 1

For sample X2 no of signals is 1

For sample X3 no of signals is 6

For sample X4 no of signals is 6

Some of the assumptions made in this are that observed n different measurements are complex Gaussian distributed, the antenna arrays are Uniform Linear Arrays with no calibration errors. The signals are stationary ergodic Gaussian processes with zero mean. And SNR is greater than 15 dB and noise is white noise.

Matlab Code:

```
%BIC and MUSIC using normal sample correlation matrix

clear all;
close all
load('V:\EECS-844\Exam-4\P6.mat');

for sample=1:4
if sample==1
  X=X1;
elseif sample==2
  X=X2;
elseif sample==3
  X=X3;
else X=X4;
end

[M,N]=size(X);
p=M;      %Number of array elements
n=N;      %Number of time samples
R=1/n*(X*X');     %Correlation Matrix
eig_vals=eig(R);
eig_vals=sort(real(eig_vals),'descend'); %Sort if not sorted
%
for q=1:p
  term1=sum(log(eig_vals(q:p)));
  term2=(p-q+1).*log(sum(eig_vals(q:p)/(p-q+1)));
  Loglq=n*(term1-term2);
  BIC(q)=-2*Loglq+((q-1)*(2*p-q+1)+1)*log(n);
end

% figure;plot([1:p],(BIC)); title(sprintf('BIC for sample %d ',sample))
```

```
[min_val,min_idx]=min(BIC);

%Number of signals
p=min_idx-1;
fprintf('For sample %d no of signals is  %d\n',sample,p)

%MUSIC Implementation
[Q,D]=eig(R);
eig_values=real(diag(D));  %eigen values
sorted_eig=(sort(eig_values,'descend')); %Sorted eigen values
if D(1)~=sorted_eig(1)
  Q=fliplr(Q);      %Arrange Q based on decreading eigen values
end

sampling=600;  %Number of samples
phi=linspace(-pi/2,pi/2,sampling);
theta=pi*sin(phi);

for idx=1:sampling
  U=0;
  for k=p+1:M
    sv=transpose(exp((-1i)*theta(idx)*[0:M-1]));   %Steering vectors
    U=U+(abs(sv'*Q(:,k)))^2;
  end
  P(idx)=1/U;
end
figure(1);hold on; plot(theta *180/pi,20*log10(P));
end
figure(1); title('MUSIC Psedospectrum using Bayesian Information Criteria');
xlabel('Theta (deg)');
ylabel('Magnitude in dB')
legend('X1','X2','X3','X4')
```

Q7. Here BIC was applied to the Covariance matrix obtained from Forward Backward Averaging. Here the advantage of FB averaging is that the signal subspace is increased by a factor of 2 for coherent signals. The disadvantage of FB averaging is that It assumes ideal ULA so if there is calibration error it shows false signals.

For sample X1 no of signals is 2 (1 for SCM)

For sample X2 no of signals is 2 (1 for SCM)

For sample X3 no of signals is 6 (6 for SCM)

For sample X4 no of signals is 12 (6 for SCM)

The signals increased for all by a factor of two except X3.

MATLAB Code:

```
%BIC and MUSIC using FB Averaging

clear all;
close all
load('V:\EECS-844\Exam-4\P6.mat');

for sample=1:4
if sample==1
  X=X1;
elseif sample==2
  X=X2;
elseif sample==3
  X=X3;
else X=X4;
end

[M,N]=size(X);
p=M;      %Number of array elements
n=N;        %Number of time samples
J=flipud(eye(p));

R=(1/(2*n))*(X*X'+J*conj(X)*transpose(X)*J);     %Correlation Matrix using FB
averaging
eig_vals=eig(R);
eig_vals=sort(real(eig_vals),'descend'); %Sort eigen values if not sorted

for q=1:p
  term1=sum(log(eig_vals(q:p)));
  term2=(p-q+1)*log(sum(eig_vals(q:p)/(p-q+1)));
  Loglq=n*(term1-term2);
  BIC(q)=-2*Loglq+((q-1)*(2*p-q+1)+1)*log(n);
end

[min_val,min_idx]=min(BIC);

%Number of signals
p=min_idx-1;
fprintf('For sample %d no of signals is  %d\n',sample,p)

%MUSIC Implementation
[Q,D]=eig(R);
eig_values=real(diag(D));  %eigen values
sorted_eig=(sort(eig_values,'descend')); %Sorted eigen values
if D(1)~=sorted_eig(1)
  Q=fliplr(Q);     %Arrange Q based on decreasing eigen values
end

sampling=600;
phi=linspace(-pi/2,pi/2,sampling);
theta=pi*sin(phi);

for idx=1:sampling
  U=0;
```

```
  for k=p+1:M
    sv=transpose(exp((-1i)*theta(idx)*[0:M-1]));   %Steering vectors
    U=U+(abs(sv'*Q(:,k)))^2;
  end
  P(idx)=1/U;
end
figure(1);hold on;plot(theta*180/pi,20*log10(P));
end
figure(1); title('MUSIC Psedospectrum using FB Averaging');
xlabel('Theta (deg)');
ylabel('Magnitude in dB')
legend('X1','X2','X3','X4')
```

Q8 Here BIC was applied to SCM calculated by spatial smoothing which uses sub arrays which increases the rank of the SCM and can reveal more signals that are coherent. The error caused by FB can be removed by Spatial Smoothing as it is less sensitive to array calibrations or mismatch. But the disadvantage is that it degrades the resolution.

	X1	X2	X3	X4
SCM	1	1	6	6
FB Averaging	2	2	6	12
Spatial Smoothing	6	10	6	4

Here it has revealed more signals for X1 and X2. Undid the effects of array mismatch for X4 and for X3 no effects which seems to be ideal.

MATLAB Code:

```
%BIC and MUSIC Implementation using Spatial Smoothing

clear all;
close all
load('V:\EECS-844\Exam-4\P6.mat');


for sample=1:4
if sample==1
  X=X1;
elseif sample==2
  X=X2;
elseif sample==3
  X=X3;
else X=X4;
end

[M,N]=size(X);
p=M;  %Number of antenna
n=N;   %Number of time samples
Mbar=12; %Sub array size
K=M-Mbar+1; %Number of subarrays
Rss=zeros(Mbar,Mbar);
```

```
for subarrayidx=1:K

  Xnew=X(subarrayidx:subarrayidx+Mbar-1,:);
  Rss=Rss+Xnew*Xnew';       %Correlation matrix using Spatial Smoothing

end
Rss=Rss/(n*K);

eig_vals=eig(Rss);
eig_vals=sort(real(eig_vals),'descend'); %Sort eig vals if not sorted
p=Mbar;
for q=1:p
  term1=sum(log(eig_vals(q:p)));
  term2=(p-q+1)*log(sum(eig_vals(q:p)/(p-q+1)));
  Loglq=n*(term1-term2);
  BIC(q)=-2*Loglq+((q-1)*(2*p-q+1)+1)*log(n);
end

%figure;plot([1:p],(BIC))
[min_val,min_idx]=min(BIC);

%Number of signals
p=min_idx-1;
fprintf('For sample %d no of signals is  %d\n',sample,p)

%MUSIC Implementation
[Q,D]=eig(Rss);
eig_values=real(diag(D));  %eigen values
sorted_eig=(sort(eig_values,'descend')); %Sorted eigen values
if D(1)~=sorted_eig(1)
  Q=fliplr(Q);     %Arrange Q based on decreasing eigen values
end

sampling=600;
phi=linspace(-pi/2,pi/2,sampling);
theta=pi*sin(phi);

for idx=1:sampling
  U=0;
  for k=p+1:Mbar
    sv=transpose(exp((-1i)*theta(idx)*linspace(0,Mbar-1,Mbar)));   %Steering
vectors
    U=U+(abs(sv'*Q(:,k)))^2;
  end
  P(idx)=1/U;
end
figure(1);hold on;plot(theta*180/pi,20*log10(abs(P)));
end
figure(1); title('MUSIC Psedospectrum using Spatial Smoothing');
xlabel('Theta (deg)');
ylabel('Magnitude in dB')
legend('X1','X2','X3','X4')
```

Q9.

Here as obtained from before the number of signals estimated are:

	X1	X2	X3	X4
SCM	1	1	6	6
FB Averaging	2	2	6	12
Spatial Smoothing	6	10	6	4

Here X3 has 6 signals for all so it seems to be the ideal case and it has no calibration errors and the signals are uncorrelated. For X4 there are 6 signals shown by SCM but doubled by FB averaging and SS showed only 4 signals so it must have some calibration error as SS is less prone to calibration error but FB averaging is. So X4 is realistic array but the signals are uncorrelated.

For X1 there are 1, 2 and 6 signals so this seems to be the temporally correlated case and it doesn't seem to have calibration errors. X2 has some calibration errors as it shows more signals than there is which is 6.

Ideal Array & Temporally Uncorrelated	X3
Ideal Array & Temporally Correlated	X1
Realistic Array & Temporally Uncorrelated	X4
Realistic Array & Temporally Correlated	X2

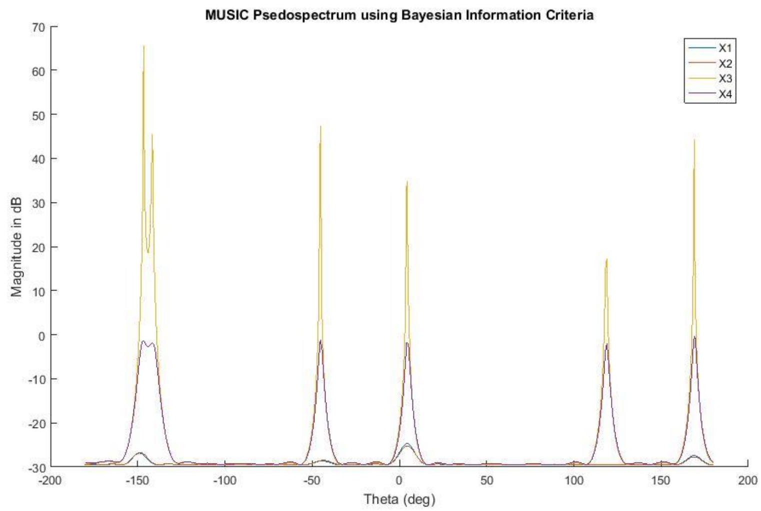
Q10. Using all methods MUSIC was implemented. Here in all methods SCM, FB and SS 6 signals are distint in all cases for X3 which is the ideal array. Also 6 signals can be seen for X4 but for X1 and X2, the signal peaks are not distinct or prominent. X2 has calibration errors so it is showing more peaks in MUSIC .

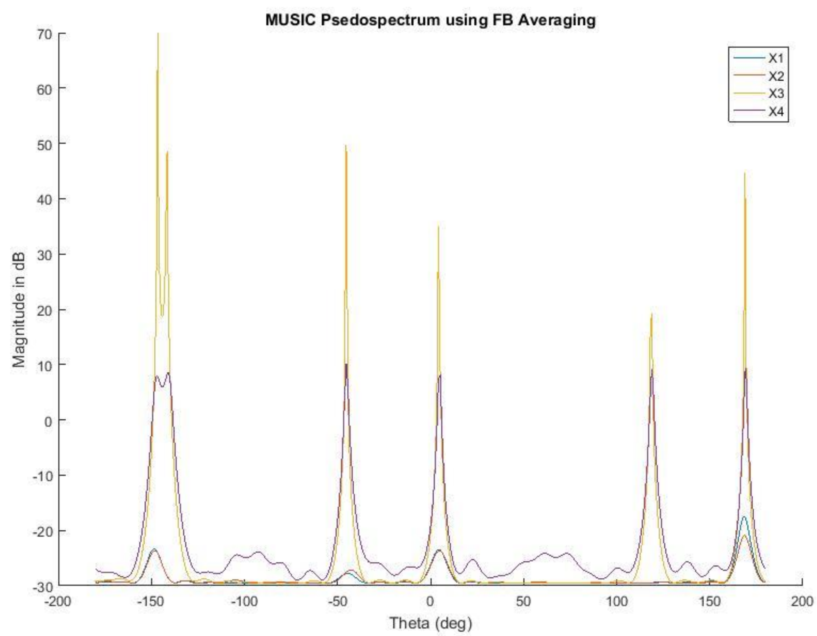
For SCM, X1 shows more 4 peaks same for X2, X3 has 6 and X4 has 6.

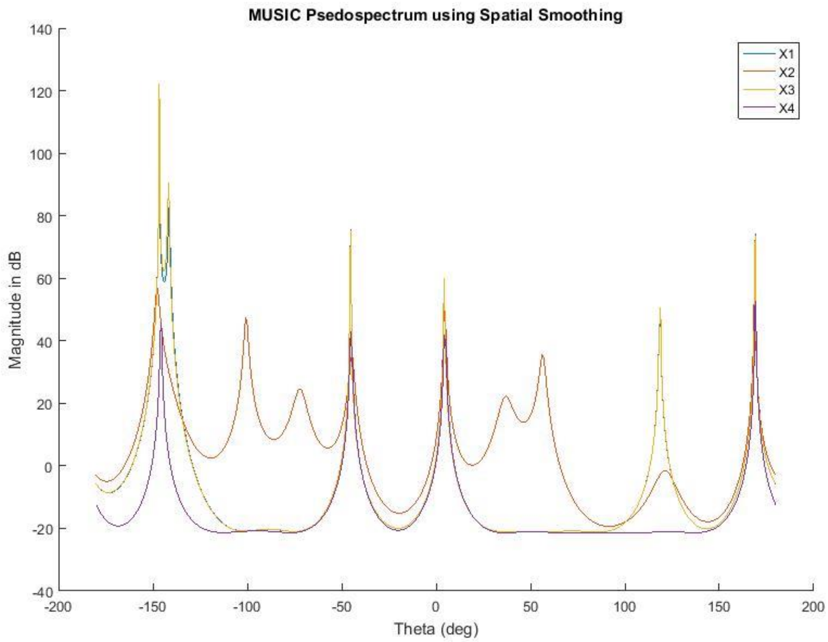
For fb: X1 has 4 peaks , so has X2, X3 and X4 has 6

For SS: X1 has 6, X2 has 9 peaks, X3 has 6 and X4 has 4 peaks.

Here so ideal arrays like X3 has better performance in MUSIC as MUSIC is affected by model mismatch and array calibration and temporal correlation.


MUSIC Psedospectrum using Bayesian Information Criteria
Magnitude in dB
Theta (deg)
X1
X2
X3
X4


MUSIC Psedospectrum using FB Averaging
Magnitude in dB
Theta (deg)
X1
X2
X3
X4



MATLAB Code:

Implemented in 6, 7, 8