

Advanced Signal Processing (ASP)

Coursework 1

Name: Di Wu

CID: 00646091

Login: dw1310

Declaration: I confirm that this submission is my own work. In it, I give references and citations whenever I refer to or use the published, or unpublished, work of others. I am aware that this course is bound by penalties as set out in the College examination offenses policy.

Signed: Di Wu

Aim: The aim of the first lab is to provide a basic understanding of statistical estimation. In this exercise random signals will be generated using MATLAB and the effect of the random signals on linear systems will be examined and discussed. This coursework exercise will serve as an introduction to the Advanced Signal Processing course.

1. Statistical Estimation

1.1 Sample Mean of Uniform Distribution

First 1000 sample realizations of X_n where $X_n \sim U(0, 1)$, $\forall k$ is generated and plotted using the rand function on MATLAB

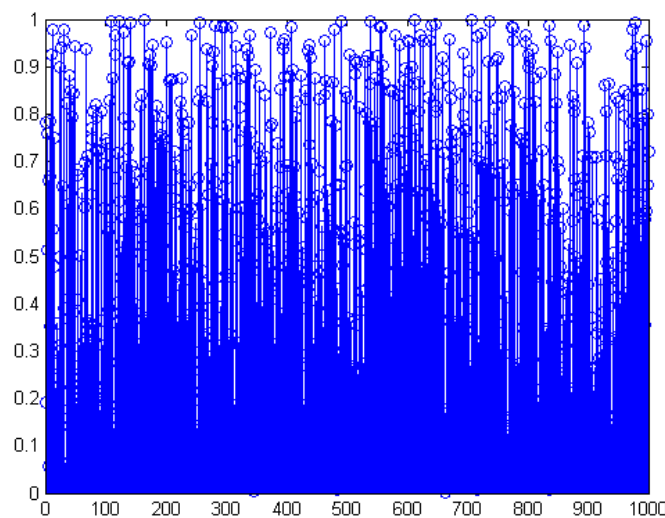


Fig 1. 1000 realizations of the uniform distribution $X_n \sim U(0, 1)$

The theoretical mean $m = E\{X\}$ of the uniform distribution $X_n \sim U(0, 1)$ is **0.5** and using the mean function and the expression $\frac{1}{N} \sum_1^N x[n]$ we can calculate the sample mean \hat{m} of the 1000 samples. MATLAB calculated the sample mean to be

```
3 - N = 1000; %number of samples
4 - x = rand([N,1]); %generate one column of 1000 samples
5 - mean = sum(x)/N; %find the sample mean
```

mean = 0.4968

we can see that the estimate is slightly off from the theoretical mean.

1.2 Sample Standard Deviation of Uniform Distribution

The theoretical standard deviation of the uniform distribution is $\sigma = \sqrt{1/12} = \mathbf{0.2887}$. Using the MATLAB std function and the expression $\hat{\sigma} = \sqrt{\frac{1}{N-1} \sum_1^N (x[n] - \hat{m})^2}$, we can evaluate the sample standard deviation of the 1000 samples to be

```
N = 1000; %number of samples
x = rand([N,1]); %generate one column of 1000 samples
k = x - sample_mean;
x = x.*x;
sample_std = sqrt(sum(x)*(1/(N-1))); %find sample standard deviation
```

sample_std = 0.2857

Noice that the sample standard deviation is also slightly off from the theoretical standard deviation.

1.3 Bias in Estimations

Then by generating ten 1000 sample realizations of X , we calculate the sample means and sample standard deviations for each realizations.

Sequence	Sample mean	Sequence	Sample std
m1	0.4888	σ_1	0.2878
m2	0.4920	σ_2	0.2853
m3	0.4895	σ_3	0.2799
m4	0.5218	σ_4	0.2846
m5	0.5012	σ_5	0.2881
m6	0.5047	σ_6	0.2971
m7	0.4919	σ_7	0.2811
m8	0.4997	σ_8	0.2896
m9	0.5096	σ_9	0.2844
m10	0.4821	σ_{10}	0.2845

Table 1. sample means and standard deviation for ten 1000 samples realizations

By plotting $m_{1:10}$ and $\sigma_{1:10}$ we can observe clearly that the estimates cluster tightly around the theoretical values.

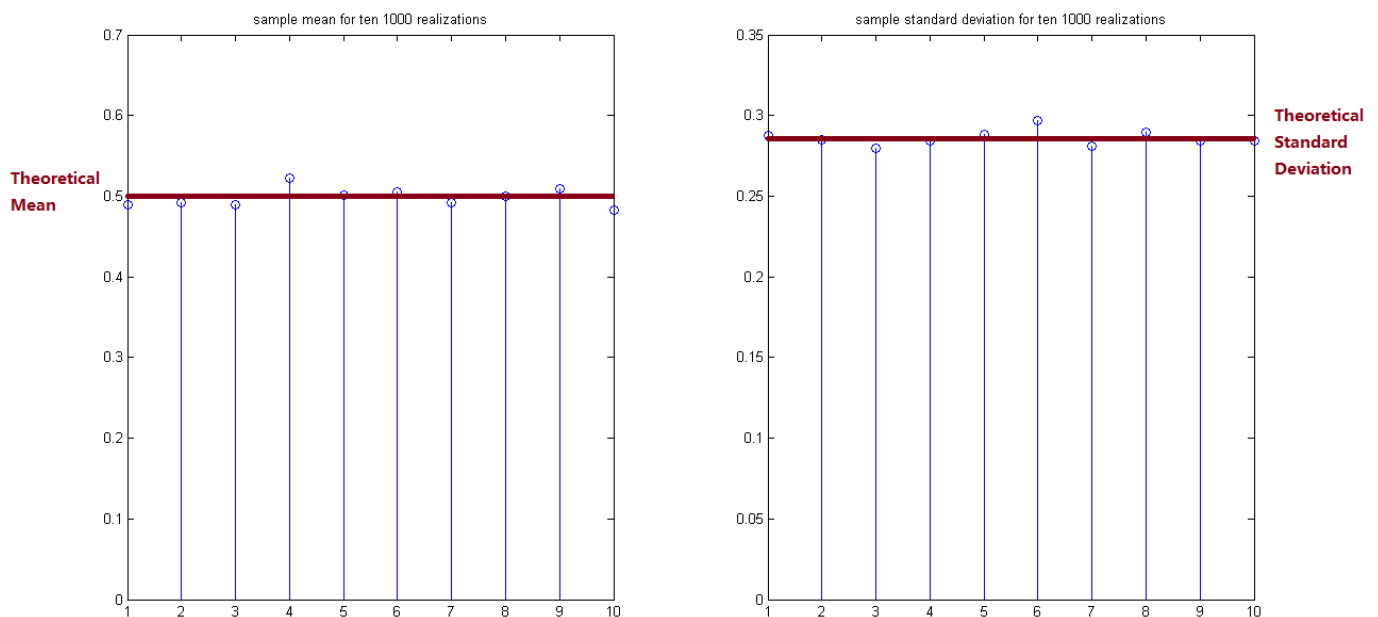


Fig 2. Estimated sample mean and sample standard deviation for ten 1000 realizations

The estimation bias of the sample mean can be found by the equation $B = E\{X\} - m$. Since the values of $m_{1:10}$ are clustered closely to the theoretical mean, we can expect the estimation bias B to be fairly small as well.

```
theoreticalmean = 0.5;
b = sample_mean - theoreticalmean; %find bias
stem(b,'x');title('Estimation Bias of ten 1000 samples') % plot bias
```

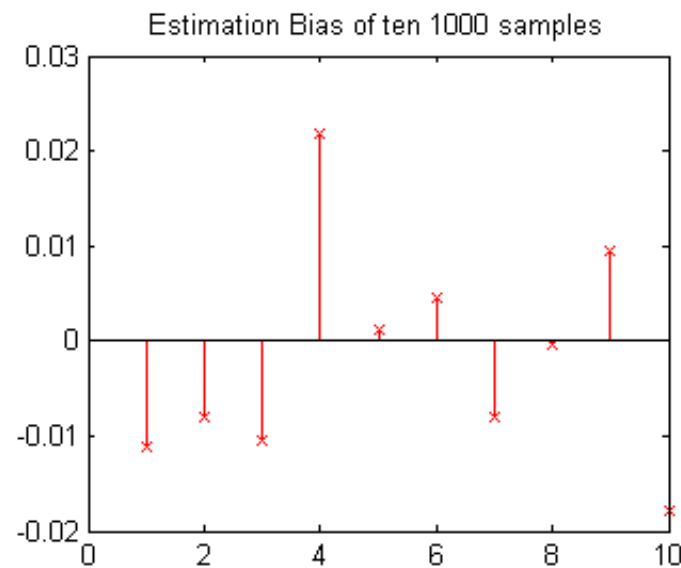


Fig 3. Sample mean estimation bias for the ten 1000 samples realizations

1.4 Probability Density Function of Uniform Distributed Samples

We can use the hist function in MATLAB to approximate the pdf of X , we first draw the pdf using 1000 samples.

```
N=1000; x = rand([N,1]); %generate 100 samples
[count , xaxis] = hist(x1); count = count/N; %calculate and draw pdf
bar(xaxis,count);title('pdf of 1000 uniform distributed samples')
```

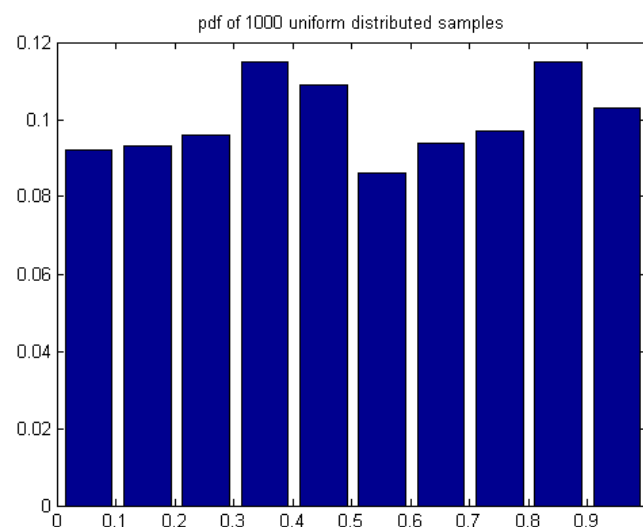


Fig 4. pdf of 1000 uniformly distributed samples

We know that the theoretical pdf of the uniform distribution should appear as follows:

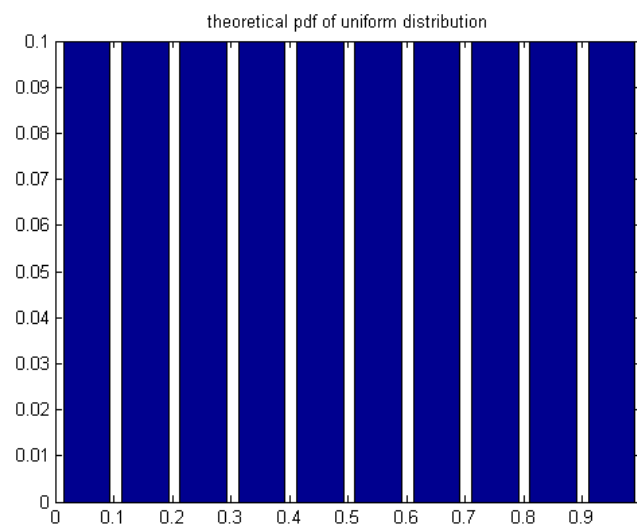


Fig 5. Theoretical pdf of uniform distribution

However, if we increase the number of samples used, we can see that the pdf of the samples will approach the theoretical pdf of the uniform distribution.

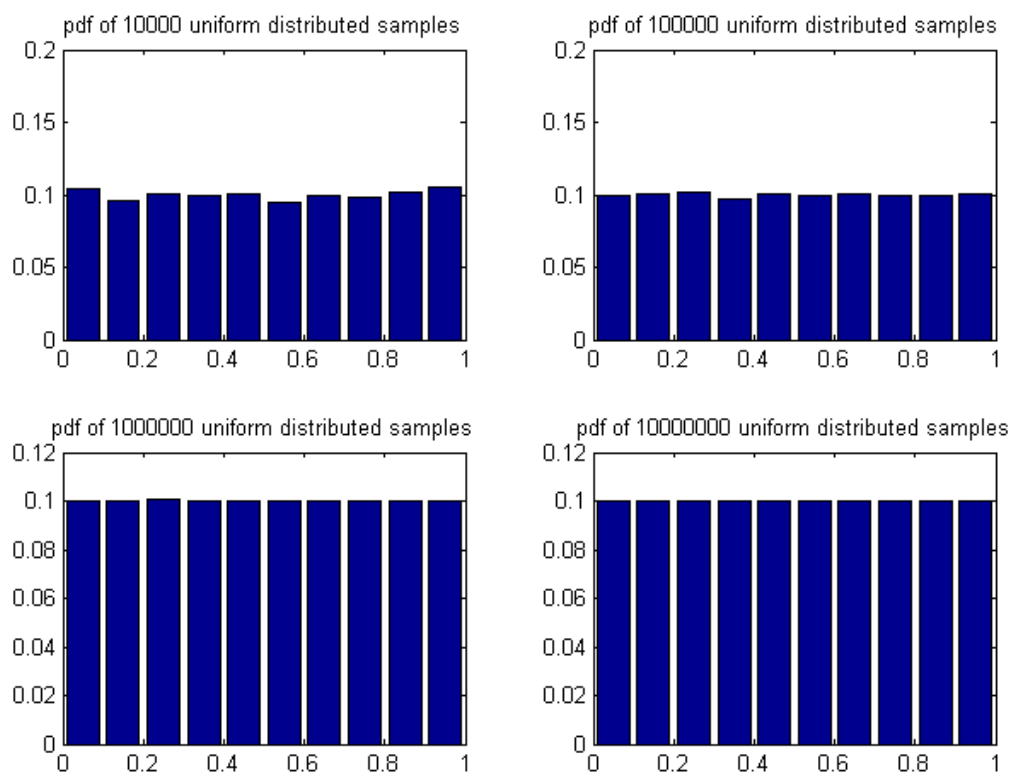


Fig 6. pdf of increasing number of uniformly distributed samples

1.5 Sample Mean of Gaussian Distribution

Having investigated the statistical properties of the samples generated from a uniform distribution, we now focus our attention to the Gaussian Distribution. The `randn` function can be used in MATLAB to generate 1000 sample realizations of the zero mean and unit standard deviation Gaussian random distribution.

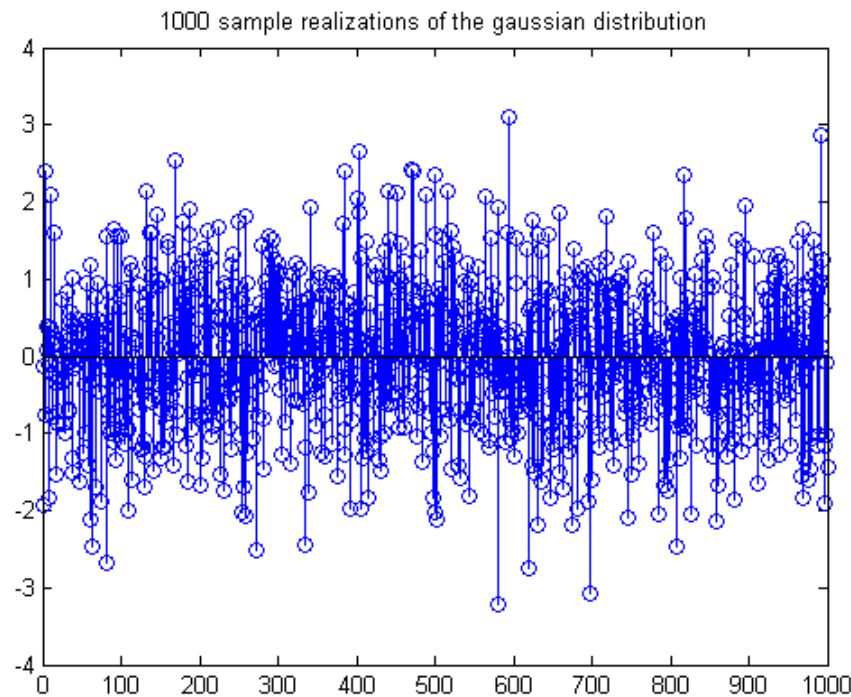


Fig 7. 1000 realizations of the zero mean unit standard deviation Gaussian distribution

The sample mean of the 1000 sample realizations is found using the same functions as the one used for the uniform distribution.

```
N=1000; %define number of samples
x = randn([N,1]); %generate one column of 1000 samples
sample_mean = sum(x)/N; %calculate the sample mean

sample_mean = -0.0271
```

The sample mean is found to be -0.0271 which, as before, is slightly off from the theoretical mean of zero.

Sample Standard Deviation of Gaussian Distribution

Repeating the analysis for the sample standard deviation using the equation

$\hat{\sigma} = \sqrt{\frac{1}{N-1} \sum_{n=1}^N (x[n] - \hat{m})^2}$ we can calculate the sample standard deviation as follows

```
d = x - sample_mean; %difference to the sample mean
d = d.*d; %take square of the difference
sample_std = sqrt(sum(d)*(1/(N-1))); %find sample standard deviation
sample_std = 0.9711
```

The theoretical standard deviation is 1, and the sample standard deviation is again slight different from the theoretical standard deviation.

Bias in Estimations

Then by generating ten 1000 samples realizations again, we can observe the sample mean and sample standard deviations $\hat{m}_{1:10}$ and $\hat{\sigma}_{1:10}$

Sequence	Sample mean	Sequence	Sample std
m1	-0.0100	σ_1	0.9928
m2	-0.0063	σ_2	0.9656
m3	0.0046	σ_3	1.0255
m4	0.0141	σ_4	1.0201
m5	0.0073	σ_5	0.9537
m6	0.0273	σ_6	1.0119
m7	-0.0084	σ_7	1.0310
m8	-0.0286	σ_8	0.9728
m9	0.0157	σ_9	1.0522
m10	0.0184	σ_{10}	1.0045

Table 2. sample means and standard deviation for ten 1000 samples realizations

As with before these values closely cluster to their theoretical values.

And the estimation bias of the ten 1000 samples can be represented as follows:

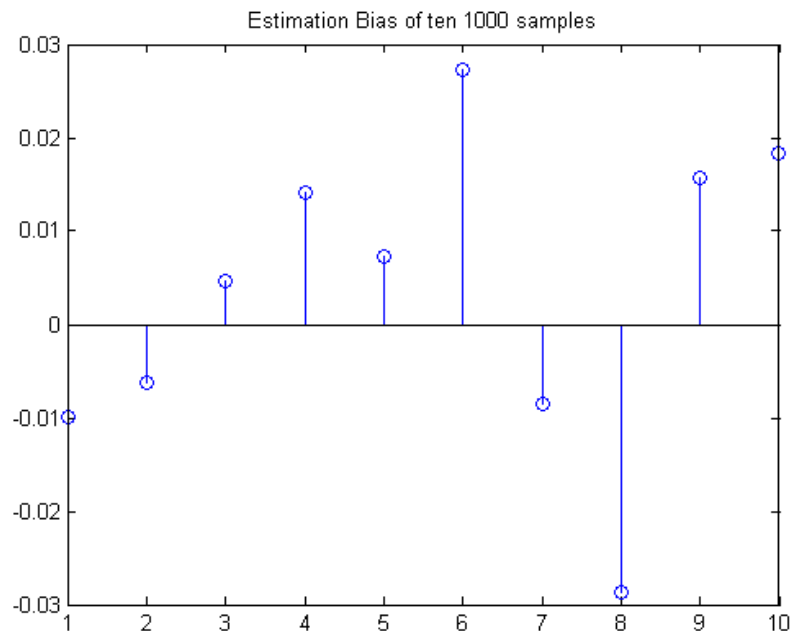


Fig 8. estimation bias of sample mean for the ten 1000 samples realizations

Probability Density Function of Gaussian Distributed Samples

The hist function is used again to approximate the pdf of the Gaussian distributed samples.

```
N=1000;                                %define number of samples
x = randn([N,1]);                       %generate one column of 1000 samples
[count , xaxis] = hist(x); count = count/N; %calc the pdf
bar(xaxis,count);title('pdf of 1000 uniform distributed samples')hold on;
plot(xaxis,count,'r');hold off
```

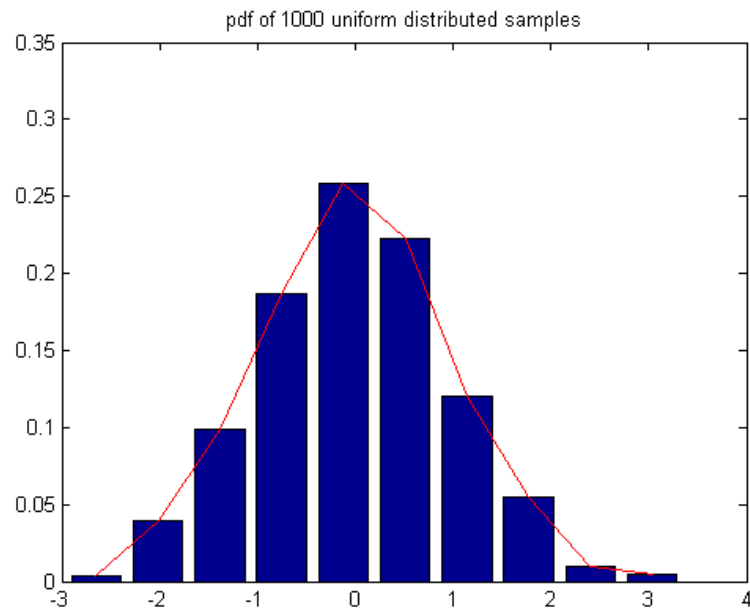


Fig 9. pdf of 1000 uniformly distributed samples

The theoretical pdf of the zero mean unit standard deviation Gaussian distribution can be represented as follows:

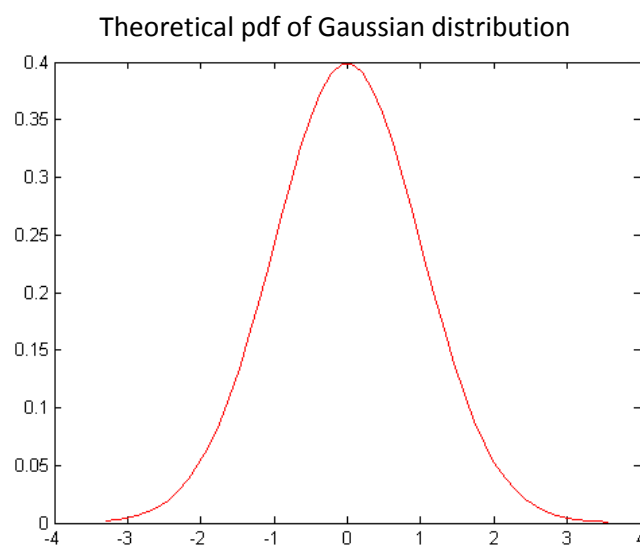


Fig 10. Theoretical pdf of Gaussian distributed samples

By increasing the number of samples again, we could observe that as the number of samples increases, the pdf of the sample approach the theoretical pdf.

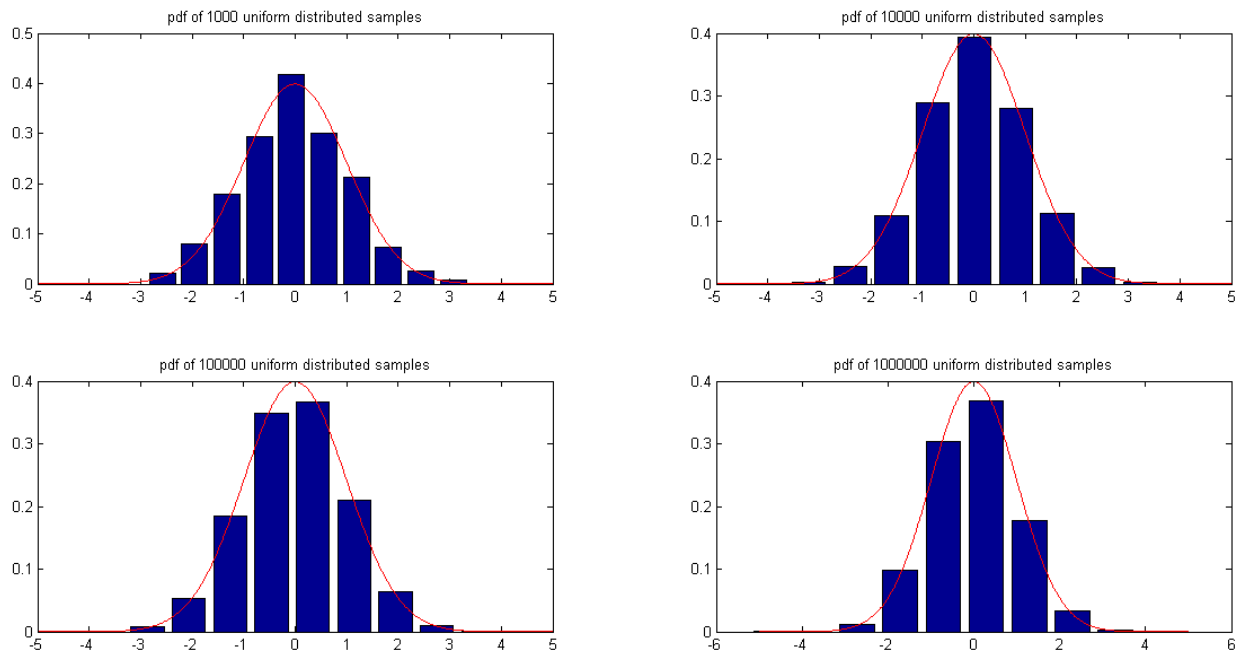


Fig 11. pdf of increasing number of Gaussian distributed samples

2. Stochastic Processes

2.1 ensemble mean and standard deviation

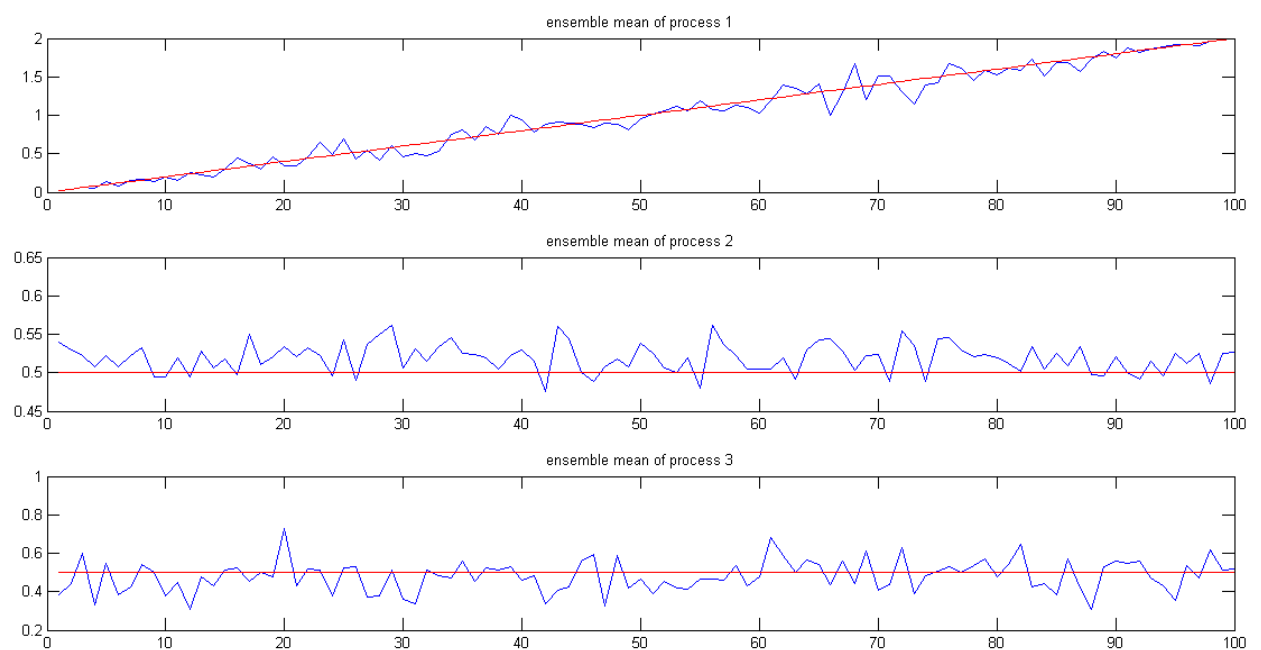


Fig 12. Ensemble mean of three stochastic processes

The ensemble mean of the three stochastic processes with $M=100$ and $N=100$ are generated and plotted. The red line in the plots represents the theoretical mean of each process. We can see that having taken the ensemble mean for $M=100$, the ensemble mean closely follows the theoretical mean. However if we reduce the number of M to 1, we could observe that the signal to noise ratio would increase.

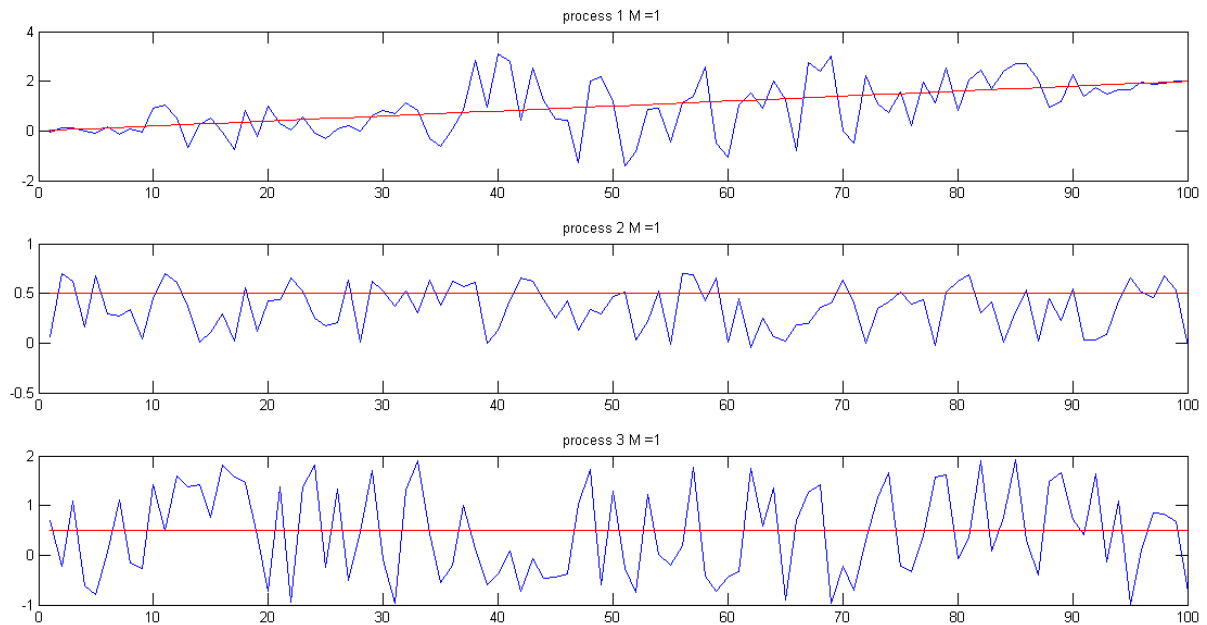


Fig 13. three stochastic processes when $M = 1$

The ensemble standard deviation of the three processes when $M = 100$ can be plotted as follows:

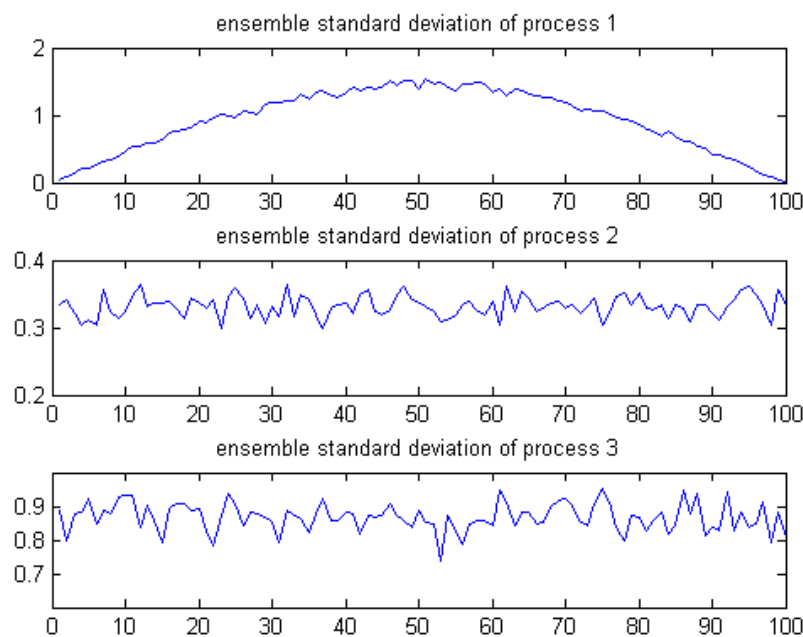


Fig 14. Ensemble standard deviation of three stochastic processes

The criteria used to determine whether if a process is stationary are as follows:

- ✧ Mean remains constant with time variance
- ✧ Constant Variance
- ✧ Covariance only depends on the time lag

By observing the plot of ensemble mean and ensemble variance we can clear see that process 1 is non-stationary whereas processes 2 and 3 are stationary.

Process Name	Stationary
Process 1	No
Process 2	Yes
Process 3	Yes

Table 3. Stationarity of three random processes

2.2 Ergodicity

It is very common in real life situations that we only have one observation of a signal and it would be therefore impossible to find the statistical average. In such situations we could find the time averages instead of the statistical average(ensemble mean).

A wide sense stationary stochastic process is said to be ergodic if the sample mean and standard deviation of the process converges. (i.e. the mean and standard deviation can be deduced from a sufficiently long realization)

In this exercise, we are asked to generate 4 observations with 1000 samples for each of the three processes discussed in 2.1 and calculate the sample mean and standard deviation for each of the observation.

Process	Time average 1	Time average 2	Time average 3	Time average 4
Process 1	10.058	10.028	10.066	10.013
Process 2	0.229	0.061	0.920	0.918
Process 3	0.484	0.514	0.541	0.517

Table 4. Time average for 4 observations

Process	Std of observe 1	Std of observe 2	Std of observe 3	Std of observe 4
Process 1	5.871	5.902	5.843	5.877
Process 2	0.180	0.127	0.215	0.057
Process 3	0.864	0.860	0.873	0.844

Table 5. Standard deviation of 4 observations

Since process 1 is not stationary, it is not ergodic. For process 2, the sample mean for a large number of samples does not converge to the theoretical mean and is hence non-ergodic. Whereas for process 3 the process is ergodic

Process Name	Ergodic
Process 1	No
Process 2	No
Process 3	Yes

Table 6. Ergodicity of three random processes

2.3 Theoretical mean/variance

Process 1 can be represented mathematically as follows:

$$y_1[n] = 5*a[n]*\sin(n\omega_0)+0.02*n$$

$$a[n] \sim U(-0.5,0.5)$$

the mean of process 1 is $E\{y_1[n]\}$ for a large number of realization is therefore

$$E\{y_1[n]\} = E\{5*a[n]*\sin(n\omega_0)\} + E\{0.02n\}$$

$$E\{y_1[n]\} = 0.02n$$

Its theoretical variance is calculated as

$$\text{Var}\{y_1[n]\} = \text{Var}\{5*a[n]*\sin(n\omega_0)\} + \text{Var}\{0.02n\}$$

$$\text{Var}\{y_1[n]\} = 25* \text{Var}\{a[n]*\sin(n\omega_0)\} + \text{Var}\{0.02n\}$$

$$\text{Var}\{y_1[n]\} = \frac{25}{12}\sin(n\omega_0)^2$$

$$\sigma = \sqrt{\frac{25}{12}\sin(n\omega_0)^2}$$

Process 2 can be represented mathematically as follows:

$$y_2[n] = a[n]*b[n]+c[n]$$

$$a[n] \sim U(-0.5,0.5)$$

$$b[n] \sim U(0,1)$$

$$c[n] \sim U(0,1)$$

Its theoretical mean is calculated as

$$E\{y_2[n]\} = E\{a[n]b[n]\}+E\{c[n]\}$$

As $a[n]$ and $b[n]$ are independent

$$E\{y_2[n]\} = E\{a[n]\}E\{b[n]\}+E\{c[n]\}=0.5$$

Its theoretical variance is calculated as

$$\text{Var}\{y_2[n]\} = \text{Var}\{a[n]b[n]\}+\text{Var}\{c[n]\}$$

$$\text{Var}\{y_2[n]\} = \text{Var}\{a[n]\}\text{Var}\{b[n]\}+\text{Var}\{a[n]\}^2E\{b[n]\}^2 + \text{Var}\{b[n]\}^2E\{a[n]\}^2+\text{Var}\{c[n]\}$$

$$\text{Var}\{y_2[n]\} = \frac{1}{12}*\frac{1}{12} + \frac{1}{12}*0.25 + 0 + \frac{1}{12} = \frac{1}{9}$$

$$\sigma = \frac{1}{3}$$

Process 3 can be represented mathematically as follows:

$$y_3[n] = 3*a[n] + 0.5$$

$$a[n] \sim U(-0.5,0.5)$$

Its theoretical mean is calculated as

$$E\{y_3[n]\} = E\{3*a[n]\} + E\{0.5\}$$

$$E\{y_3[n]\} = 0.5$$

Its theoretical variance is calculated as

$$\text{Var}\{y_3[n]\} = \text{Var}\{3*a[n]\} + \text{Var}\{0.5\}$$

$$\text{Var}\{y_3[n]\} = 9*\text{Var}\{a[n]\} + 0$$

$$\text{Var}\{y_3[n]\} = \frac{9}{12}$$

$$\sigma = \sqrt{\frac{9}{12}}$$

Comparing these theoretical values with the value obtained by sample averaging, we can see that the two values agree quite well. The red line represents the theoretical mean and the blue line represent the mean obtain from sample averaging. 100 realizations of 100 samples are generated for each process.

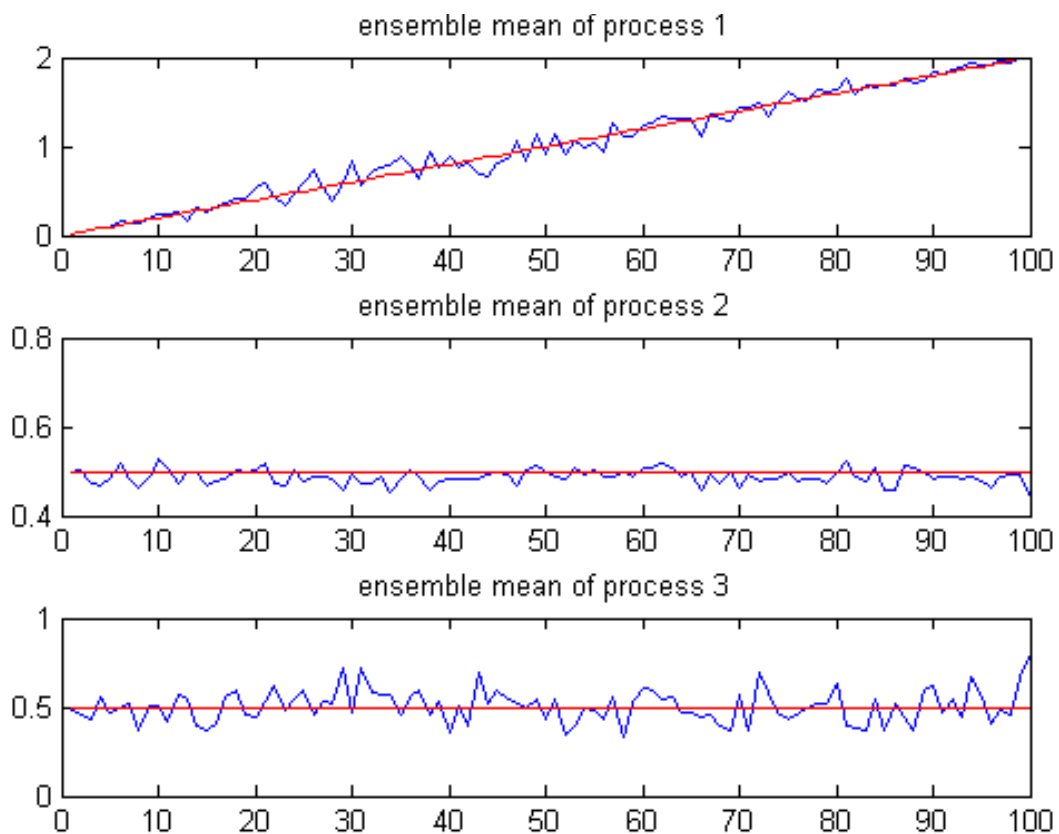


Fig 15. Comparison of theoretical mean to sample averaged mean

For the case of variance (standard deviation is plotted in this case), the theoretical value also agrees quite well with the sample averaged value as shown below for the three processes.

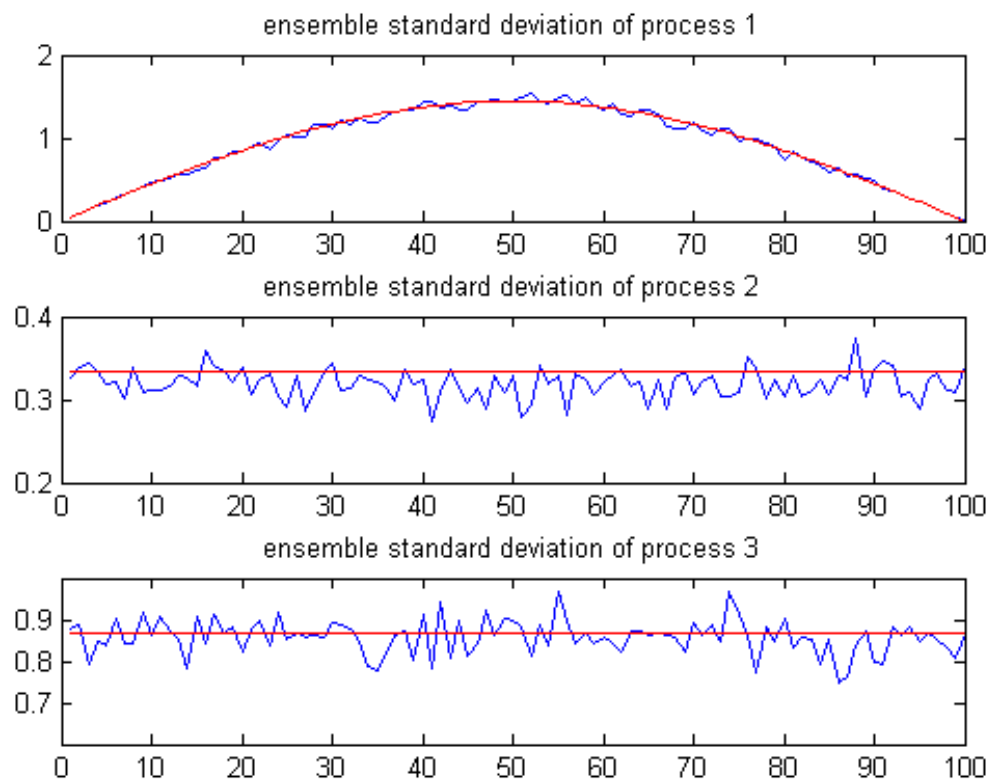


Fig 16. Comparison of theoretical standard deviation to sample averaged standard deviation

3. Estimation of Probability Distributions

3.1 Designing a pdf function

A function called mypdf is written in MATLAB, this functions takes an array o randomly generated samples as its input and calculate the pdf of these samples.

```
function mypdf(v);  
%function takes samples generated from a random variable  
[f x] = hist(v); %calculate and draw pdf  
bar(x,f/length(v));title('probability distribution function')  
end
```

The function is first tested using a Gaussian pdf $v = \text{randn}(1,N)$ with $N = 100$. The following pdf is returned which demonstrates that the pdf function is working as expected.

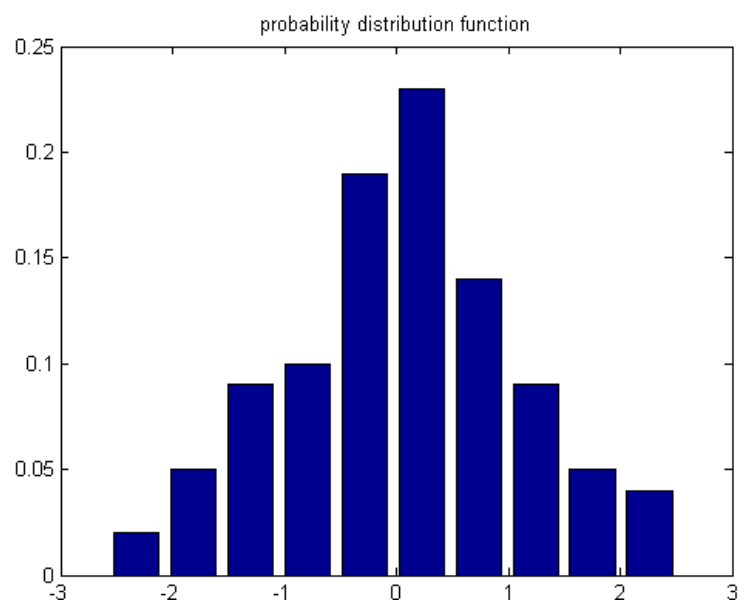


Fig 17. Pdf of Gaussian distribution $v = \text{randn}(1,N)$

3.2 pdf of Stationary and Ergodic Process

We know that the third stochastic process in section 2 is both stationary and ergodic, so by plotting the pdf of this process for $N \in \{100, 1000, 10000\}$ we compare the pdf for this process with its theoretical pdf (in red). We can see that as the number of samples is increased the pdf of the ergodic process approaches the theoretical pdf.

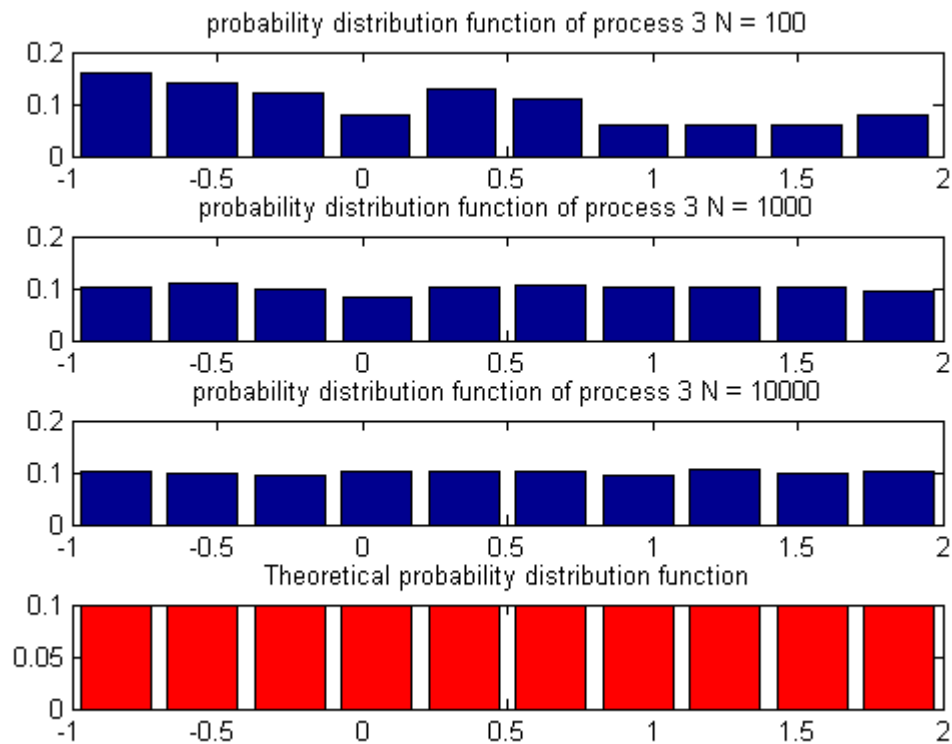


Fig 17. Pdf of process 3

3.3 pdf of non-stationary process

It would not be possible to estimate the pdf for a non-stationary process using the mypdf function that was created for the previous exercises. This is because the mypdf function exploits a MATLAB hist function. The hist function records the frequency that each item appears. As for the case of process 1, as we move in time, the output of the non-stationary process increases, so it would not be possible for the hist function to tally the frequencies correctly. Furthermore, the pdf of a non-stationary process is changing and depends on the time. Thus it would not make much sense for the pdf of such a process to be estimated.

Advanced Signal Processing (ASP)

Coursework 2

Name: Di Wu

CID: 00646091

Login: dw1310

Declaration: I confirm that this submission is my own work. In it, I give references and citations whenever I refer to or use the published, or unpublished, work of others. I am aware that this course is bound by penalties as set out in the College examination offenses policy.

Signed: Di Wu

Aim: The aim of the second coursework is to introduce us to some practical estimators of the autocorrelation and cross-correlation functions and to gain some degree of understanding about linear stochastic modelling i.e. autoregressive modelling

1 Autocorrelation Function of Uncorrelated Sequences

1.1 ACF of WGN

The autocorrelation function of the White Gaussian Noise is generated and plotted using the following MATLAB command

```
N=1000;           %using 1000 samples
x = randn([1,N]);
y=xcorr(x,'unbiased'); %find the ACF
tau=-999:1:999; % generate the xaxis label
plot(tau,y)
```

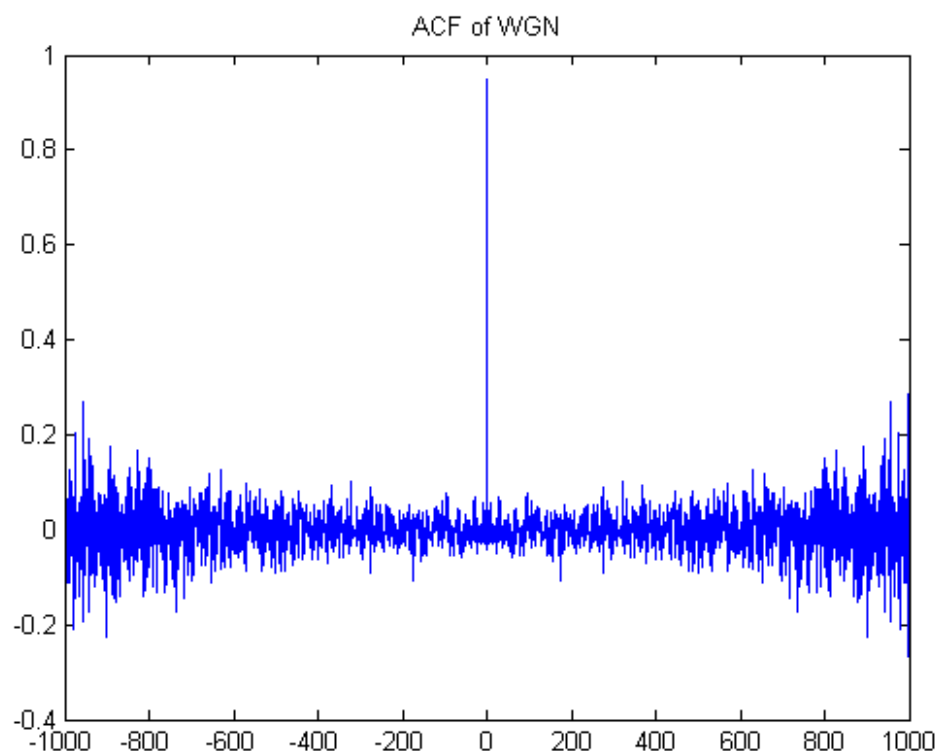


Fig 1. Autocorrelation function of WGN for $N = 1000$

From the graph we can observe that the autocorrelation function has a spike of 1 at $\tau = 0$, this is because that the samples in the WGN is independent and one sample does not affect another. We can also observe that the ACF of the WGN is symmetric, this arises from the stationary properties of the WGN. For stationary processes $\tau = -\tau$ implying that $R(\tau) = R(-\tau)$.

1.2 Zoomed ACF of WGN

If we zoom in to the range of $|\tau| < 50$, the auto-correlation function appears as follows:

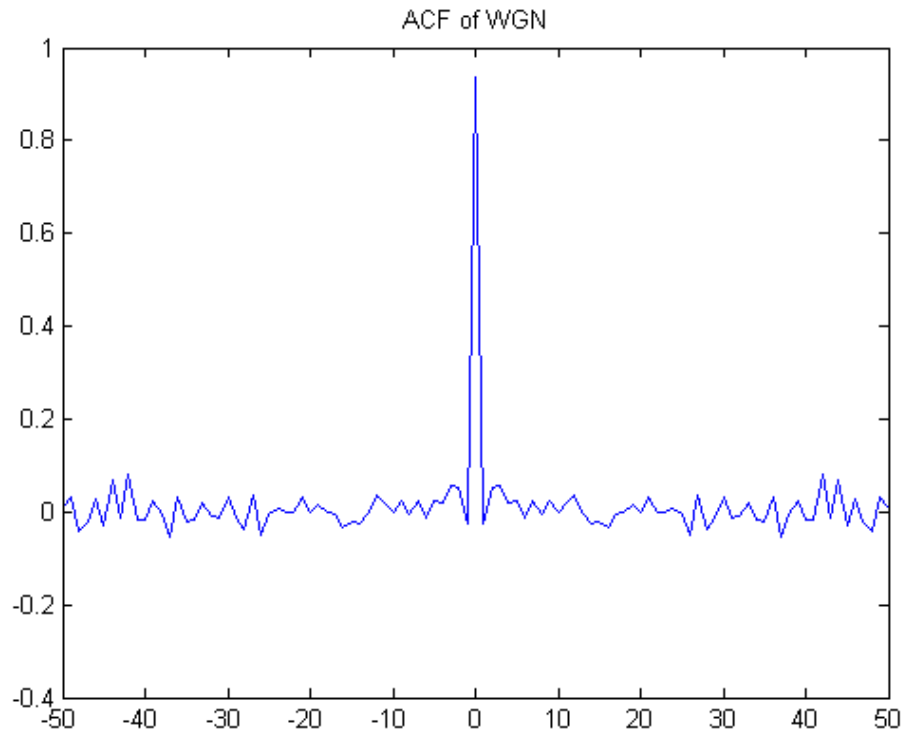


Fig 2. Autocorrelation function of WGN for $N = 1000$ (zoomed)

The variation in the value of the autocorrelation functions for small values of τ appears to be smaller than that if τ is large. This arises from the definition of the autocorrelation function for a finite number of samples:

$$\hat{R}_x(\tau) = \frac{1}{N-|\tau|} \sum_{n=1}^{N-|\tau|} x[n]x[n + \tau]$$

Notice that when τ is small the autocorrelation function is found by averaging the product of many terms. The large number of averaged values means that the auto-correlation at smaller values of τ is closer to the theoretical value. (zero in this case as the WGN is generated by an independent process)

1.3 Accuracy of the ACF with large autocorrelation lag τ

Following the discuss highlighted in the previous section the value of the autocorrelation function for large value of τ is no longer accurate due to the lack of the number of samples that can be averaged. For example for $N = 1000$ and $\tau=999$, the autocorrelation contains only one product term. A sensible empirical bound for τ from observing figure 1 should be set to $|\tau| < 800$. In this case the autocorrelation function would at least contain the average of 200 products.

2 ACF of filtered sequences

2.1 ACF and MA9 filter

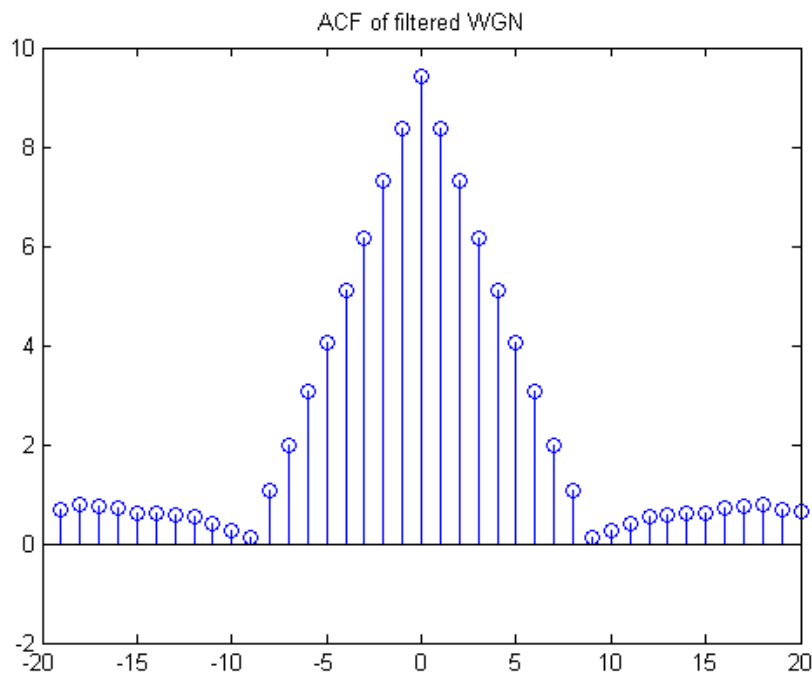


Fig 3. Autocorrelation function of filtered WGN with an MA(9) filter

A moving average 9 filter's output depends on 9 of its previous inputs, thus for the ACF of the output we would expect it to have a cut off after $\tau > q$ where q is the order of the MA filter.

Note that the sample ACF does not exactly match the theoretical pattern of the MA(9) filter. In the theoretical pattern all ACF for $\tau > q$ will be 0. If we are to generate a different set of Gaussian sequence x , then the ACF of such a sequence will be again different from the one shown above. However, for an MA process with the same order, the general features of the ACF of the MA process will be preserved.

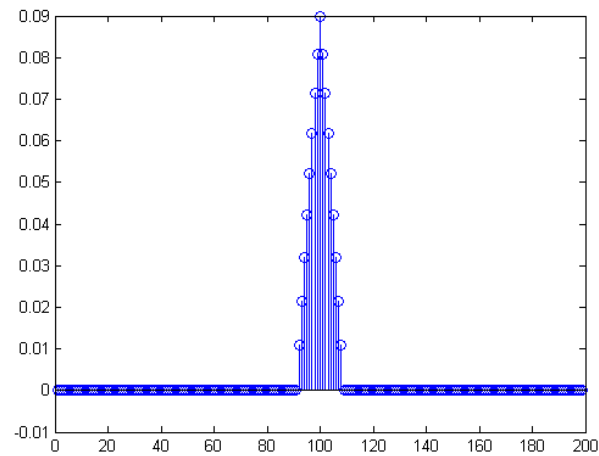
With regard to the output signal, as the length of the MA filter is increased, the variation in the output of the filter would also increase.

2.2 autocorrelation of input, output and filter's impulse response

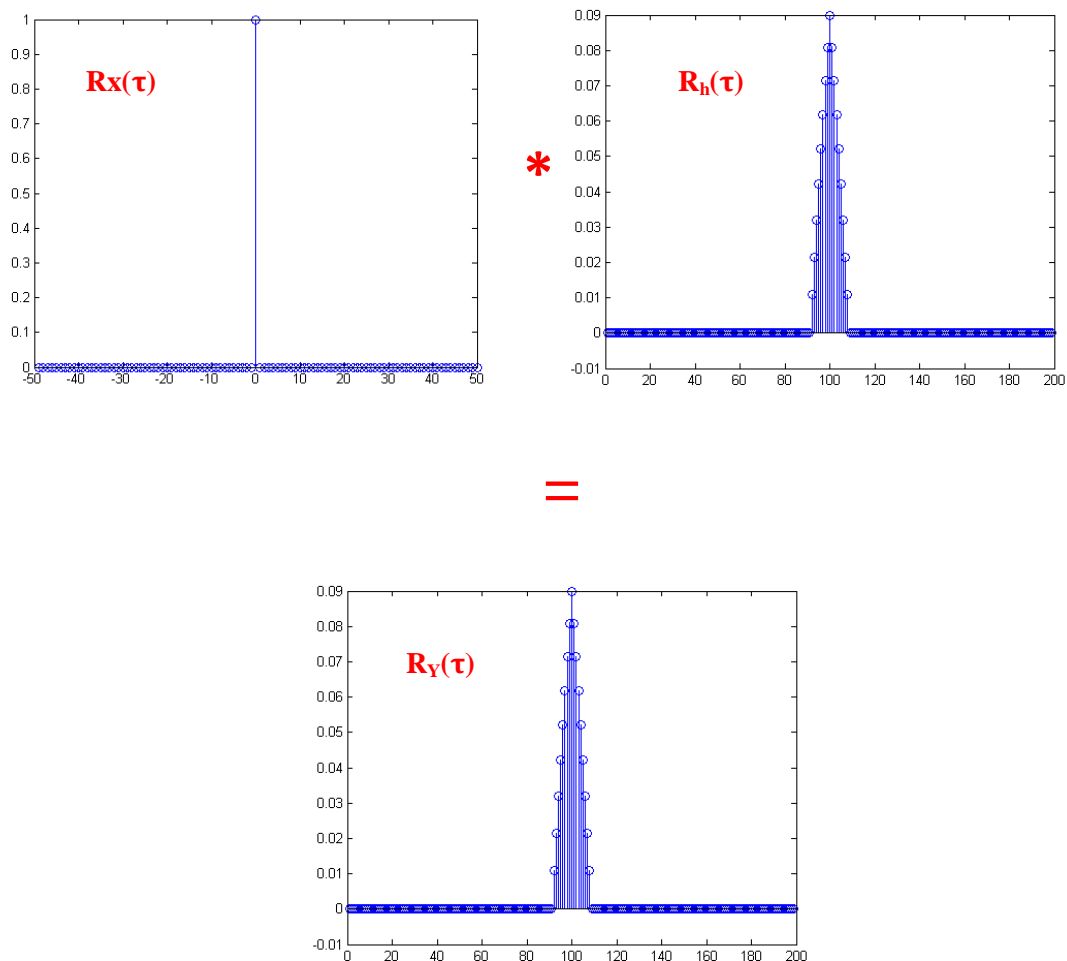
If X_n is an uncorrelated process then the autocorrelation function of X_n ($R_x(\tau)$) would be a dirac function. Since the autocorrelation function of the output represents a convolution between the input autocorrelation and autocorrelation of the impulse response. The autocorrelation of the output would simply be a mirrored version of the impulse response autocorrelation about the y-axis.

The autocorrelation of the impulse response can be generated by the following MATLAB command

```
x=zeros(1,50);
x(1)=1;          %generate an impulse for the input
y=filter(ones(1,9),1,x);      %filter using an MA9 process
autocorr = xcorr(y,'unbiased'); %find Rh(TAU)
stem(autocorr);           %plot Rh(TAU)
```

Fig 4. Autocorrelation function of $R_h(\tau)$

Hence $R_Y(\tau) =$

Fig 5. Demonstration of the relationship between $R_X(\tau)$, $R_Y(\tau)$ and $R_h(\tau)$

This can be verified by plotting the ACF of the output directly, and from observation, we see that $R_Y(\tau)$ is of the same shape as the above figure shows.

3 Cross-correlation Function

3.1 investigating the cross-correlation function CCF

As with the previous example, we consider two sequences x and y related by a MA(9) process

$$y[n] = x[n] + x[n-1] + \dots + x[n-8]$$

the cross-correlation function of x and y can be found using the MATLAB command:

```
N=1000; %using 1000 samples
x = randn([1,N]); %generate 1000 GWN
y = filter(ones(9,1),[1],x); %pass input through the MA9 filter
ccf=xcorr(x,y,'unbiased'); %find the ccf
tau=-999:1:999; % generate the xaxis label
stem(tau,ccf); axis([-20 20 -1 3])
```

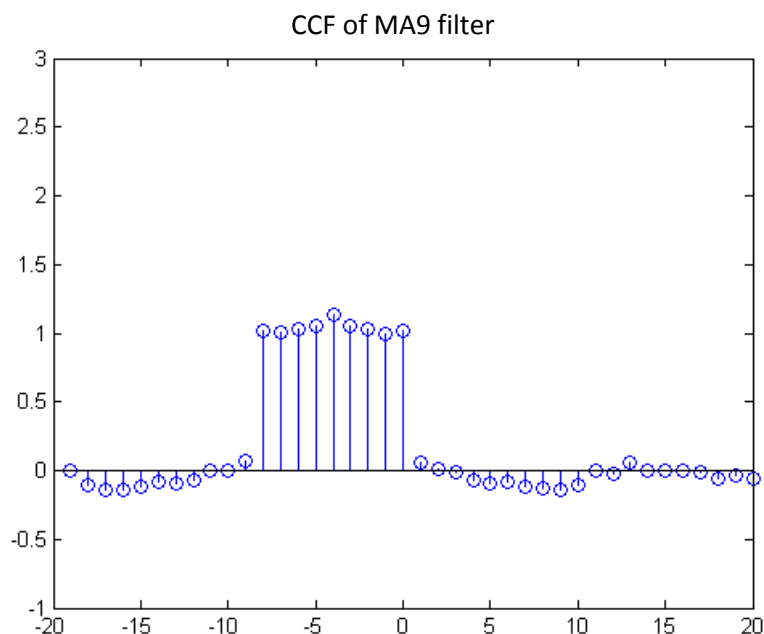


Fig 6. Cross-correlation of x and y

The cross correlation of x and y appears to be a gate function. This is because the cross-correlation function of x and y is related to the autocorrelation function of x by the following expression.

$$R_{xy}(\tau) = h(\tau) * R_x(\tau)$$

Since x represents an uncorrelated stochastic process its correlation would appear to be a dirac function as shown in the section before. The impulse response can be found by the following MATLAB command:

```
x=zeros(1,50);
x(1)=1; %generate an impulse for the input
y=filter(ones(1,9),1,x); %find the impulse response for the MA9 filter
```

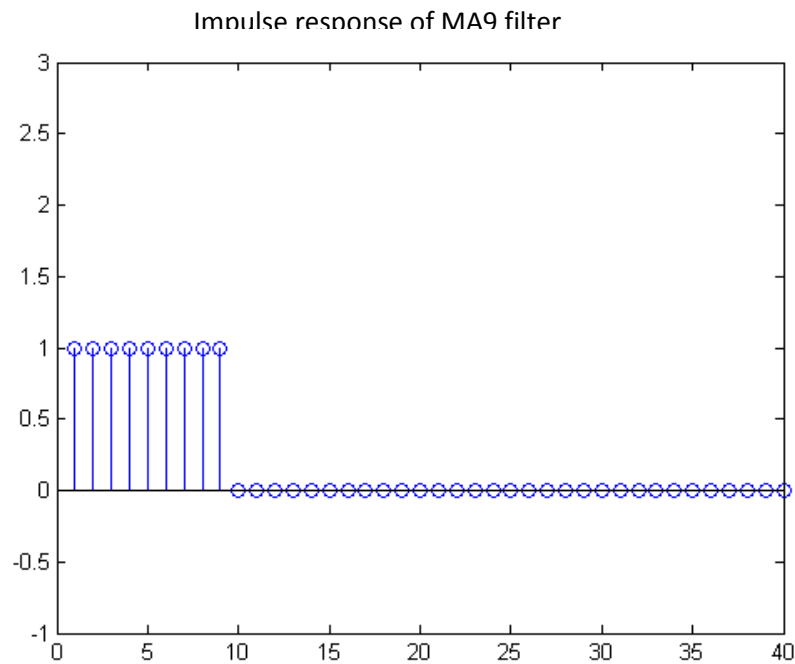


Fig 7. Impulse response of the MA(9) filter

Hence by convoluting a dirac function with the above impulse response we would expect the output $R_{xy}(\tau)$ to be a mirror image of the impulse response of the MA(9) filter. (Exactly as what we have observed in figure 6)

3.2 System identification

From the above observation, we can tell that if the input of the filter is an uncorrelated stochastic process, we could identify the system information from the cross-correlation function between the input and the output. The order of the filter can be identified by counting the number of non-zero terms generated by the cross-correlation function. (i.e. identifying the point of cut-off on the cross-correlation function). For example we can use this technique to identify that the filter order used on the left is 3 and on the right is 7.

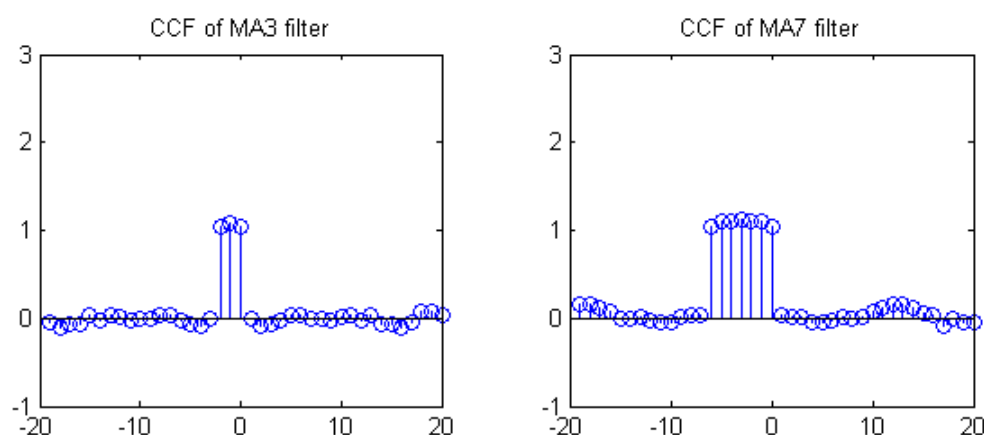


Fig 8. Demonstration of System Identification

4 Autoregressive Modelling

4.1 ACF of sunspot data

The MATLAB inbuilt sunspot time series was loaded to the program and the ACF was calculated for data length $N=5, 20, 250$

```
load sunspot.dat;
data1 = sunspot(1:5,2);
subplot(3,1,1);
acf1 = xcorr(data1,'unbiased');
plot(-4:1:4,acf1);title('sunspot acf N = 5');

subplot(3,1,2);
data2 = sunspot(1:20,2);
acf2 = xcorr(data2,'unbiased');
plot(-19:1:19,acf2);title('sunspot acf N = 20');

subplot(3,1,3);
data3 = sunspot(1:250,2);
acf3 = xcorr(data3,'unbiased');
plot(-249:1:249,acf3);title('sunspot acf N = 250');
```

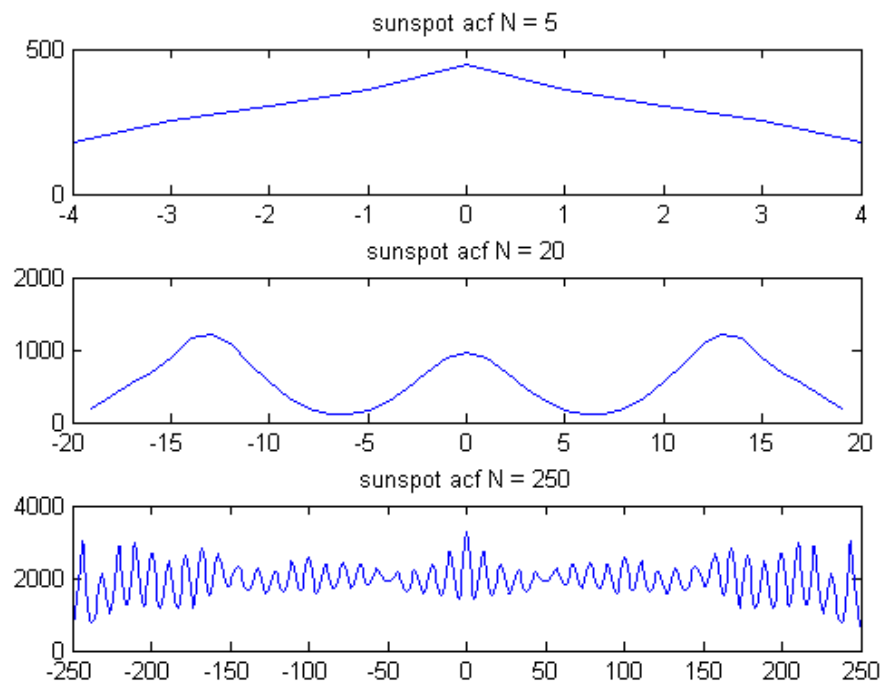


Fig 9.ACF of sun spot using $N = 5, 20, 250$

The first thing to note is the symmetry in the ACF in three of the plots. The main difference we can observe from the three plots is that in the first plot the ACF decreases as $|N|$ increases and there exists no periodic patterns whereas in the other two plots, as the number of samples increases we can clearly observe the periodic nature of the ACF.

4.2 ACF of zero mean sunspot data

```
load sunspot.dat;
data1 = sunspot(1:5,2);
sunspot_mean = mean(sunspot);
sunspot_mean = sunspot_mean(1,2);
subplot(3,1,1);
acf1 = xcorr(data1-sunspot_mean, 'unbiased');
plot(-4:1:4,acf1);title('sunspot acf N = 5');
```

The ACF for the zero mean sunspot data for $N = 5, 20, 250$ is as follows

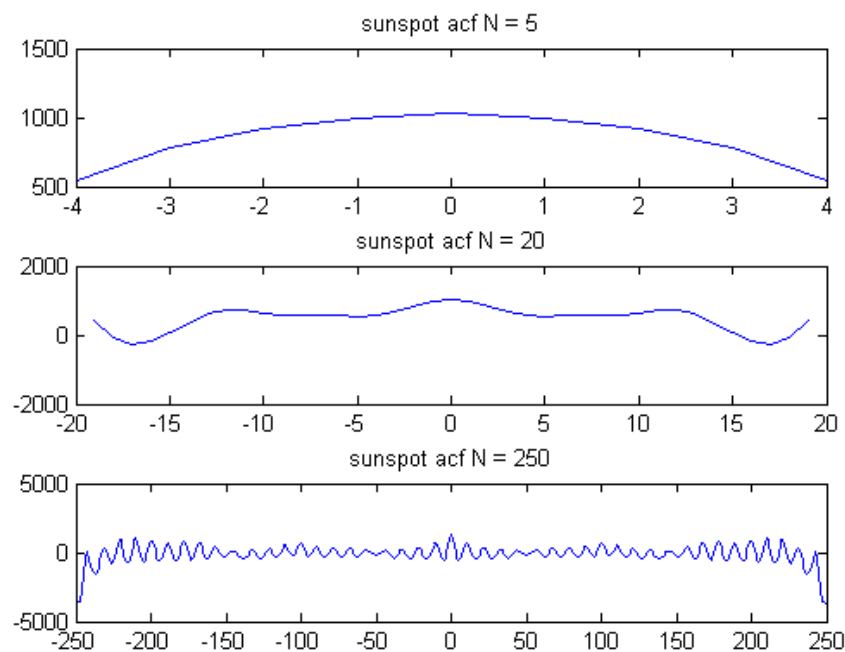


Fig 10.ACF of zero mean sun spot data using $N = 5, 20, 250$

By taking the zero mean of the sunspot data and plotting its ACF we can see that for a smaller number of samples, little information can be observed from its autocorrelation function. This also applies for the $N = 20$ case. In the previous section, we can observe the periodic nature of the ACF of the sunspot time series using $N = 20$. However, while using the zero mean sunspot data, such periodic nature is no longer apparent. In the final case where $N = 250$, despite the fact that we are using the zero mean sunspot data, we can still observe the periodic nature of the autocorrelation function.

4.3 Stability of AR(2) process

Two random uniformly distributed variable a_1 and a_2 where $a_1 \sim U(-2.5, 2.5)$ $a_2 \sim U(-1.5, 1.5)$ and generated and used as the coefficient of an AR(2) process.

$$x[n] = a_1 x[n-1] + a_2 x[n-2] + w[n] \quad w[n] \sim N(0,1)$$

By designing a MATLAB script the divergence of the AR(2) process is investigated. If the AR(2) process diverges for a particular coefficient then a '*' is mark on the output graph which correspond to the position of the coefficient of a_1 and a_2 .

```
x(1)=0; % initialization
x(2)=0;
N = 10000; % test for N pairs of a1 and a2
a1 = 5*(rand([1,N])-0.5); % a1 = U~(-2.5,2.5)
a2 = 3*(rand([1,N])-0.5); % a2 = U~(-1.5,1.5)
for i=1:N

    coeff1 = a1(i); %take a pair
    coeff2 = a2(i);
    for j=3:200
        %AR2 process
        x(j)=coeff1*x(j-1)+coeff2*x(j-2)+randn(1);
    end
    if abs(x(200))>300
        %plot * if diverges
        plot(coeff1,coeff2,'*'); hold on
    end

end
title('Coefficients for which the AR2 Process Diverges')
xlabel('a1');
ylabel('a2');

hold off
```

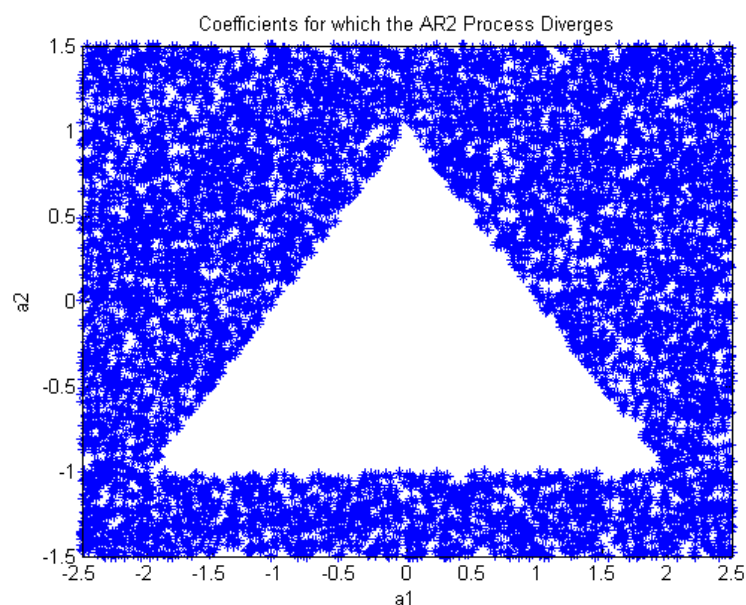


Fig 11. Coefficients for which the AR(2) process diverges

The stability of the AR process is pretty tough to determine, but in the very general sense. If an AR process is stable, the process should converge for a large number of samples. Hence the criterion for determining whether the AR process diverges is by checking if the final output of the AR process exceeds a certain value. `[if abs(x(200)) > 300]`

We observe that there exist a triangle on the graph where the mark '*' is absent representing that if the coefficients a_1 and a_2 lie in this area, then the AR(2) system would be stable. The triangle can be described by the equations $a_1 + a_2 < 1$, $a_2 - a_1 < 1$ and $-1 < a_2 < 1$. From this observation we can say that in order for the stability of the AR(2) process to be preserved the coefficients a_1 and a_2 must satisfy the above conditions.

4.4 PAC using Yule-Walker equations

From the Yule-Walker equations, we know that the coefficient of the Partial Correlation Function (PAC) is related to the autocorrelation functions by the following relationship:

$$a_{11} = \rho_1, \quad a_{22} = \frac{\rho_2 - \rho_1^2}{1 - \rho_1^2}, \quad a_{33} = \frac{\begin{vmatrix} 1 & \rho_1 & \rho_1 \\ \rho_1 & 1 & \rho_2 \\ \rho_2 & \rho_1 & \rho_3 \end{vmatrix}}{\begin{vmatrix} 1 & \rho_1 & \rho_2 \\ \rho_1 & 1 & \rho_1 \\ \rho_2 & \rho_1 & 1 \end{vmatrix}}, \quad \text{etc}$$

Fig 11. Partial Correlation function determined using Yule-Walker equations¹

So using MATLAB we can calculate the PAC of the sunspot.dat data set to determine the order of the sunspot time series.

Partial Autocorrelation Function	Value
a_{11}	0.8219
a_{22}	-0.6853
a_{33}	-0.1146
a_{44}	0.0515
a_{55}	-0.0171
a_{66}	0.1666
a_{77}	0.2126
a_{88}	0.2336
a_{99}	0.2098
a_{1010}	0.0262

Table 1. PAC of the sunspot time series

From the definition of the an AR(p) process, the PAC terms will be nonzero for k smaller than the order of the AR process and zero otherwise. Hence, from the table we can observe that the order of the sunspot time series is most likely to be 2 as the first two terms are large whereas the remainder terms are small.

¹ Danilo Mandic, Advanced Signal Processing: Linear Stochastic Models 2013

This result can be verified by using the inbuilt `parcorr` function in MATLAB. The blue line on the graph represents the confidence interval. The graph confirms that the time series is most likely to be an AR(2) process.

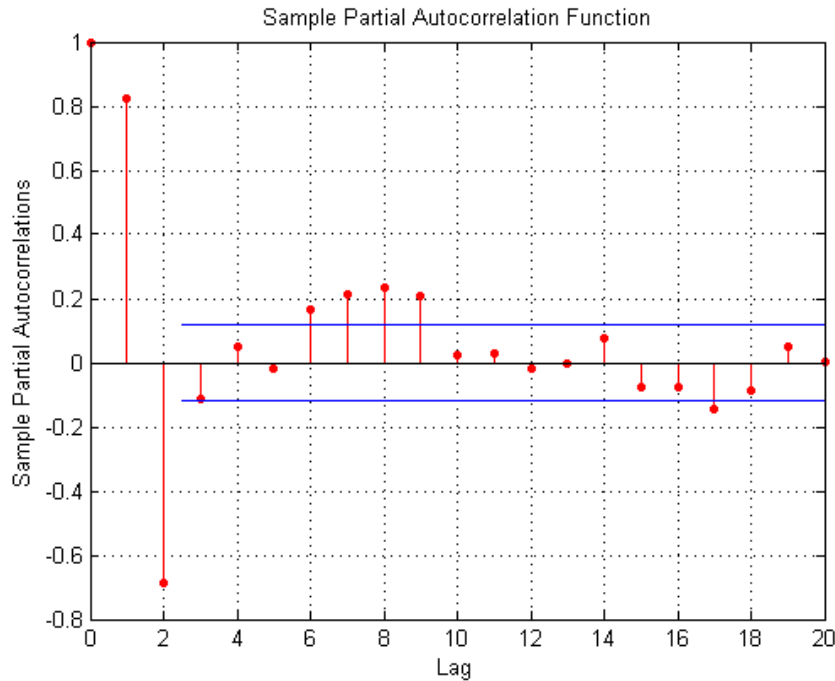


Fig 12. PAC of the sunspot time series generated using the `parcorr` function

4.5 Order estimation using MDL and AIC

We now use two other method to estimate the order of the AR process. The MDL stands for the minimum description length which is defined as:

$$\text{MDL} = \log E + \frac{p \log N}{N}$$

The AIC stands for the Akaike Information Criterion:

$$\text{AIC} = \log E + \frac{2p}{N}$$

The MDL and AIC functions will provide us with a curve demonstrating the trade-off between the loss function and the model order. For both of these functions, the $\log E$ part assesses the quality of the two models and the second term serves as a penalty term which causes the curve to grow as the order is increase. This feature ensures that we will obtain the lowest possible estimate for the order of the AR process.

The loss function used in this case is the cumulative squared error is as follows:

$$E[n] = \sum_{k=0}^n (\hat{x}[k] - x[k])^2, n = 1, 2, \dots, N.$$

And for an AR(p) process the first-order predictor is given by

$$\hat{x}[n] = E\{x[n]\} = a_1 x[n-1] + \dots + a_p x[n-p]$$

The coefficients of the AR process can be found by the `aryule` function on MATLAB

```
load sunspot.dat
data1 = sunspot(:,2);
a = aryule(data1,p);
```

Having this knowledge, we can now compute the cumulative squared error hence the MLD and AIC using the following MATLAB command (only MDL is shown)

```
load sunspot.dat
data1 = sunspot(:,2);
N = length(data1);
for p=1:10 %calculate the MDL
    a = aryule(data1,p);
    data1_estimate = filter(-1*a,1,data1); %find the error from
    mean
    E = (data1_estimate - data1).*(data1_estimate - data1);
    E = sum(E); % this is the cumulative squared
    mean error
    loss(p) = log(E)/log(10);
    MDL(p) = log(E)/log(10) + p * log(N)/(log(10)*N);
end;
figure(1);
plot(1:10,MDL,'b');hold on;title('MDL for Sunspot.dat');
xlabel('p');
plot(1:10,loss,'r');hold off;
```

The AIC and MDL for the sunspot time series is plotted,, from both of the graphs we can observe that the minima occurs at $p=2$, again confirming that the sunspot time series is an AR(2) process. This result is consistent with the result obtained from section 4.4.

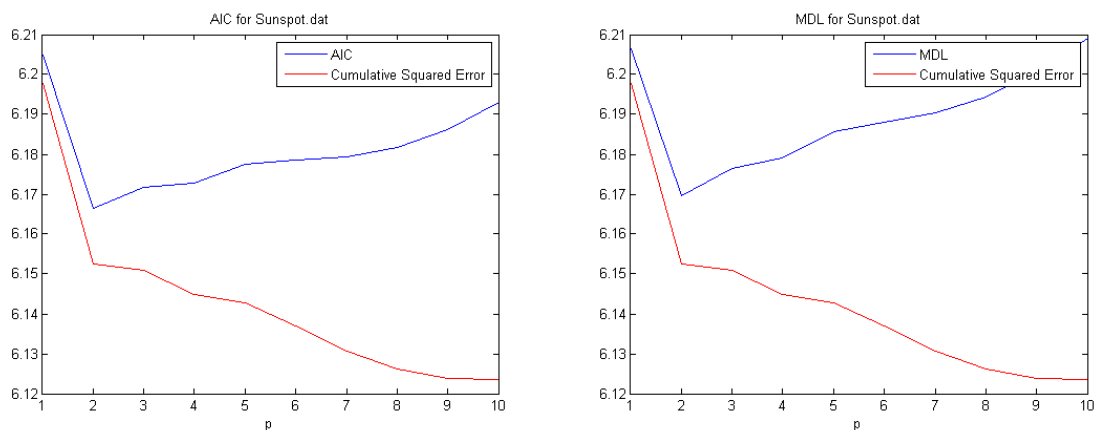


Fig 13. Order Estimation using MDL and AIC

4.6 Future prediction using the AR process

An AR model can be used for predicting m steps ahead using the following equations:

$$\hat{x}[n+m] = a_1x[n-1] + \dots + a_px[n-p]$$

This is implemented in MATLAB using the following commands:

```

clear all;
N = 10;                                %number of steps ahead
x(1) = 5;
x(2) = 11;
x(3) = 16;
x(4) = 23;
x(5) = 36;
x(6) = 58;
load sunspot.dat
subplot(2,1,1);
data = sunspot(:,2);
a = aryule(data,6);
for index=7:6+N
    x(index) = a(2)*x(index-1) + a(3)*x(index-2) +
a(4)*x(index-3) + ...
    a(5)*x(index-4) + a(6)*x(index-5) + a(7)*x(index-6);
end;
plot(x);hold on;
plot(data(1:6+N), 'r');hold off;
title('prediction using AR 6');
legend('Predicted sunspot', 'Actual sunspot');

x1(1) = 5;
x1(2) = 11;
x1(3) = 16;
x1(4) = 23;
x1(5) = 36;
x1(6) = 58;
load sunspot.dat
subplot(2,1,2);
data1 = sunspot(:,2);
a = aryule(data1,2);
for index=7:6+N
    x1(index) = a(2)*x1(index-1) + a(3)*x1(index-2);
end;
plot(x1);hold on;
plot(data1(1:6+N), 'r');hold off;
title('prediction using AR 2');
legend('Predicted sunspot', 'Actual sunspot');

```

An AR(6) and AR(2) model will be used to make the prediction.

For 1 step ahead:

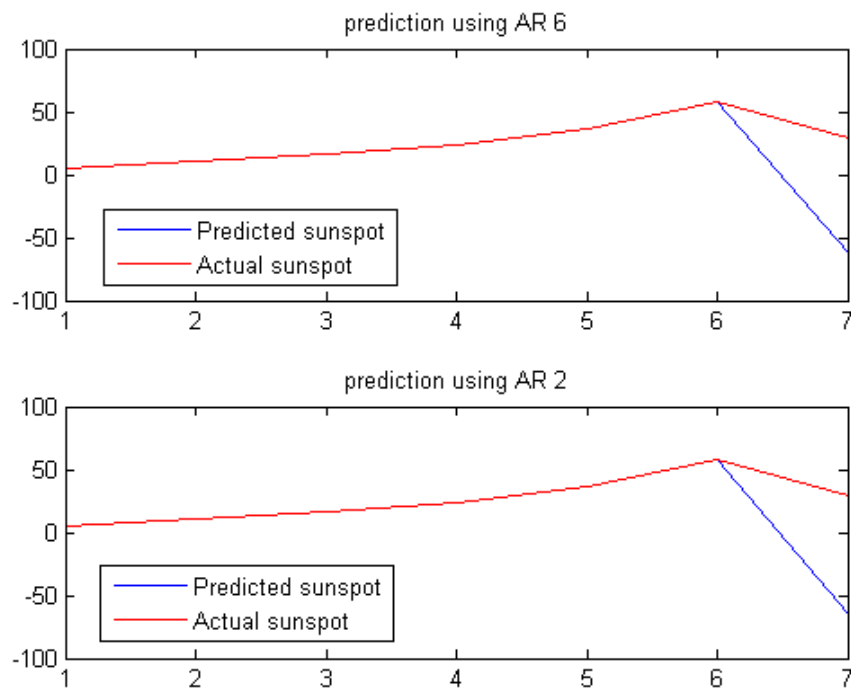


Fig 14. Using AR model to predict 1 step ahead

For 2 steps ahead:

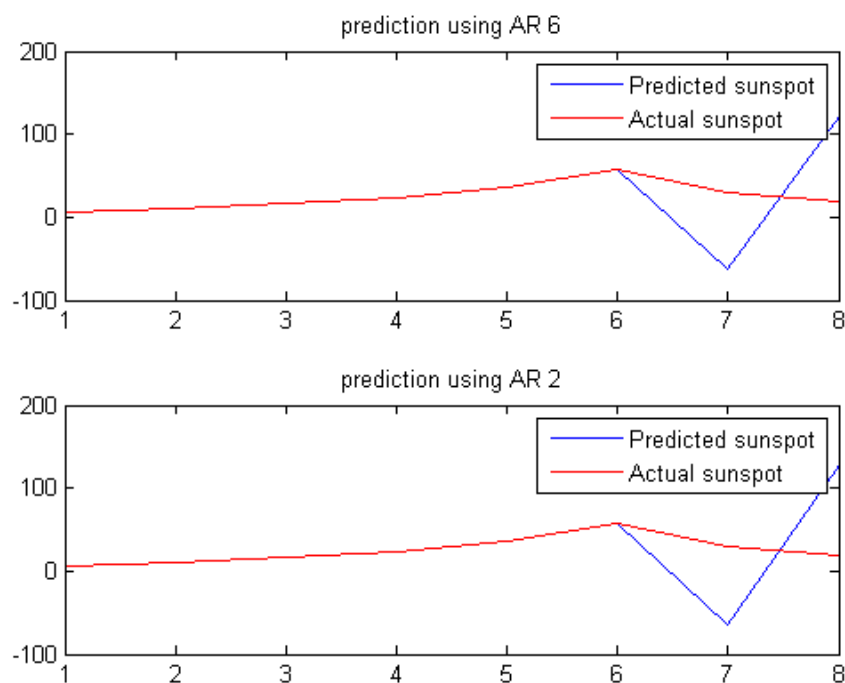


Fig 15. Using AR model to predict 2 steps ahead

For 5 steps ahead:

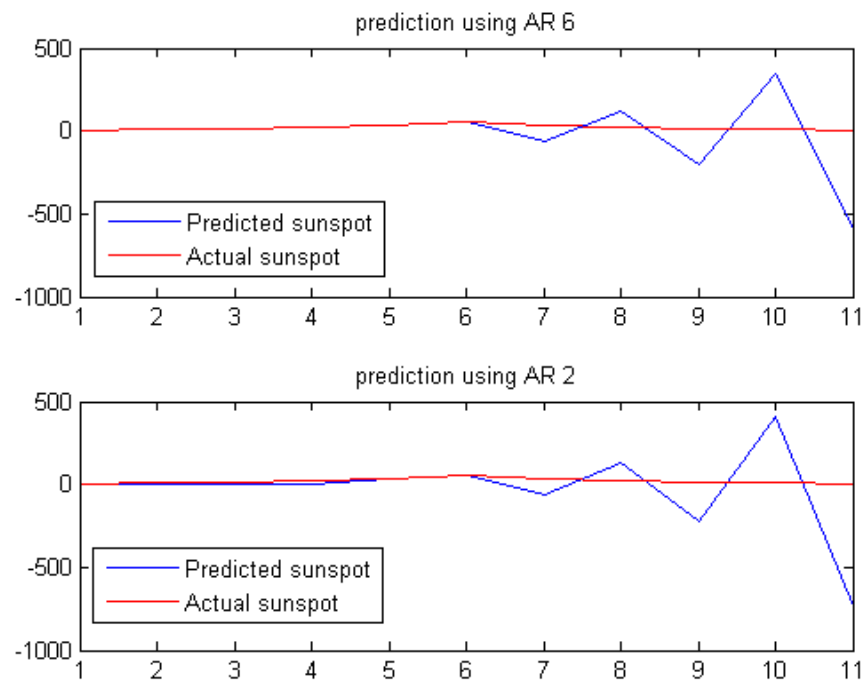


Fig 16. Using AR model to predict 5 steps ahead

For 10 steps ahead:

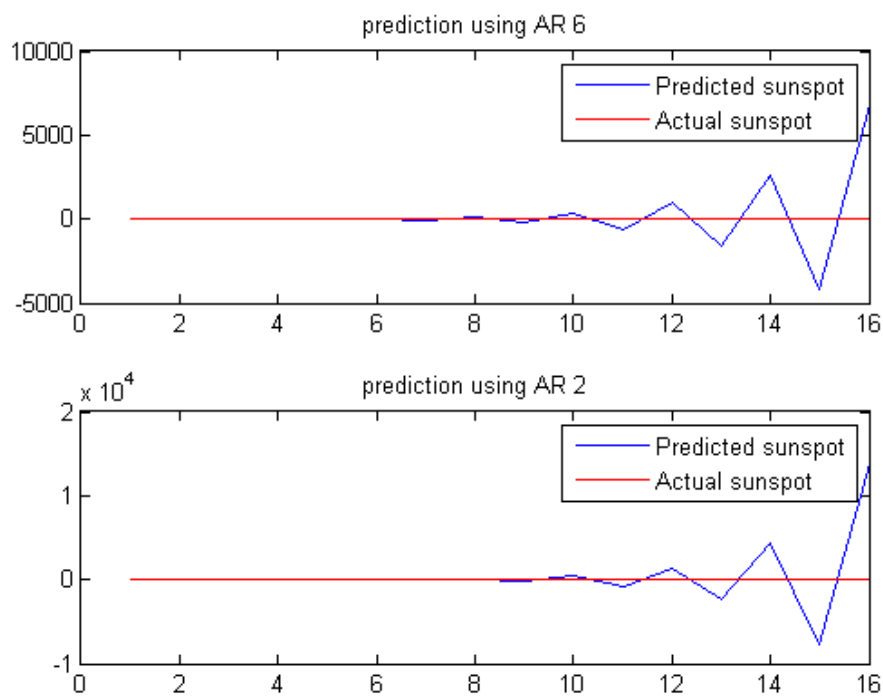


Fig 17. Using AR model to predict 10 steps ahead

From the graphs above we can see that for a small number of steps, both the AR(2) and AR(6) models can in general track the shape of sunspot time series, but as the number of steps increase, the tracking becomes less good. When the step reaches 10, however, we can see that the prediction error becomes very significant for both of the models. In general the AR(6) model does a better job to track the original sunspot time series when compared to the AR(2) model (has a smaller prediction error).

The prediction errors are resulted from the following reasons. Since the ARYULE function is an estimate of the AR coefficients, we are uncertain of whether the AR model we are using is the correct model. In addition, we will be using forecasted values in the right hand side of the equation to predict future values, but since these predicted values have errors (and we are uncertain about the errors). These errors will be compounded as the number of steps increase; hence for a large number of steps, the prediction error becomes increasingly significant.

Advanced Signal Processing (ASP)

Coursework 3

Name: Di Wu

CID: 00646091

Login: dw1310

Declaration: I confirm that this submission is my own work. In it, I give references and citations whenever I refer to or use the published, or unpublished, work of others. I am aware that this course is bound by penalties as set out in the College examination offenses policy.

Signed: Di Wu

Aim: The aim of the third coursework is to introduce us to the Power Spectral Density of a random signal. We will study classical methods for PSD estimation and some model based methods for PSD estimation. Finally, the coursework will investigate the a practical application of spectrum estimation with the dial pad example.

0. Developing a Periodogram Function

The first assignment is to develop a periodogram function which can be used to estimate the PSD of an ergodic stochastic process on MATLAB using the relationship:

$$\hat{P}_x(f) = \frac{1}{N} \left| \sum_{n=0}^{N-1} x[n] e^{-j2\pi f \frac{n}{N}} \right|^2$$

Where $x[n]$ is an input sequence and f represents the normalized frequency $f = 0, 1/N, \dots, (N-1)/N$. By referring to page 394 of the recommended text book ‘statistical digital signal processing and modelling’ the MATLAB command for generating the periodogram can be seen as follows:

```
function [ p ] = pgm( x )
%ASPLAB30 Periodogram computation function
accumulator = 0;
N = length(x);      %find the number of samples
f = (0:1:N-1)/N;    %find the normalized frequencies
p = (abs(fft(x)).^2)/N; %summation replaced by a
fourier transform
plot(f,p); title ('Estimated PSD');
xlabel('Normalized Frequency');
ylabel('Power/Frequency');
end
```

From which we obtain the following periodograms:

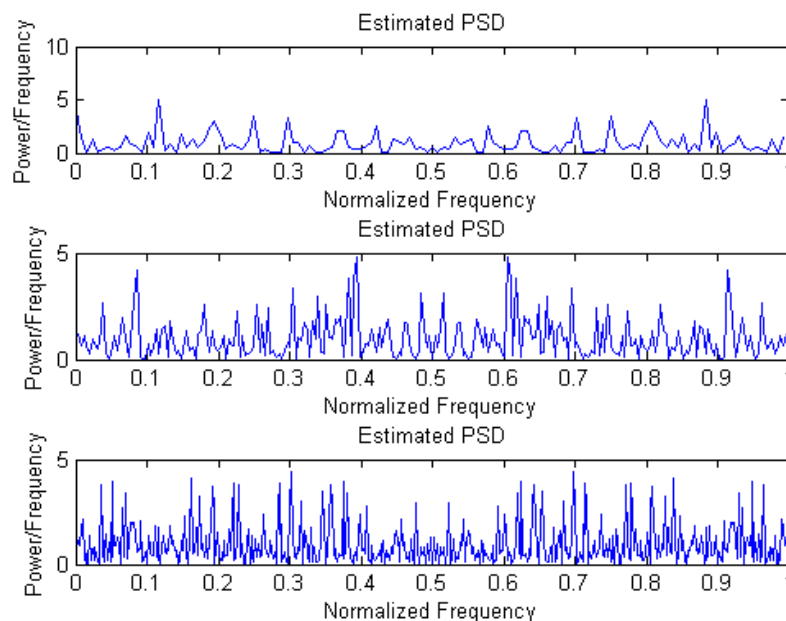


Fig 1. Periodogram of Gaussian process

From the graphs we can observe that as the number of random samples N is generated the resolution of the frequency axis of the periodogram is increased. As the number of samples N is increased, the PSD resembles more to the ideal PSD of white noise. (i.e. unity for all frequencies)

1. Average Periodogram Estimates

1.1 Smoothing the PSD estimate

The PSD needs to be smoothened in order for the PSD to better resemble the ideal PSD of the white noise. To do this we implement a FIR filter with impulse response $0.2*[1 \ 1 \ 1 \ 1 \ 1]$ using the matlab `filtfilt` function.

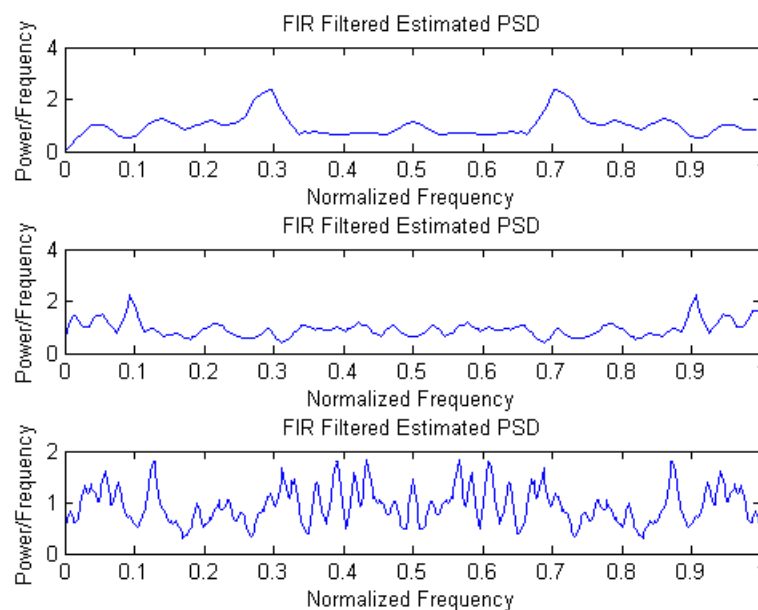


Fig 2. Smoothened periodogram of Gaussian process

From the output graphs we can observe that the PSD of the Gaussian process has been improved by the FIR filter.

1.2 PSD of 1024 samples divided into 128 sample segments

We generated 1024 samples using the `randn` function and divided the 1024 into 8 parts to generate their individual PSD. We can observe from the graphs plotted on the next page that the variation of 8 individual estimates are very similar.

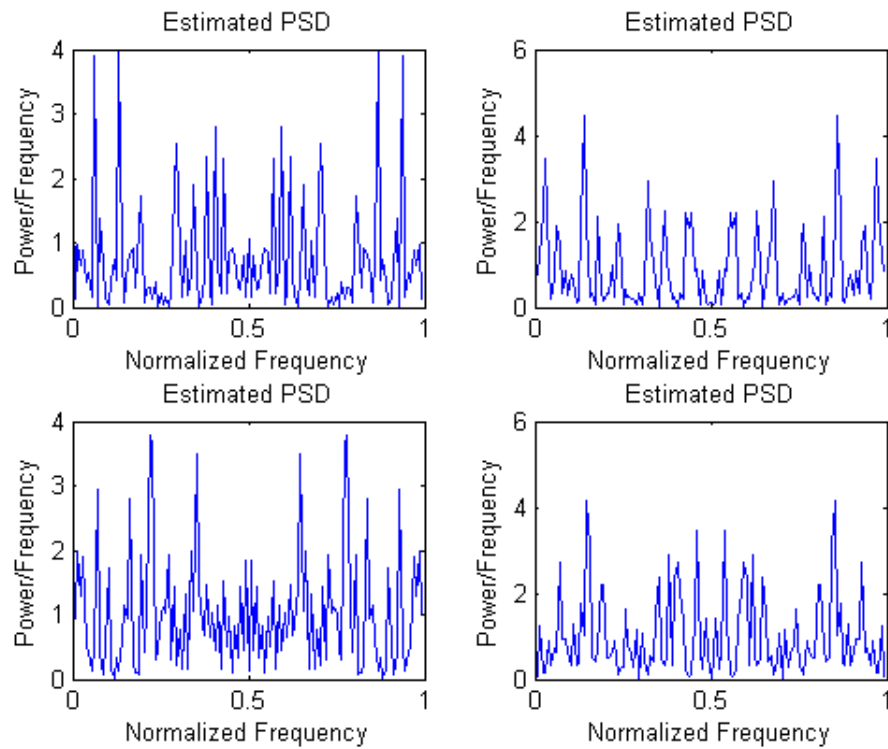


Fig 3. PSD of 1024 samples of the Gaussian process segmented into 8 parts

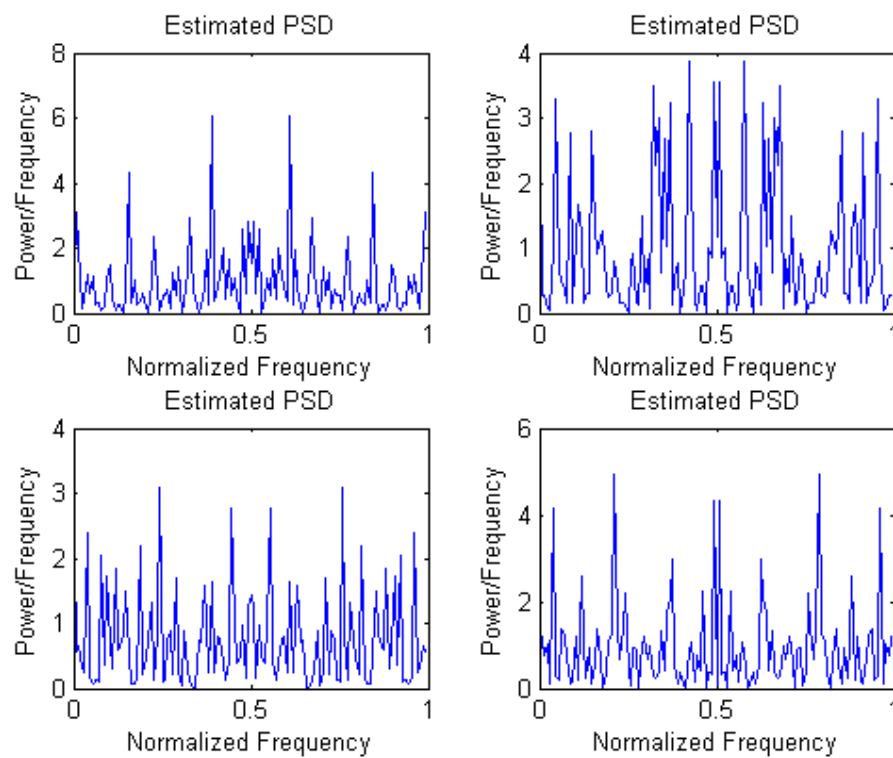


Fig 4. PSD of 1024 samples of the Gaussian process segmented into 8 parts

1.3 Averaged Periodogram

The eight estimated PSDs generated in the previous section can be averaged to produce a better estimate of the PSD:

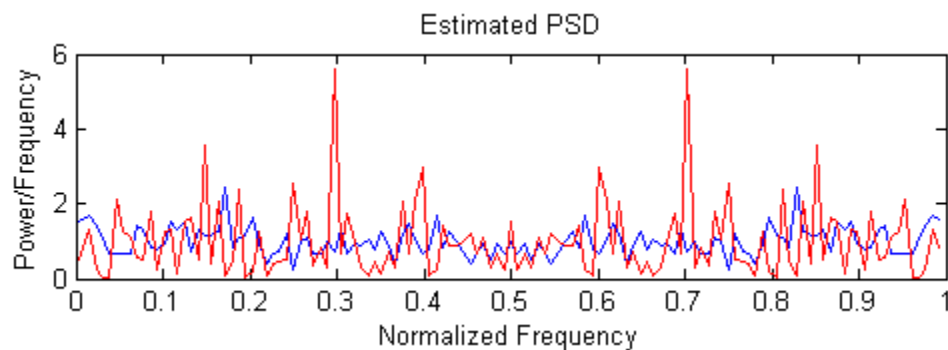


Fig 5. The Averaged Periodogram of 1024 samples divided into 8 segments

We can see that after averaging the periodogram the variation of the averaged PSD (plotted in blue) is much less compared to the periodogram we have generated in previous sections (plotted in red). According to the lab notes, the variation is reduced by a factor of K where K is the number of individual non-overlapping segments.

2. Spectrum of autoregressive process

First a 1064 sample sequence (x) is generated by `randn` and passed through a first order AR filter. The input and output (sequence y) are shown as below:

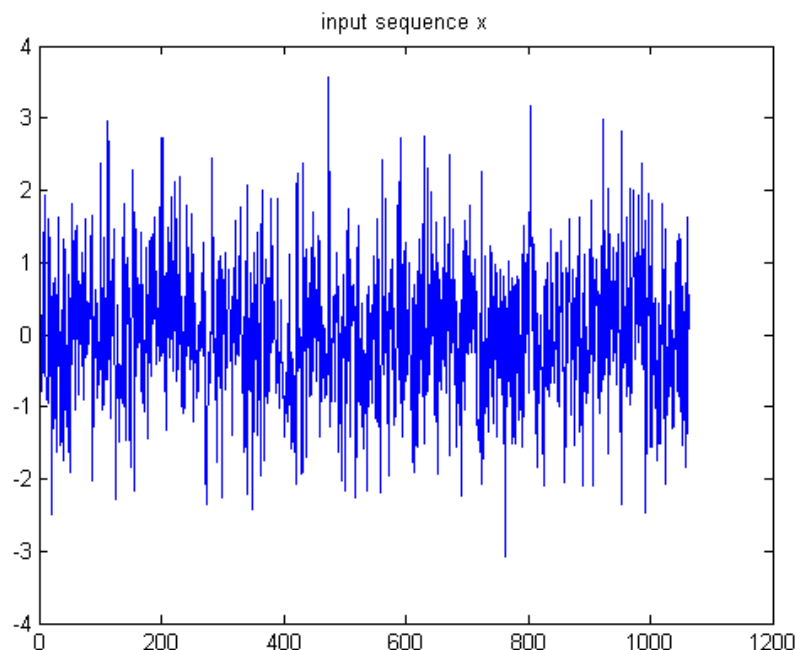


Fig 6. Plot of 1064 sample generated by the Gaussian process

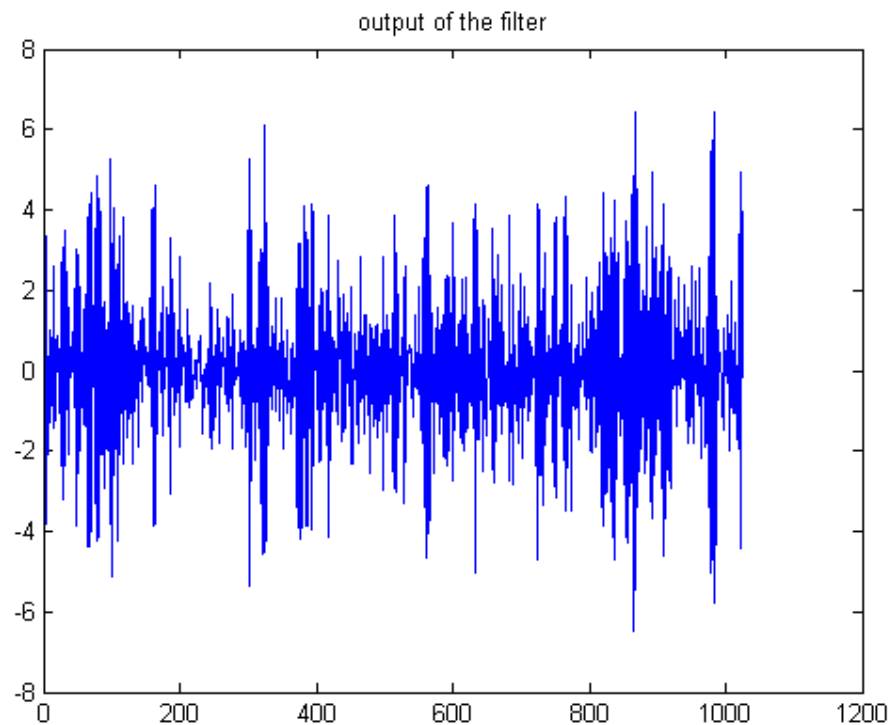


Fig 7. Output of the AR(1) filter

We can see that the variation of the output increased when comparing to the input.

2.1 Frequency response of AR(1) filter

Using the inbuilt `freqz` function in MATLAB, we can calculate the frequency response of the AR1 filter form which we can plot the PSD of the AR(1) filtered signal:

$$y[n] = x[n] - 0.9*y[n-1]$$

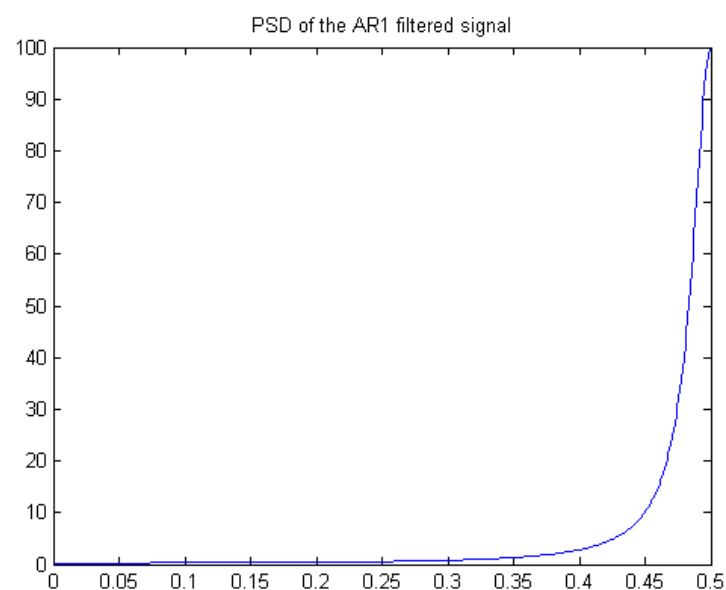


Fig 8. PSD of the AR1 filtered signal

The PSD of the AR1 filter is given by

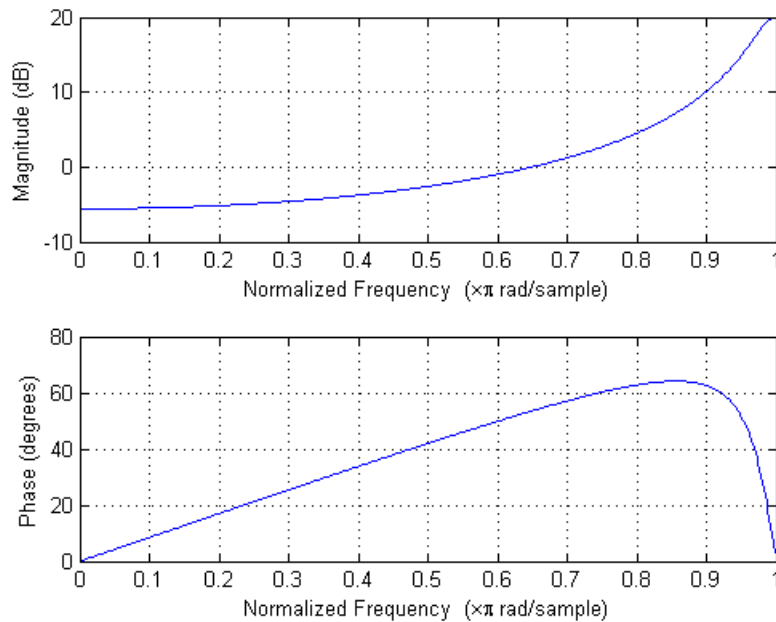
$$P_Y(f) = \left| \frac{1}{1 + 0.9e^{-j2\pi kf}} \right|^2 = \frac{1}{1 + 0.9^2 + 2 \cdot 0.9 \cos(2\pi f)}$$

For $0 < f < 0.5$ the cosine term of the function is decreasing hence this AR1 process represents a high pass filter. The shape of PSD generated adheres to this (i.e. PSD increases for large frequencies).

The cut off frequency of the AR1 filter can be investigated by plotting its frequency response and observing the transfer function.

$$P_Y(f) = \left| \frac{1}{1 + 0.9e^{-j2\pi kf}} \right|^2 = \frac{1}{1 + 0.9^2 + 2 \cdot 0.9 \cos(2\pi f)}$$

We can see that for normalized frequencies in the range of 0 to 0.5 the denominator will not equal to zero, hence indicating that there is no pole in the transfer function. By looking at the frequency response below, it can be concluded that the AR1 filter does not strictly have a 3dB cut-off frequency.



2.2 Comparison between PSD generated by the periodogram and the freqz function

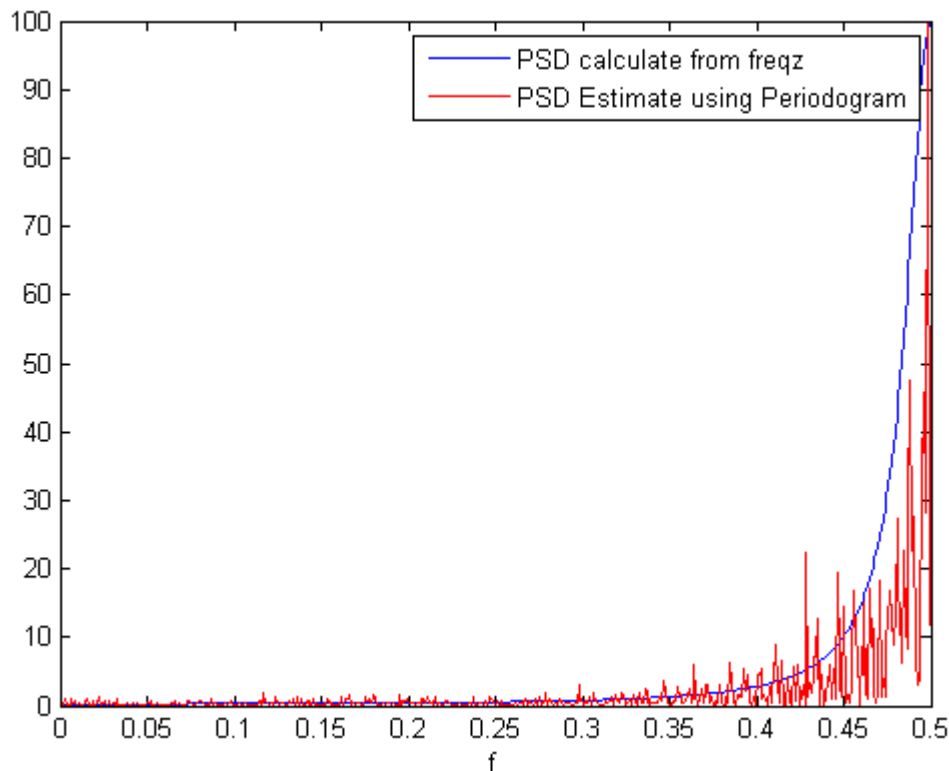
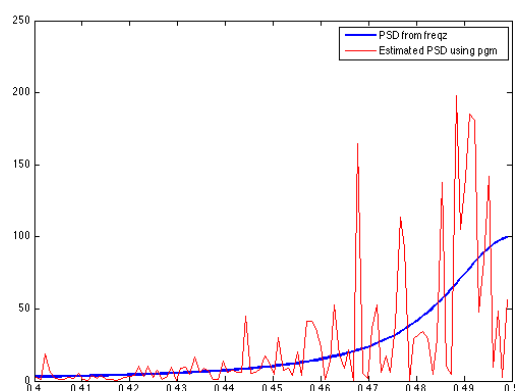


Fig 9. Comparison between PSD generated by the periodogram and the freqz function

We can see that the PSD estimated using the periodogram follows the PSD calculated from freqz pretty well for $f < 0.4$. After this point the estimated PSD deviates significantly.

2.3 Role of the Rectangular Window



Zooming into the interval of $f = [0.4 \ 0.5]$, we can see that the PSD estimated by the periodogram does not follow the PSD generated by the freqz function very well. The underlying rectangular window used in the periodogram estimator, is required as we only have a limited number of samples. However, as we can see that due to the sharp cut off of the rectangular window, some distortions have been introduced in the interval represented in the graph.

2.4 Model based PSD

The PSD can also be calculated by model based mean since the to calculate the PSD we only need to know parameters ($\hat{\sigma}_x^2$). These two quantities can be estimated using the autocorrelation function.

$$\hat{a}_1 = -\hat{R}_Y(1) / \hat{R}_Y(0)$$

$$\hat{\sigma}_x^2 = \hat{R}_Y(0) + \hat{a}_1 \hat{R}_Y(1)$$

The ACF is calculated as follows:

```
x = randn(1064,1); %generate the random samples
y = filter([1],[1 0.9],x); %filter
acf = xcorr(y); %calculate the ACF
tau = -(length(y)-1):1:(length(y)-1); %xaxis label
stem(tau,acf/max(acf));title('ACF of the AR1 Process');
xlabel('TAU');ylabel('Normalized ACF');
axis([-20 20 -1 1]); %zoomed in
```

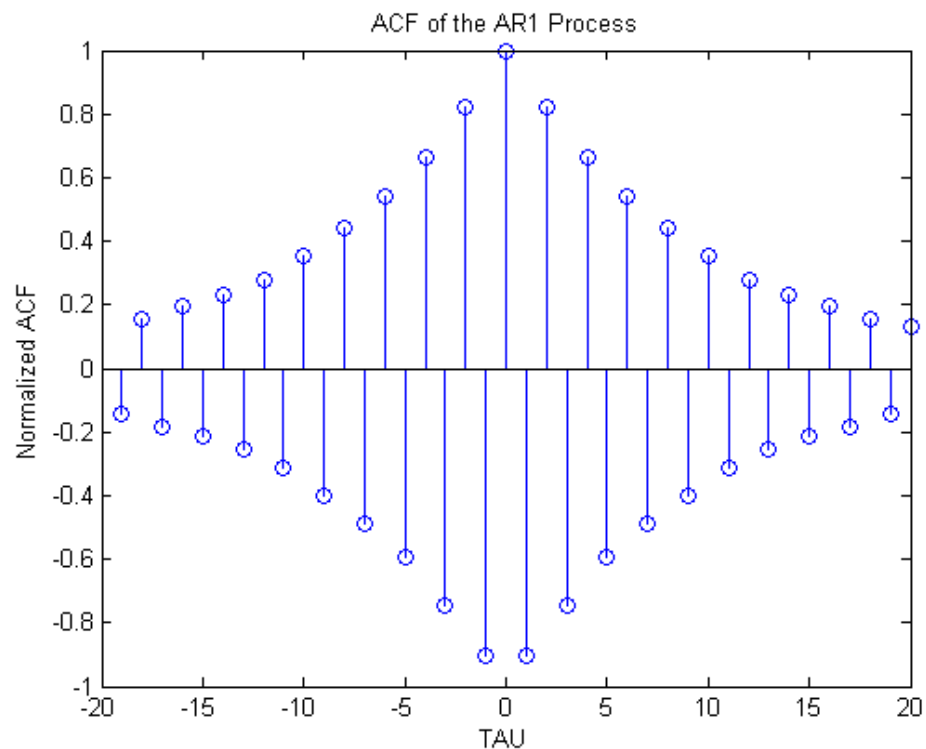


Fig 10. Autocorrelation function of the AR1 process

From the graph above we can calculate

$$\hat{a}_1 = \frac{0.9052}{1} = 0.9052$$

$$\hat{\sigma}_x^2 = 1 - 0.9052 \cdot 0.9052 = 0.1806$$

```
a = 0.9052;           %a1
variance = 0.1806;    %variance
index = 1;
for f=0:0.01:0.5      %from f=0 to f=0.5
    p(index) = variance/(abs(1+a*exp(-1i*2*pi*f))^2);
    index=index+1;
end;
plot(0:0.01:0.5,p);title('Model based PSD');
xlabel('f');
```

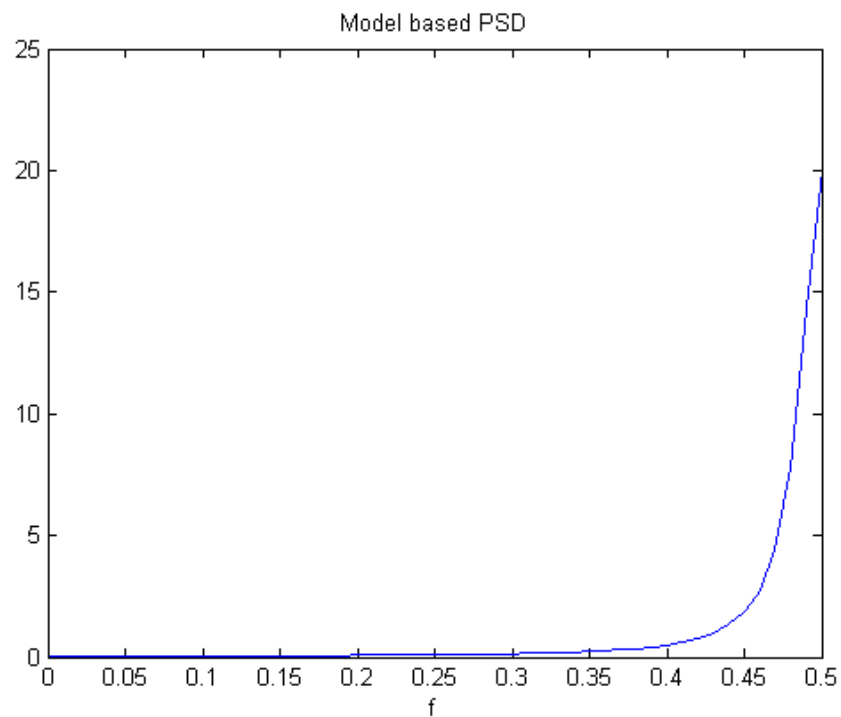


Fig 11. Model based PSD estimate

Consistent with the PSD plotted before, the process represents a high pass filter.

2.5 Comparison of PSDs

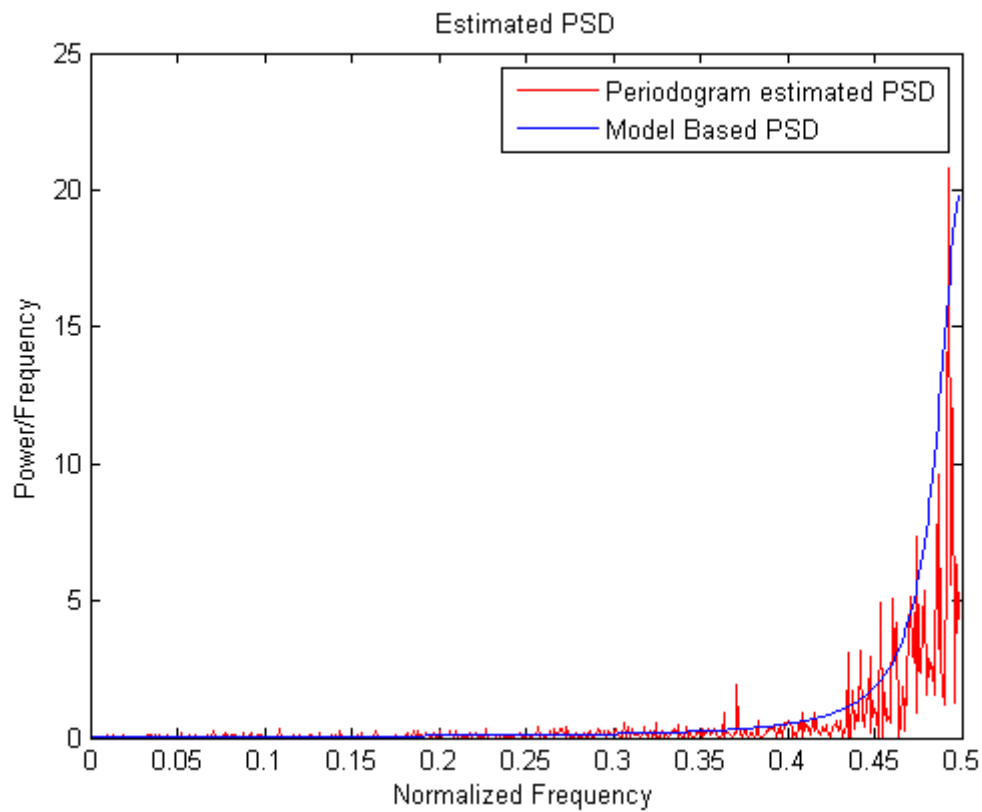


Fig 11. Comparison of Periodogram generated PSD and the Model Based PSD

As the above graph shows, the result is similar to the one obtained from the periodogram estimate both showing that the AR filter is a high pass filter. One thing to note is that when using the filter function, it is necessary to incorporate the variance of the model.

```
y = filter([1],[1 0.9],sqrt(variance)*x);
```

3. Spectrogram example: dial tone pad

Now we investigate an everyday life example of the dial tone pad. The underlying concept of a touch tone telephone is the Dual Tone Multi-Frequency (DTMF) system for every button on the telephone two sinusoids of different frequencies is assigned.

	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

Fig 12. Dial pad frequencies

3.1 Two-tone signal for a randomly generated telephone number

A random London landline number is generated using the rand function in MATLAB and the two-tone signal for this number is calculated. Note that a London landline number have 11 digits and we assume that each digit will be pressed for 0.25 second followed by an idle time of 0.25 second. So the total dialing time would be 5.25 seconds. This is implemented in MATLAB as follows:

```
% a frequency table will be used for looking up the two tone frequencies.
freq_table = [1336 1209 1336 1477 1209 1336 1477 1209 1336 1477 ...
    1209 1477; 941 697 697 697 770 770 770 852 852 941 941];
fs = 32768; % sample rate
ts = 1/fs; % sample period
N = 5.25/ts; % total number of samples
t = (0:N-1)*ts; % Time vector
tel(1)=0; tel(2)=2; tel(3)=0; % first three digits 020
for index=4:11
    tel(index) = round(rand(1)*10); % next 8 digits
end;
count = 1;
for index=1:11
    for index2=1:8192 %generate the two-tone signal
        y(count) = sin(2*pi*freq_table(1,tel(index)+1)*t(index2)) ...
            + sin(2*pi*freq_table(2,tel(index)+1)*t(index2));
        count = count + 1;
    end;
    if index ~= 11 %generate the idle time
        for index2=1:8192
            y(count) = 0;
            count = count + 1;
        end;
    end;
end;
plot(0:5.25/N:5.25-(1/N),y);
title('two-tone signal within the DTMF system for a telephone number');
xlabel('time seconds');
axis([0 0.015 -2 2]);
```

The corresponding two tone output for the randomly generated telephone number is as follows:

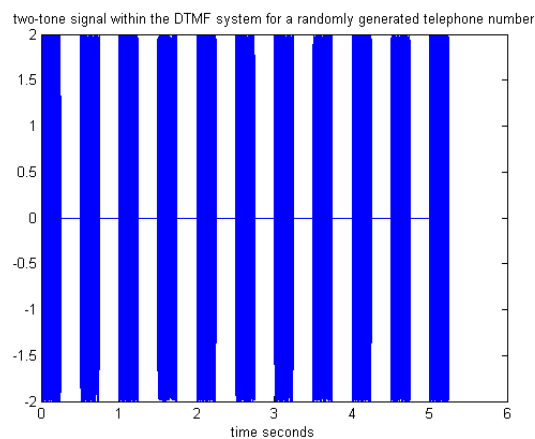


Fig 13. Two tone signal for a randomly generated telephone number

By zooming in we can plot y for two different digits 0 and 2

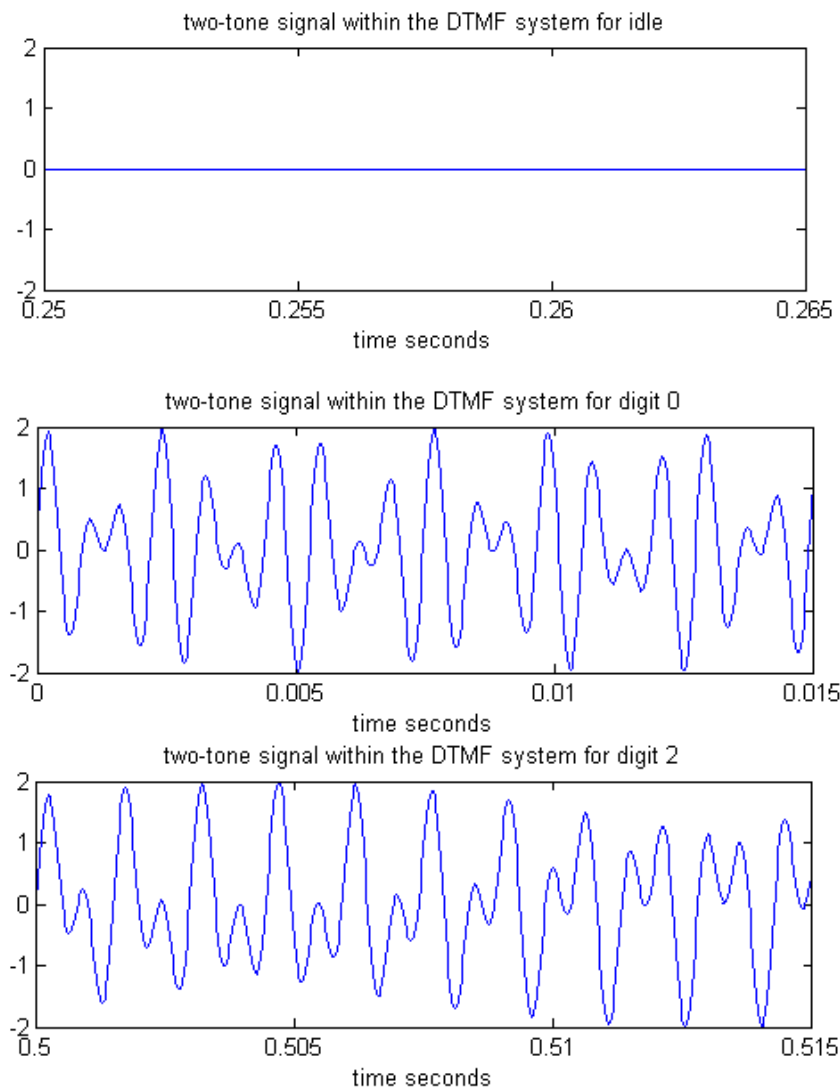


Fig 14. Two tone signal for idle, digit 0 and digit 2

The sampling rate used for the two-tone signal is 32768Hz. This particular sampling rate is used because the 32768 quartz crystal is the most commonly used low frequency oscillator. Manufacturer have developed these crystals to a very high precision. In 1963, when the touch tone telephone system was introduced by AT&T the company decided to use the most common low frequency quartz oscillator the 32768. And this remained until today after ITU-T standardized this. Another reason why the 32768 is chosen is because the 32768 is the 15^{th} power of 2. Thus using a divided by 2 circuit, we can easily obtain an accurate one second ticker.

3.2 Periodogram of the telephone number

The spectrogram function on MATLAB can be used to generate spectrograms of the y sequence corresponding to a two tone telephone number. From MATLAB documentation, we know that the spectrogram functions is as follows:

`S=spectrogram(x,window,noverlap,nfft,fs)` where x is the sequence to be analysed window is the length of the hanning window(length of each segment), noverlap is the number of overlap between segments, nfft is the fft length based on the segment length and fs is the sampling frequency.

We are required to segment the y sequence in such a way that the touch-tone information and the idle information are separated. Since the two alternates for every 0.25 second this correspond to 8192 samples at the sampling rate of 32768Hz. So based on this the following matlab command was used.

```
figure(2);
spectrogram(y,8192,0,8192,32768);
axis([500 1500 0 5.5]); title('periodogram of telephone numbers');
```

Which generated the following periodogram corresponding to the telephone number: 02067213345

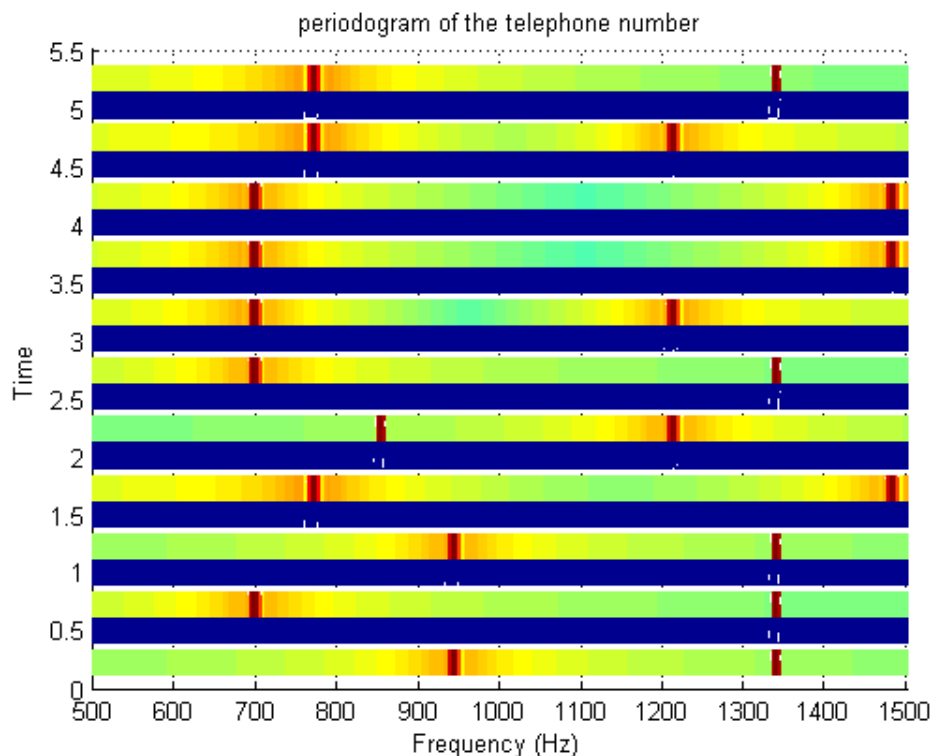


Fig 15. Periodogram of the telephone number: 02067213345

3.3 Identification of the telephone number from the periodogram

From the spectrogram plot and by referring to the frequency table in figure 12, the individual numbers can be clearly identified by its two frequency components.

	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

Taking the example for time between 0 and 0.25 second, the spectral component on the frequency spectrum peaks at 940Hz and 1330Hz. This confirms the presence of two sine waves at these frequencies. And by comparing with the frequency table we can identify the key being pressed between time 0 and 0.25 second is 0.

So the key to identifying which key is being pressed is knowing the where on the spectrogram the two peaks occur.

3.4 A noise corrupted two-tone signal

It is common in real life applications that noise will corrupt the two tone signal. This section will repeat the previous steps and examine whether the two tone signal can still be identified when noise is present. The noise used here is white noise and three difference variances of the noise will be used.

Noise Variance = 0.5

First, the noise variance is set to 0.5 by using `sqrt(0.5)*randn()`; if we plot the two tone signal for digit 0 and digit 2 again we will observe the following graphs.

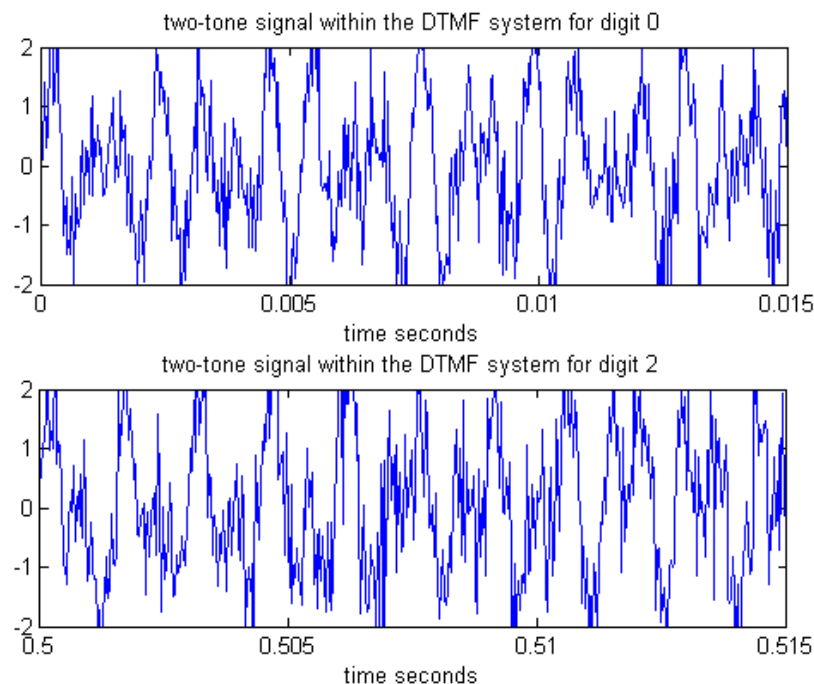


Fig 16. Two tone signal for digit 0 and digit 2 with added white noise (noise variance = 0.5)

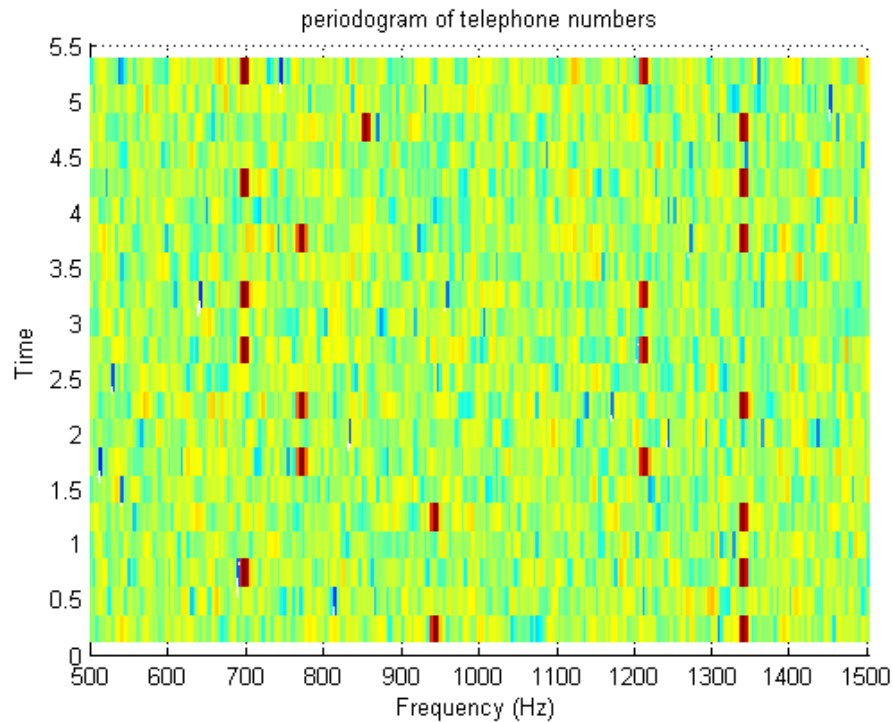


Fig 17. Periodogram of sequence y with added white noise (noise variance = 0.5)

We can see that although the original noise has been corrupted by the white noise, it is still possible to identify the spectral peaks from the periodogram hence the telephone number it represents.

Noise Variance = 1

We know increase the noise variance to 1. At this noise level the two-tone signal y is almost unrecognizable for digits 0 and 2

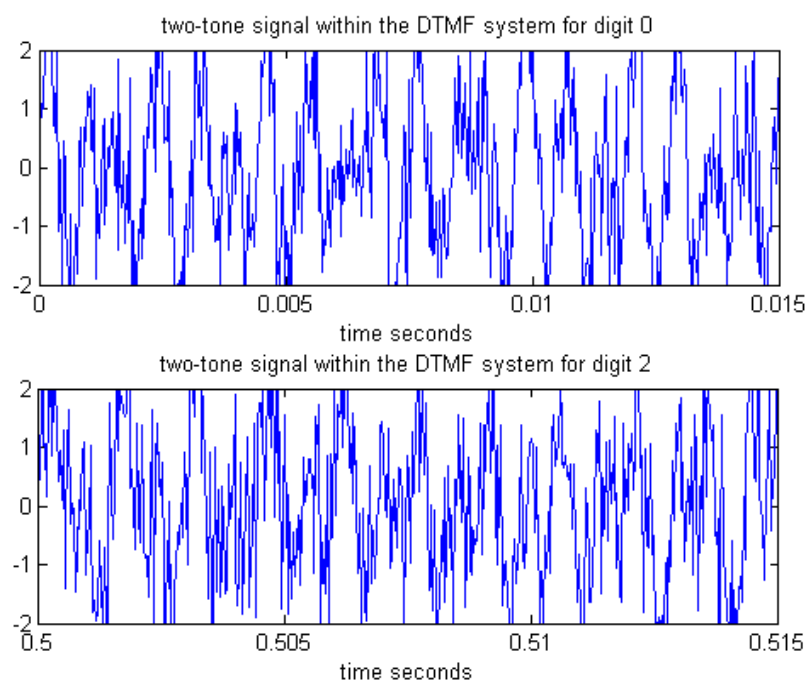


Fig 18. Two tone signal for digit 0 and digit 2 with added white noise (noise variance = 1)

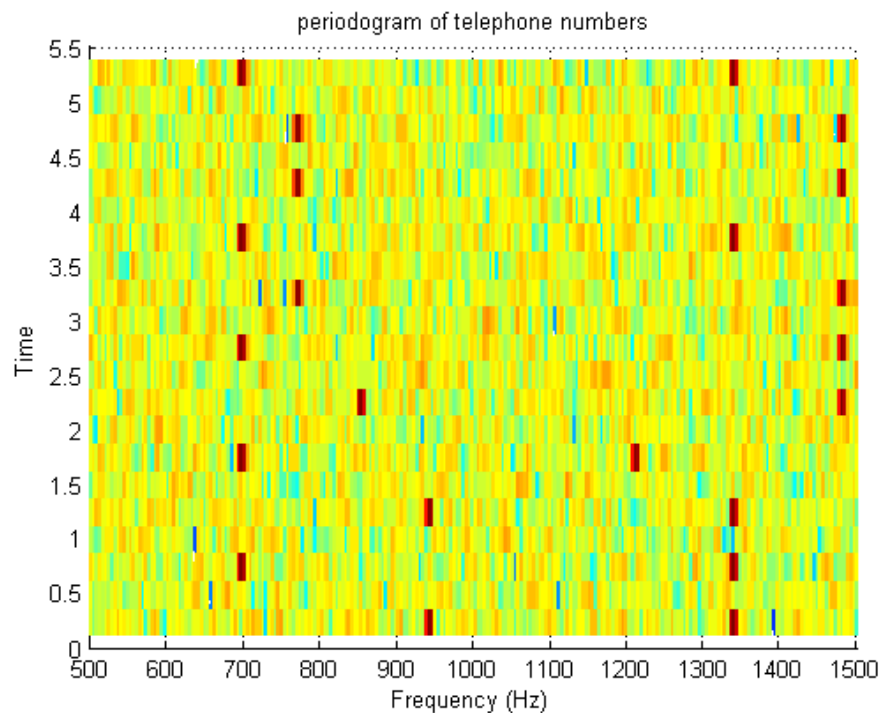


Fig 19. Periodogram of sequence y with added white noise (noise variance = 1)

Noise Variance = 2

At this level of the noise variance, if we examine sequence y, the two tone signals can no longer be identified, but if we examine the spectrogram it is still possible to tell the telephone number that we have just generated.

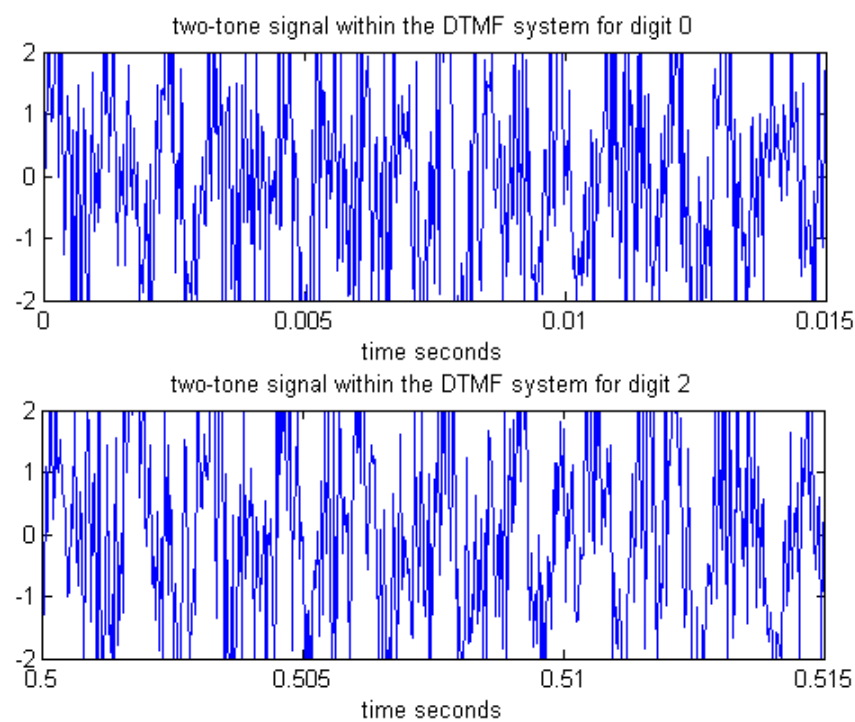


Fig 20. Two tone signal for digit 0 and digit 2 with added white noise (noise variance = 2)

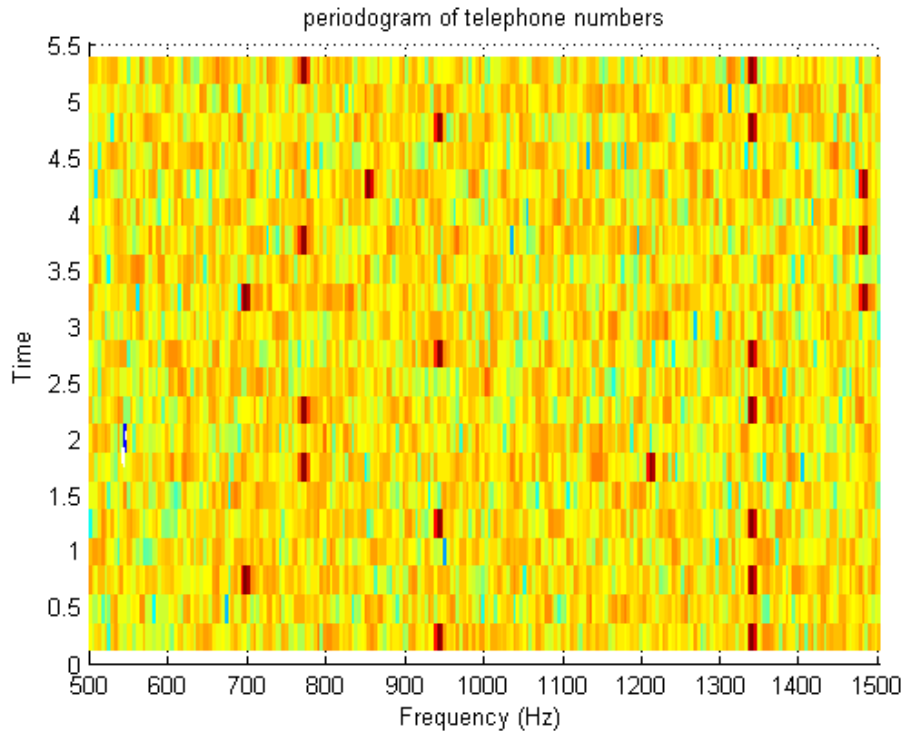


Fig 21. Periodogram of sequence y with added white noise (noise variance = 2)

In summary, as the noise level increases the identification of tones becomes increasingly difficult. However, the spectrogram still provides a robust solution for tonal identification.

4. OPTIONAL: spectrum of dialed signals using AR modeling

In addition to the periodogram, it is also possible to estimate the PSD of a process by performing autoregressive spectrum estimation to identify the tones. An example will be shown here for tones 3 and *.

First the two tones will need to be generated in MATLAB. They are stored in arrays tone1 and tone2 respectively.

```
fs = 32768; % sample rate
ts = 1/fs; % sample period
N = 0.25/ts; % total number of samples
t = (0:N-1)*ts; % Time vector
xaxis = 0:0.25/N:0.25; % axis label

count = 1;
for index=1:(0.25/ts) %generate 0.25 seconds of tone 3
    tone1(count) = sin(2*pi*697*t(index)) ...
        + sin(2*pi*1477*t(index));
    count = count + 1;
end;

count = 1;
for index=1:(0.25/ts) %generate 0.25 seconds of tone *
    tone2(count) = sin(2*pi*941*t(index)) ...
        + sin(2*pi*1209*t(index));
    count = count + 1;
end;
```

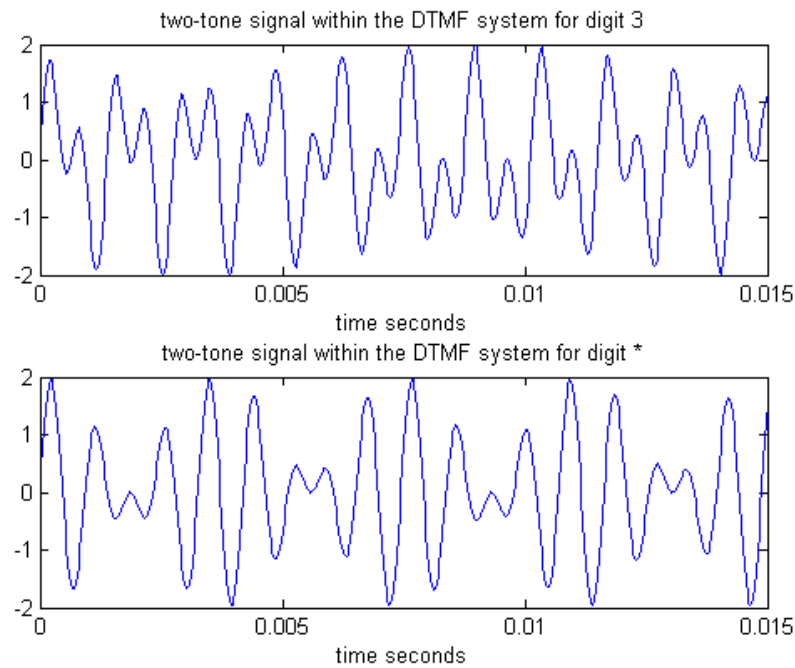


Fig 22. Two tone signal for digit 3 and digit *

In order to model the two tones by an AR process, it is necessary to identify the order of the AR process which is needed to model the signal. This is done by calculating the Partial Autocorrelation function of the 2 tones.

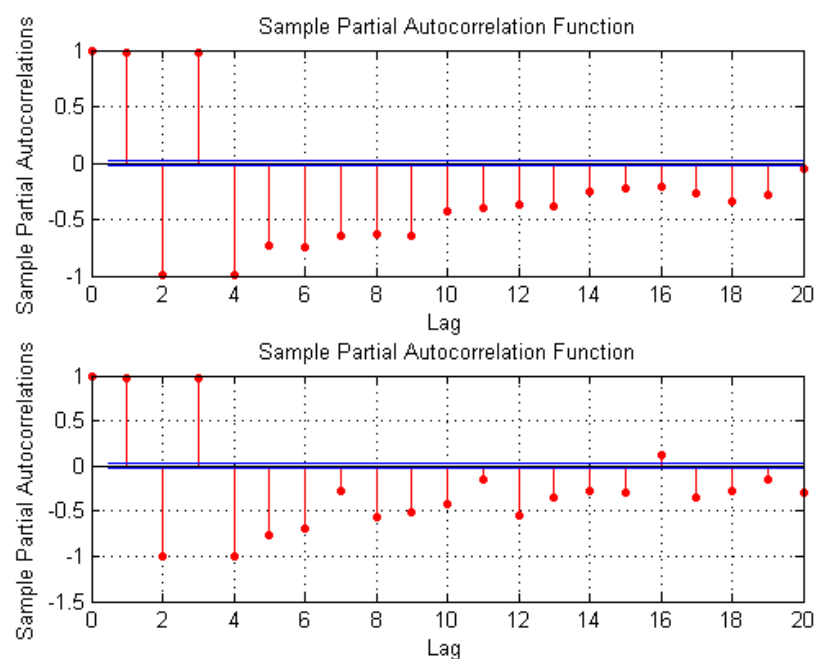


Fig 23. PAF of the tones digit 3(above) and * (below)

Despite that the lab instructions indicated that the AR process should be 4th order, from the PAF we can see that this is not the case. For digit 3, the order of AR is approximately 19 and for digit * the order is approximately 15. For simplicity, order 19 AR process will be used for both cases.

Next we use the knowledge of the order to compute the PSD of the AR19 process. But we first need to estimate the coefficient of the AR process.

```
>> coeff = aryule(tone1,19)

coeff =

Columns 1 through 10

    1.0000   -1.3372    0.1195    0.0946    0.0735    0.0557    0.0410    0.0291    0.0194    0.0116

Columns 11 through 20

    0.0052   -0.0002   -0.0053   -0.0105   -0.0162   -0.0231   -0.0315   -0.0418   -0.0546    0.1748
```

Then the PSD can be calculated by using the freqz function as it has been done in section 3.1.

```
[h w] = freqz(1,coeff,512);

plot(w/(2*pi),abs(h)/19.^2);

axis([0 0.05 0 1.6]);
```

The following PSD is generated for tone 3 and tone *.

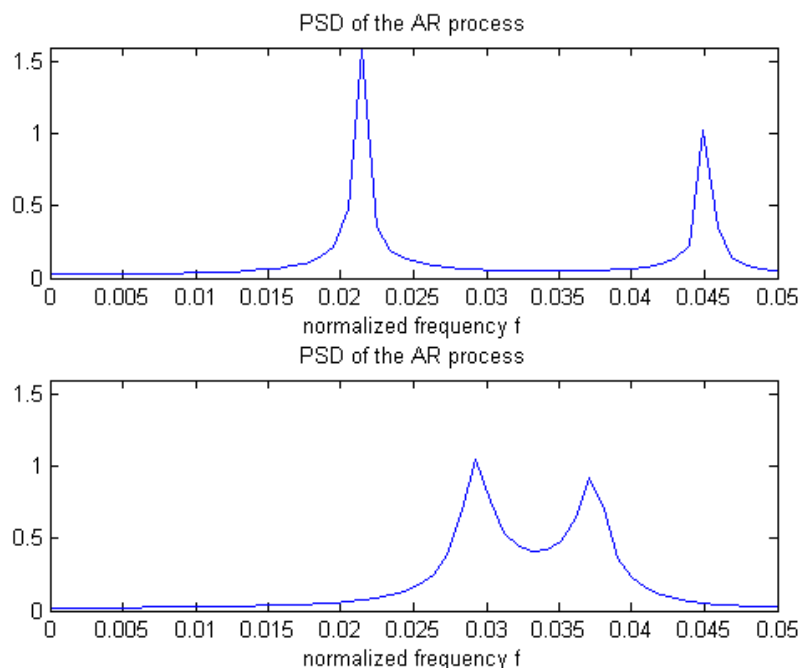


Fig 24. PSD generated using AR coefficients for tone 3 and tone *

From the graphs we can use the peaks to find the corresponding frequency pairs. In the first graph for tone 3, the peaks occur at approximately 0.021Hz and 0.045Hz for the normalized frequency. For a

sampling frequency of 32768Hz this correspond to 688Hz and 1475Hz which exactly correspond to the frequencies specified for tone 3 in the dial pad table. For the second graph the peak are at 0.029Hz and 0.037Hz on the normalized frequency axis. This corresponds to frequencies 950Hz and 1212Hz at the 32768Hz sampling rate. From the look up table, we can again clearly identify the key to be *. This shows that estimation using the AR modeling method works well.

We then add noise to the signal, but it is discovered that with the presence of noise, the performance of the PSD degrades.

At a noise variance of 0.01, although the key 3 can still be identified, the key * only appears to have a single peak due to the proximity its two frequency components

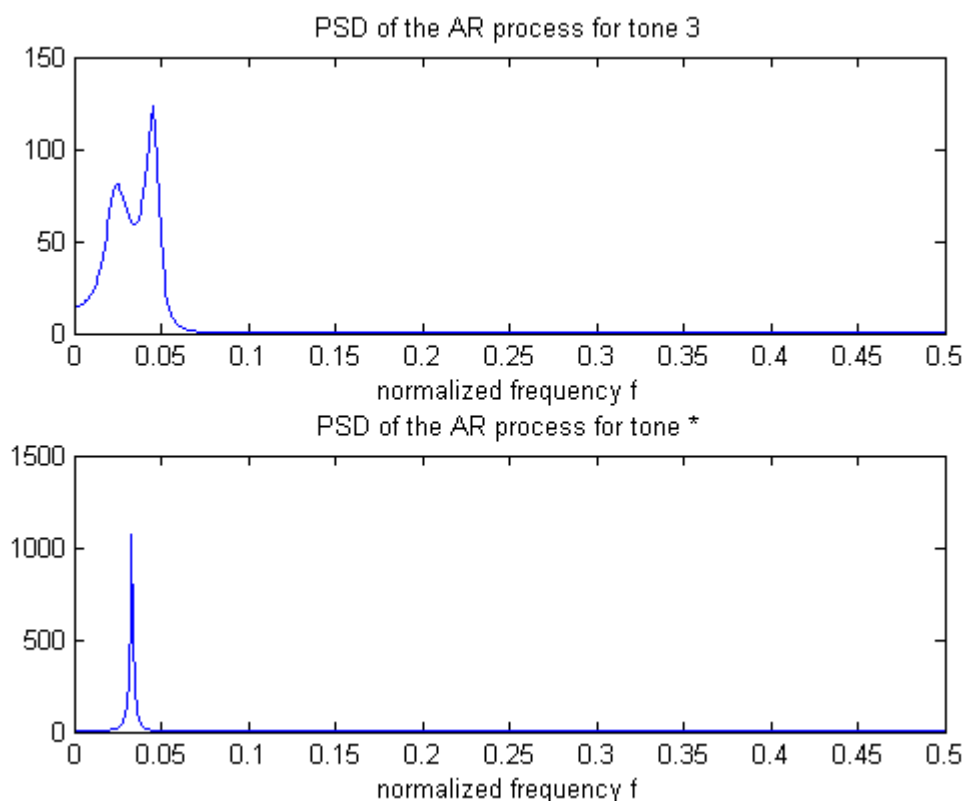


Fig 25. PSD generated using AR coefficients for tone 3 and tone * with white noise

Compared to the spectrogram estimation method, the AR model estimated spectrum shows a much degraded performance. If two of the frequency components are close together, then identification fails at a very low noise level.

PSD with other noise levels are plotted on the next page:

Noise Variance = 0.01

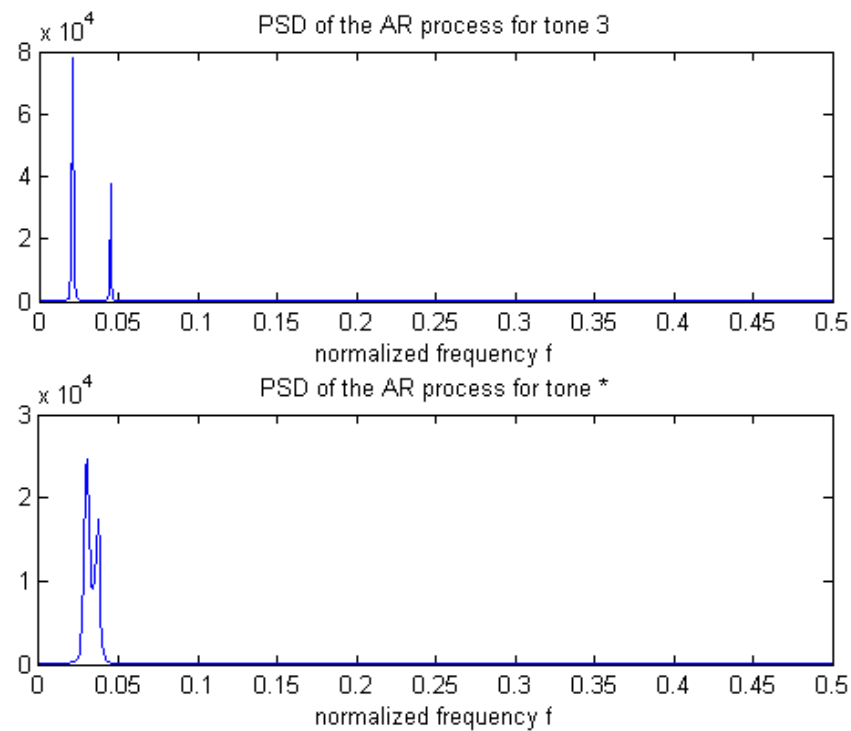


Fig 26. PSD generated using AR coefficients for tone 3 and tone * with white noise

Noise Variance = 0.5

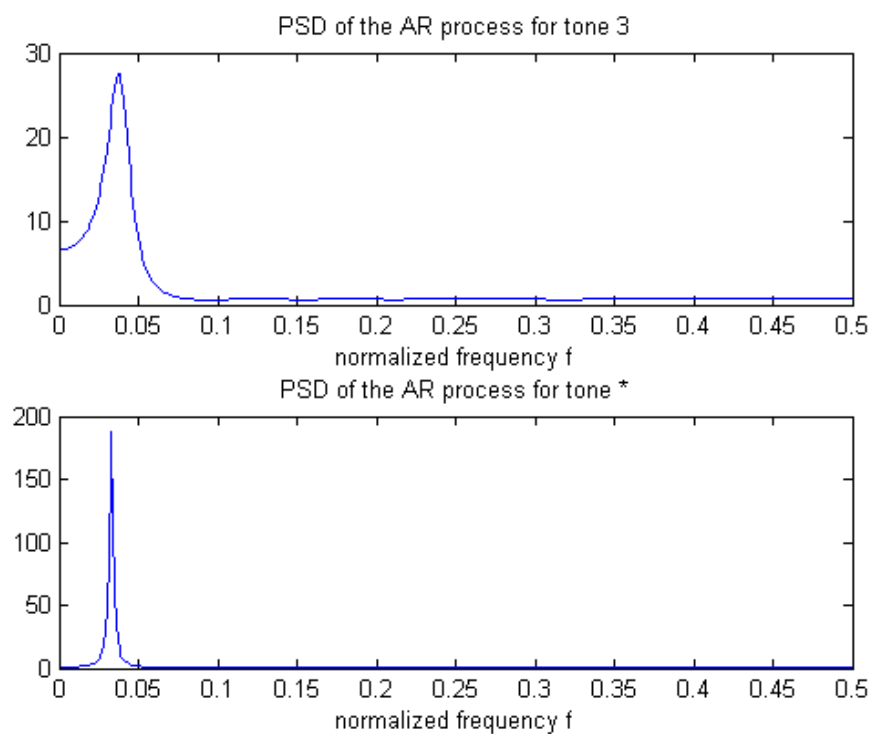


Fig 27. PSD generated using AR coefficients for tone 3 and tone * with white noise

Advanced Signal Processing (ASP)

Coursework 4

Name: Di Wu

CID: 00646091

Login: dw1310

Declaration: I confirm that this submission is my own work. In it, I give references and citations whenever I refer to or use the published, or unpublished, work of others. I am aware that this course is bound by penalties as set out in the College examination offenses policy.

Signed: Di Wu

Aim: The 4th coursework will deal with a real life example. We are faced with the scenario that we have purchased a vinyl copy of the original fourth Led Zepplin album, but we discovered that the album is defected in the sense that we found irregular dent across the grooves in Track 4. These grooves cause a whistling tick to be added to the original sound which is very unpleasant. The aim of this coursework exercise is to apply the ASP knowledge that we have gained from previous labs to come up with an algorithm that removes the tick from the music.

1. Noise identification

The first step that needs to be undertaken to remove the noise from the music is the correct identification of the noise. We first listened to the music using the sound function in MATLAB. Despite the pleasant tune of the music, we can hear a sharp whistling noise at above 1 second intervals. A periodogram is then plotted to help us to identify the exact nature of this noise. As we have covered in previous coursework exercises, a periodogram can tell us the power of each spectral component.

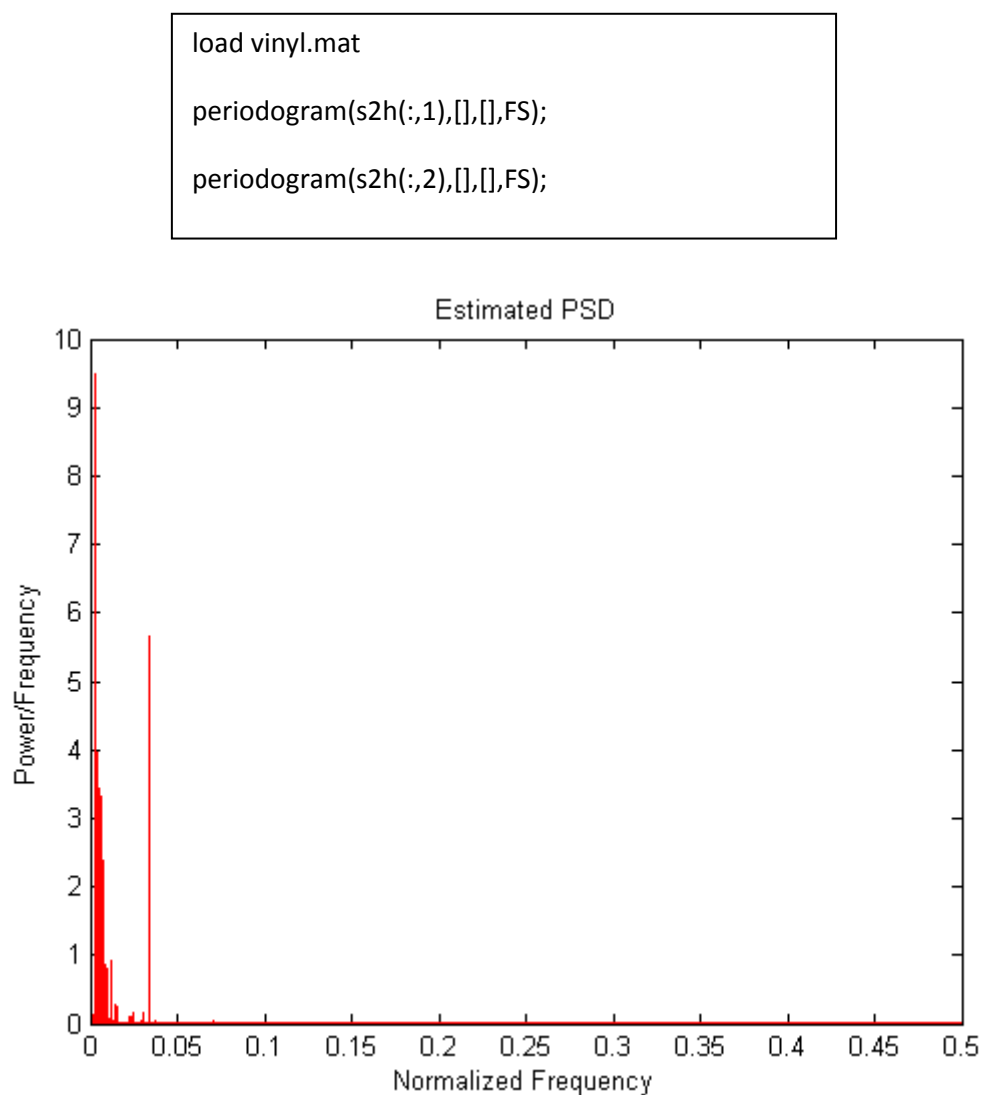


Fig 1. Periodogram of Stairway to Heaven with noise, Left Channel

The first music we are analysing is Stairways to Heaven. From background reading, it has been discovered that the human ear is most sensitive for frequencies between 200 – 3500Hz. Since we can hear the whistle clearly we can identify the whistle to be present in this frequency range.

Zooming in to the frequency range indicated above. We obtain the following zoom periodogram.

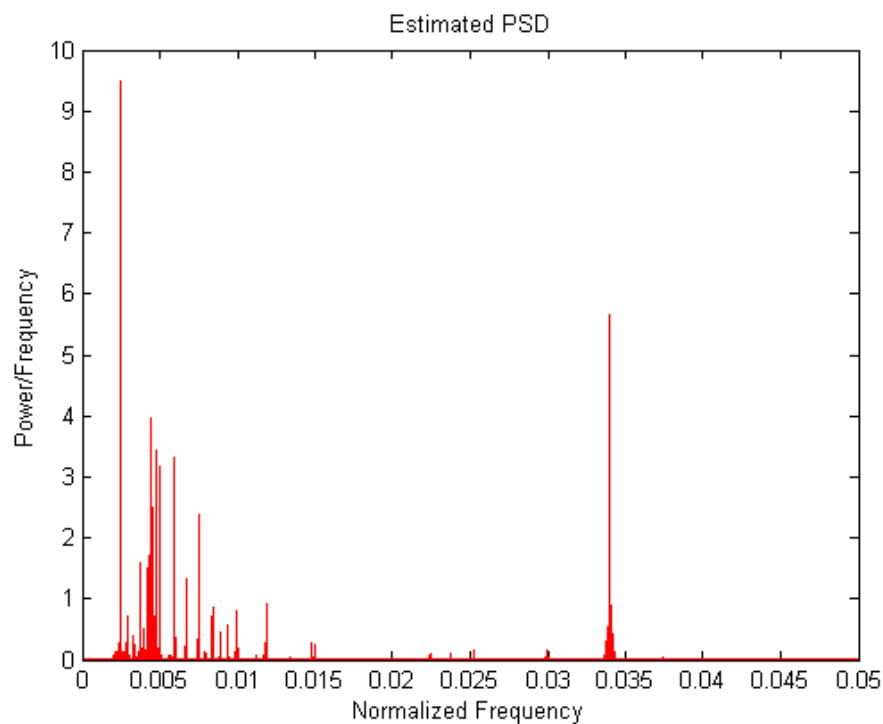


Fig 2. Periodogram of Stairway to Heaven with noise (Zoomed IN Left Channel)

From the zoom periodogram we can identify an unusual peak at $f=1.5\text{kHz}$, so this is most likely to be the spectral component of the whistle noise. We now examine the right channel to verify this.

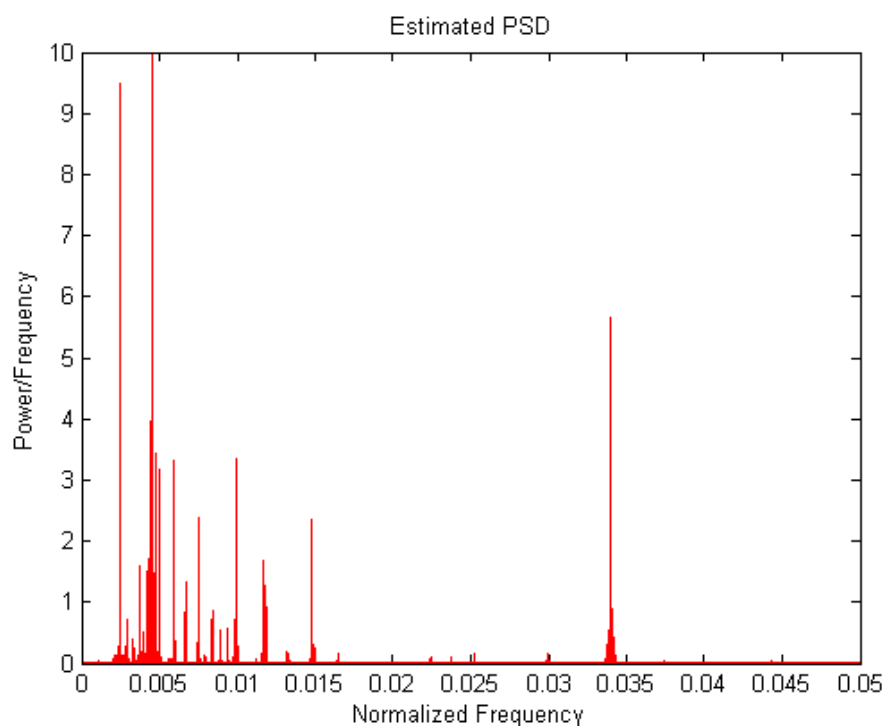


Fig 3. Periodogram of Stairway to Heaven with noise (Zoomed IN Right Channel)

From the periodogram of the right channel we observe that in addition to the spike at 1500Hz there exists an additional spike at 200Hz.

2. Comparison to Original Sound

We now compare the noise corrupted version of the song to the original version of the same song.

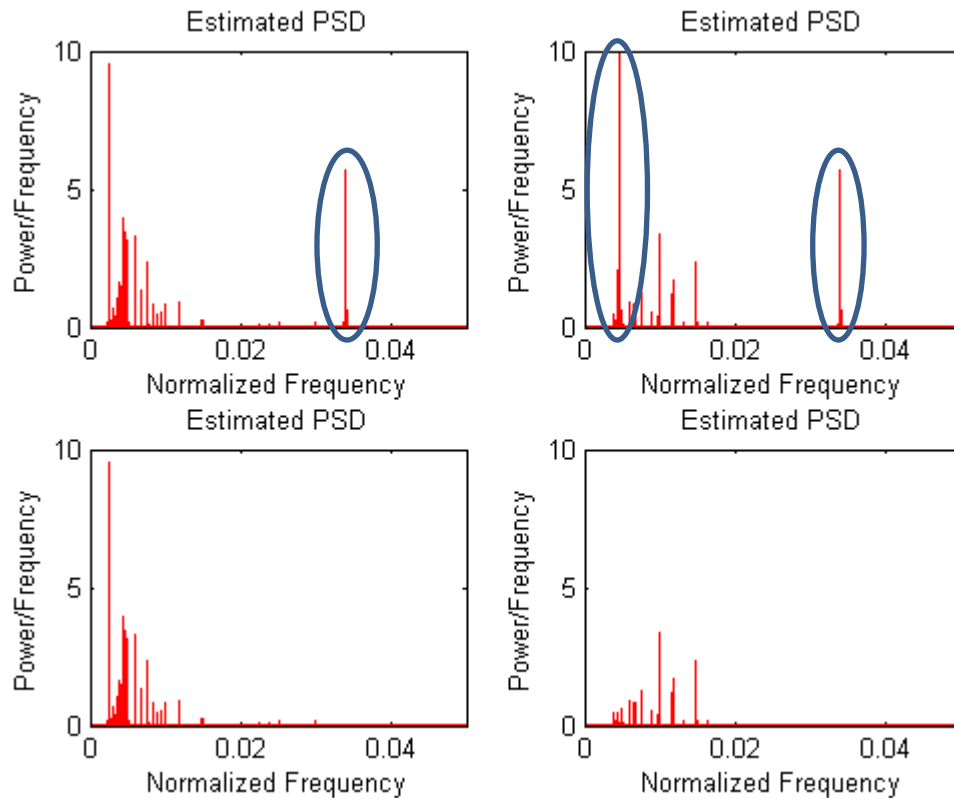


Fig 4. Periodogram of **Top Left:** Left Channel with noise **Top Right:** Right Channel with noise
Bottom Left: Left Channel original **Bottom Right:** Right Channel original

We can see that our prediction in the previous section is indeed correct. By comparing the noise corrupted periodogram to the original periodogram, we can clearly see that an extra peak exist at 1.5Hz, in addition we can also see an additional peak at 200Hz in the right channel. We can hence conclude that the spectral component of the whistle noise is concentrated around 1.5kHz and 200Hz

We can further verify this by plotting a spectrogram using these specifications:

```
spectrogram(s2h(:,1),256,0,256,FS);  
axis([0 5000 0 30])
```

From the spectrogram we can observe a periodically occurring noise (circled in the diagram) at 1500Hz and 200Hz. This observation is consistent with the result obtained for the periodogram estimate.

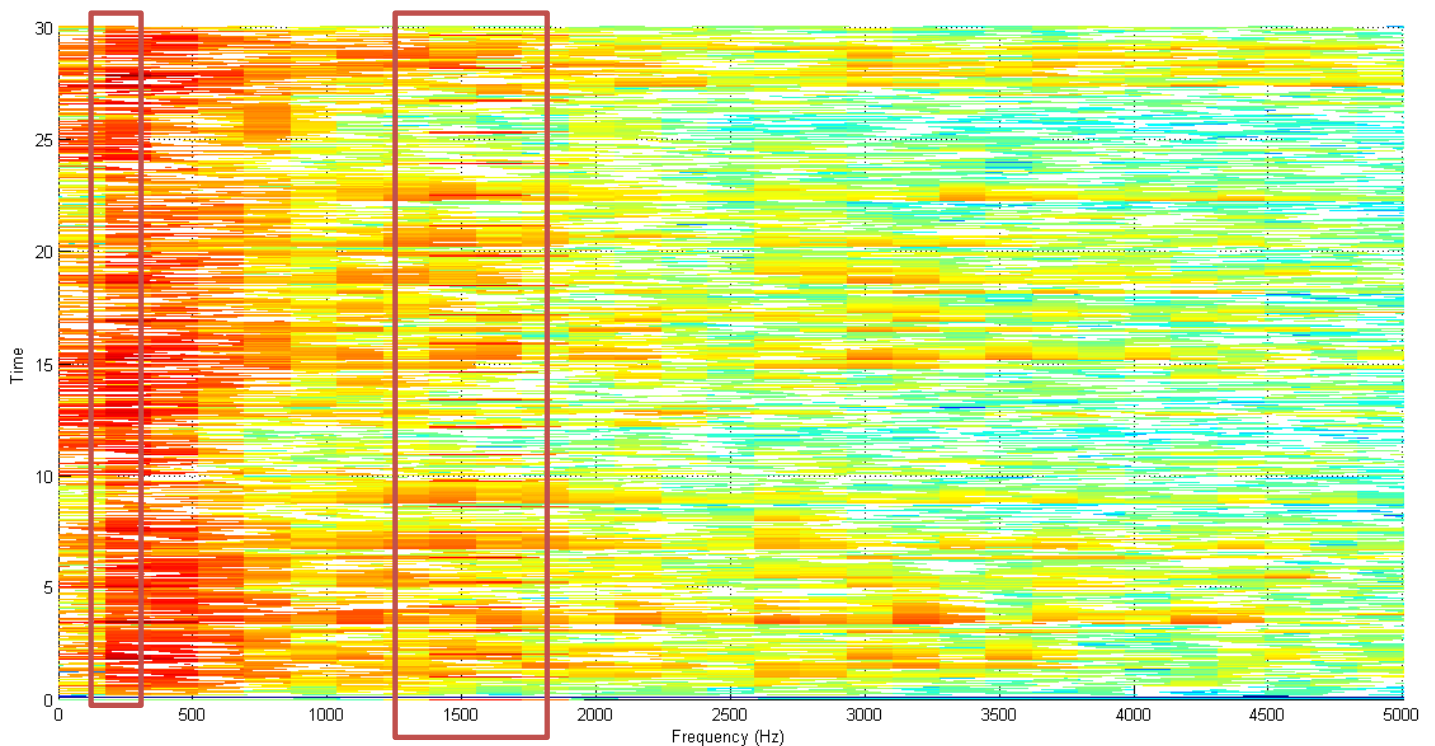


Fig 5. Spectrogram of Stair to Heaven

3. Distortion Removal using a Butterworth Filter

Having identified the frequencies of the noise components, it is now possible to remove the noise by using a butterworth filter. The butter worth filter used is a band stop filter and the corner frequencies of the filter is selection to be [150,250] for the first filter and [1400,1600] for the second filter. The frequency response of the 2 filters is shown below:

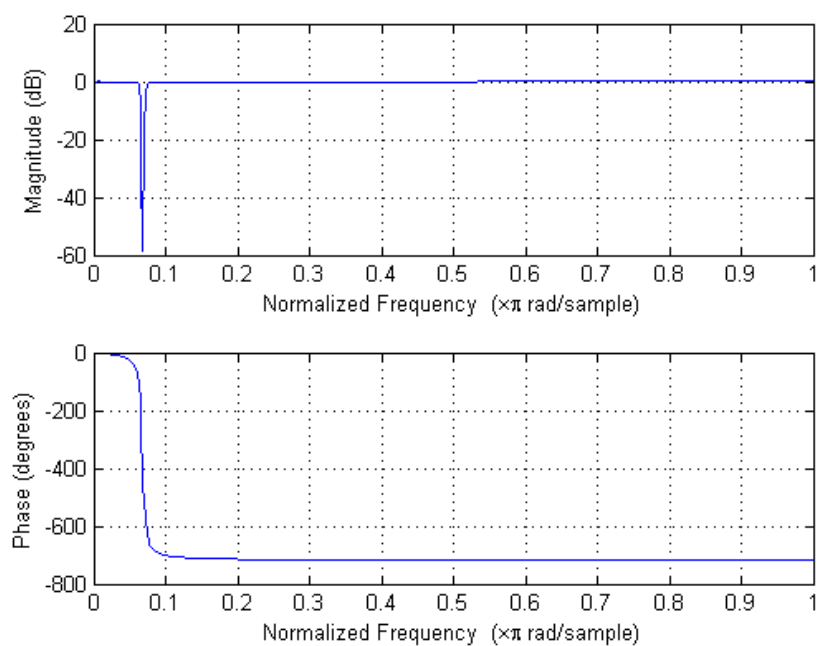


Fig 6. Frequency response of the first butterworth filter

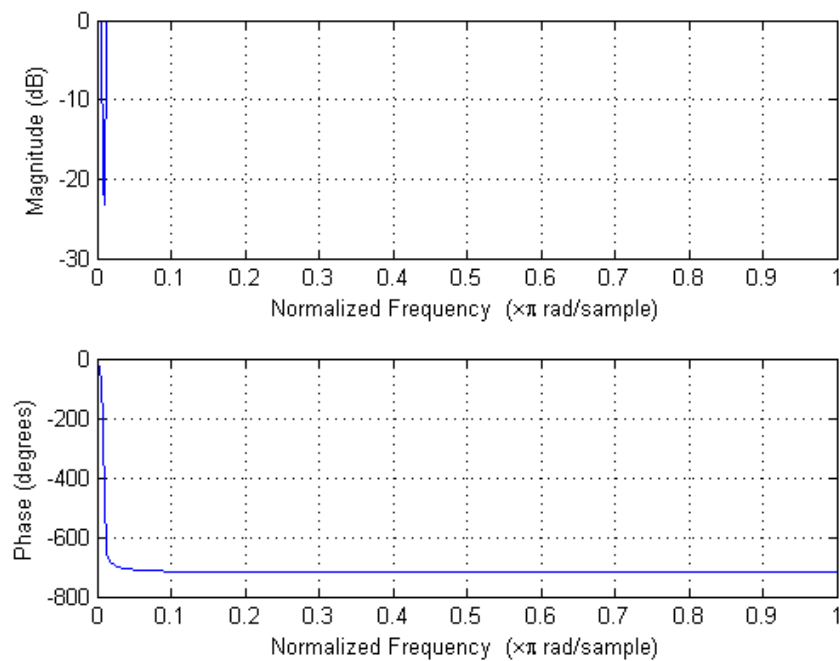


Fig 7. Frequency response of the second butterworth filter

After applying these filters to the noise corrupted music and by using the sound function, we hear that the music output is now much better. We can examine the improved version of the output using a spectrogram again which shows that the noise components are no longer present.

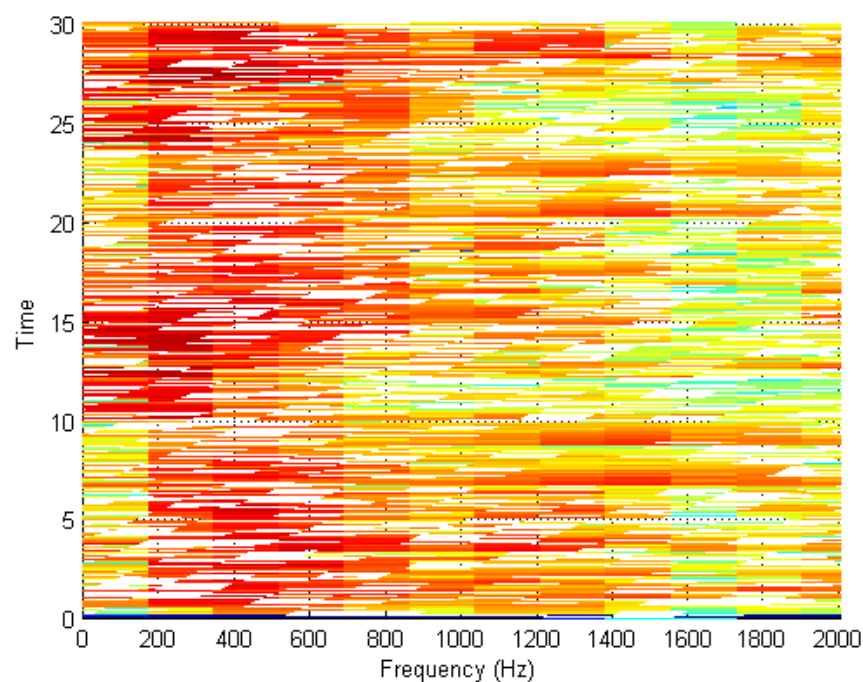


Fig 8. Spectrogram of filtered output

4. Calculating the Error

After removing the noise, the periodogram the original, corrupted and filtered music are plotted on the same set of axis for the left and right channels and compared together.

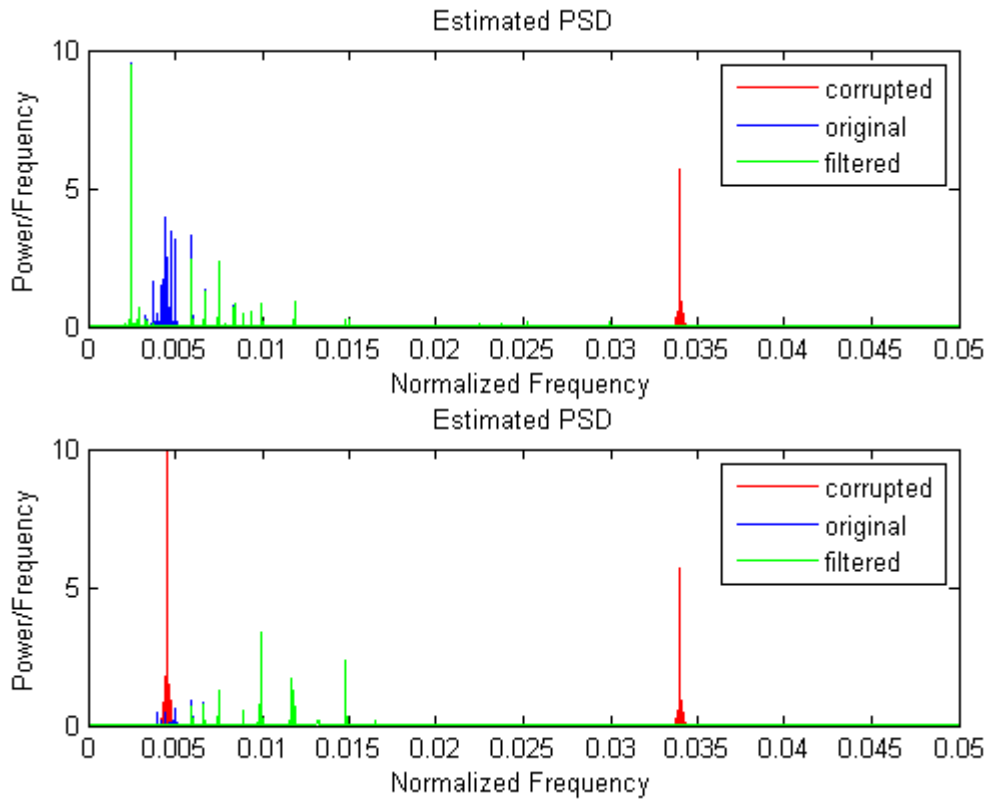


Fig 9. Periodogram of **Top**: left channel **Top Right**: right channel

Since the spectrograms are superimposed on top of each other, we can see that for the left channel it is clear that the desired frequency ranges (i.e. [150,250] Hz and [1400,1600]Hz) have been suppressed in both of the channels, but can also realized that some the original spectral component in this frequency range have also been suppressed. This will obviously affect the quality of the music that is being played.

We now assess the relative error given by the equation

$$\frac{||Pc - \hat{P}c||}{||Pc||}$$

The notation $||Pc||$ represents the Euclidian mean which can be calculated as follows:

$$||\mathbf{p}|| = \sqrt{p_1^2 + p_2^2 + \dots + p_n^2} = \sqrt{\mathbf{p} \cdot \mathbf{p}}$$

Hence by implementing this in MATLAB, we can calculate the error of the filtered signal compared to the original sample. For two sequences that match exactly, we can expect the error to be 0.

```
function [ error ] = error_calc(p,p_est)
%Calculate error WRT the gound truth recording
error = sqrt(sum((p-p_est).*(p-p_est)))/sqrt(sum(p.*p));
end
```

The following results are obtained for the left and right channels respectively.

```
e_left = error_calc(pl,pl_est)

e =      0.7044

e_right = error_calc(pr,pr_est);

e =      0.2170
```

Note that in the low frequency butterworth filter was applied to the left channel even though there was no noise present at that frequency for the particular channel. Hence the error for the left channel is larger than that of the right.

We can see that by using a butterworth filter, the output signal indeed has a large error as compared to the original music. This is because the filter also removed spectral components of the song which is certainly undesirable.

5. Enhancement using the Least Mean Square Algorithm

The previous denoising algorithm requires the analysis of the whole content of the disc, alternatively a least mean square estimator can be used that compares the faulty song with the original recording sample by sample to eliminate the noise in the noise corrupted song. From the lecture notes the least mean squared algorithm can be seen as follows:

The LMS summary:

Initialisation. $w_k(0) = 0, \quad k = 1, \dots, p \quad \equiv \quad \mathbf{w}(0) = \mathbf{0}$

Filtering. For $n = 1, \dots$ compute

$$y(n) = \sum_{j=1}^p w_j(n)x_j(n) = \mathbf{x}^T(n)\mathbf{w}(n)$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{x}(n)$$

Fig 10. Summary of LMS

\mathbf{X} in the input sequence whereas y represent the output sequence. \mathbf{W} is the gradient vector which is updated continuously to track the optimal solution. The gradient vector \mathbf{w} 's pointing average improves as the number of samples increases. The error term $e(n)$ in the equation represent the difference between the noisy and the teaching signal.

The output (improved version of the noisy signal) is given by multiplying the input with gradient vector \mathbf{w} .

The Algorithm is implemented in MATLAB as follows:

The system order used for the filter is 5 and the value for μ – the learning rate is taken to be 0.5.

```
%LMS script developed based on the sample script from MATLAB central available@
%http://www.mathworks.co.uk/matlabcentral/fileexchange/3649-lms-algorithm-demo
sysorder = 5 ; %filter order
ori = s2h_original(:,1); %the original music input
noisy = s2h(:,1); %the corrupted music input
totallength=size(noisy,1);
w = zeros ( sysorder , 1 ) ; %initialize w
for n = sysorder : totallength
    test = noisy(n:-1:n-sysorder+1) ; %pickup x(n) based on the filter order
    y(n)= w' * test; %estimated output
    e(n) = ori(n) - y(n) ; %compare estimated output with original
    miu = 0.5; %define the learning rate
    out(n) = w' * test ; %calculate the output
    w = w + miu * test * e(n) ; %update the next gradient vector
end
plot(noisy, 'b');
plot(out, 'r');
title('Comparison between the original and LMS filtered music sample') ;
legend('Noisy Version', 'LMS Filtered Version');
xlabel('Samples')
ylabel('Amplitude')
```

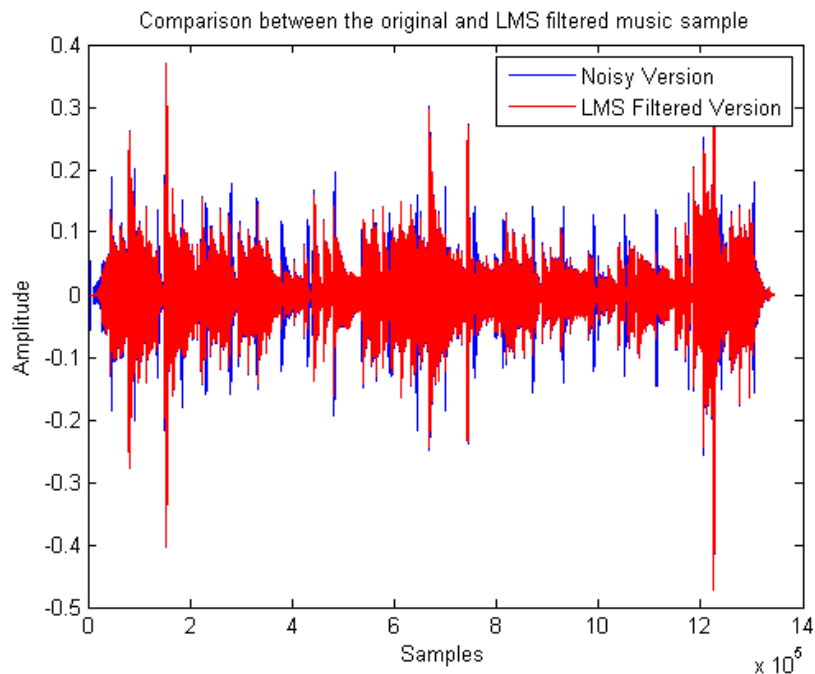


Fig 11. Comparison between the original and LMS filtered Music Sample.

By plotting the periodogram of the LMS filtered output, we can see that the spectral component for the whistling noise has been successfully suppressed showing that the LMS filter's performance is satisfactory.

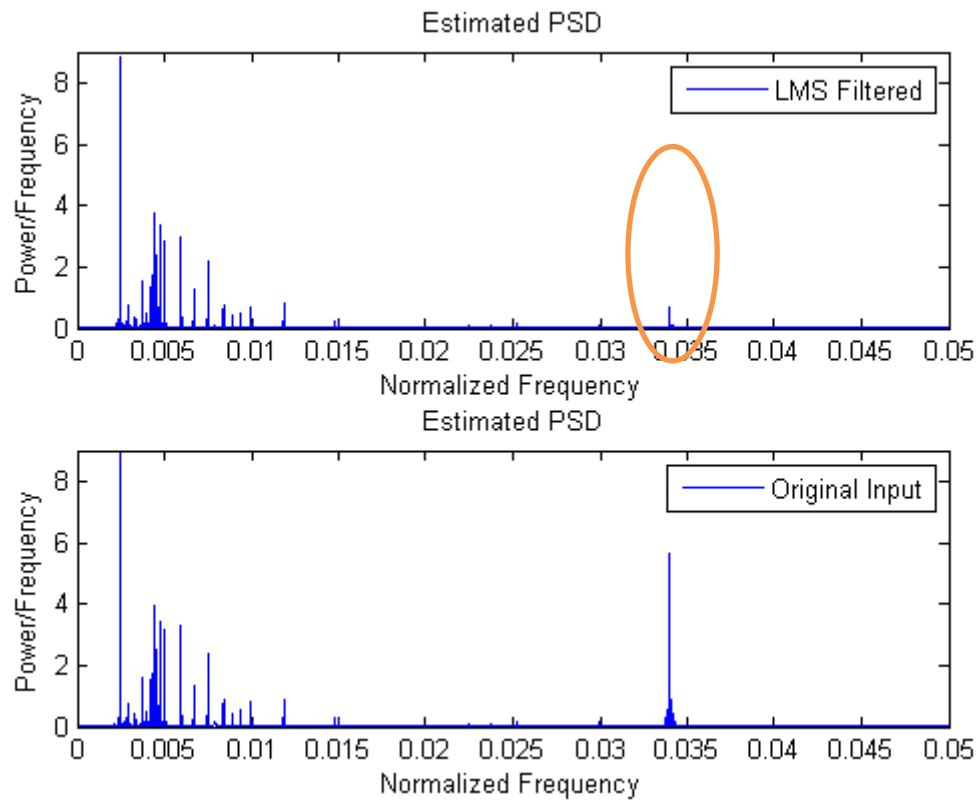


Fig 12. Comparison between the spectrogram of original and LMS filtered Music Sample.

The relative error of the filtered output is assessed again by the following equation

$$\frac{||Pc - \hat{P}c||}{||Pc||}$$

We found that the errors for both the left and the right channel have been reduced

```
e_left = error_calc(pl,pl_est)
e =      0.1519
e_right = error_calc(pr,pr_est);
e =      0.2055
```

The decreased error shows that the LMS adaptive filter algorithm has a lesser impact on the spectral component of the original song

6. Denoising the Liquid Tension Experiment's Universal Mind

Having analysed the Led Zeppelin's stairway to heaven, we now move on to analysing music of a different genre the LTE's Universal Mind. The version of universal mind we will be dealing with has also been corrupted by a whistling noise. We first plot the periodogram of the left and right channels of this noise corrupted signal.

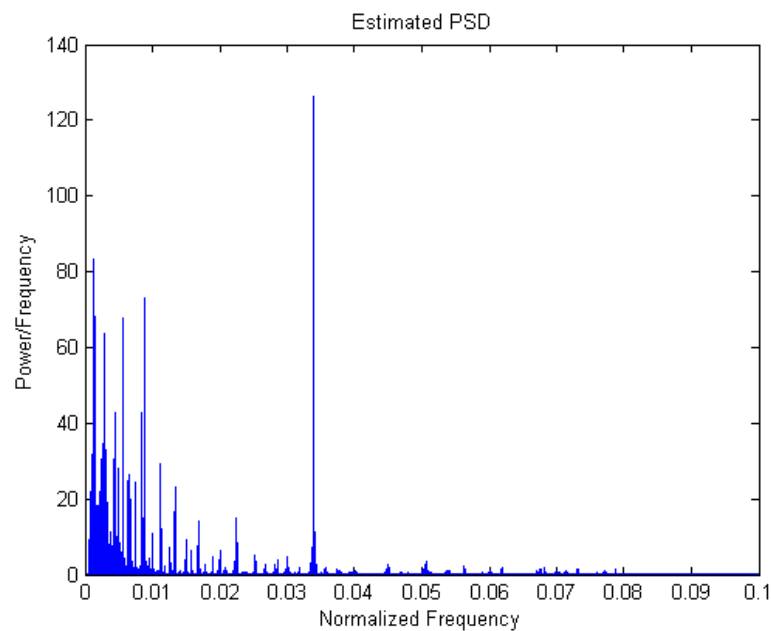


Fig 13. Periodogram of Universal Mind with noise, Left Channel

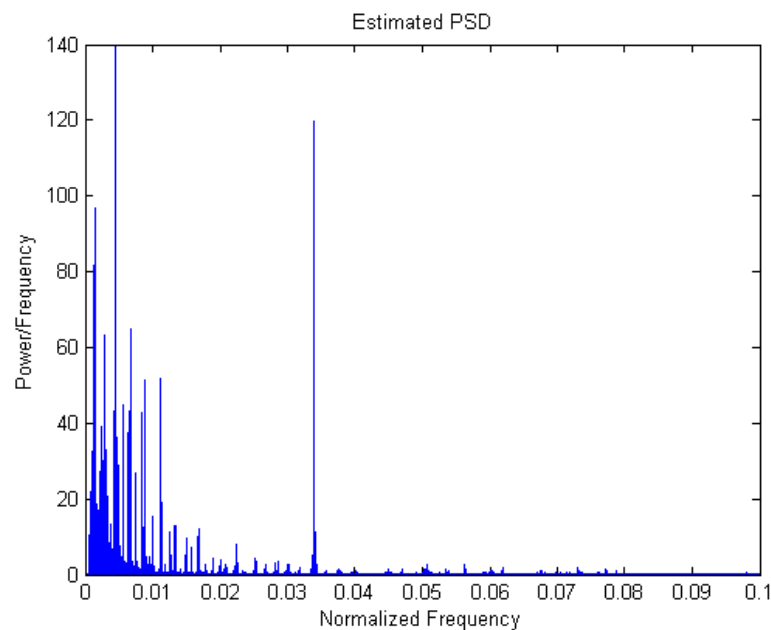


Fig 14. Periodogram of Universal Mind with noise, Right Channel

One thing that we can immediately realize is that the variety of spectral component of the Universal Mind is much greater than the case with Stairway to Heaven. For this reason, the axis of the periodogram graphs has been modified to display from 0 to 0.1 of the normalized frequency.

By comparing the periodogram of the noise corrupted version of Universal Mind to the original version for both of the channels we can easily identify the spectral component of the noise. In this case, the noise is seen at 1499Hz for both the left and right channels and there is also an additional noise component at 199Hz.

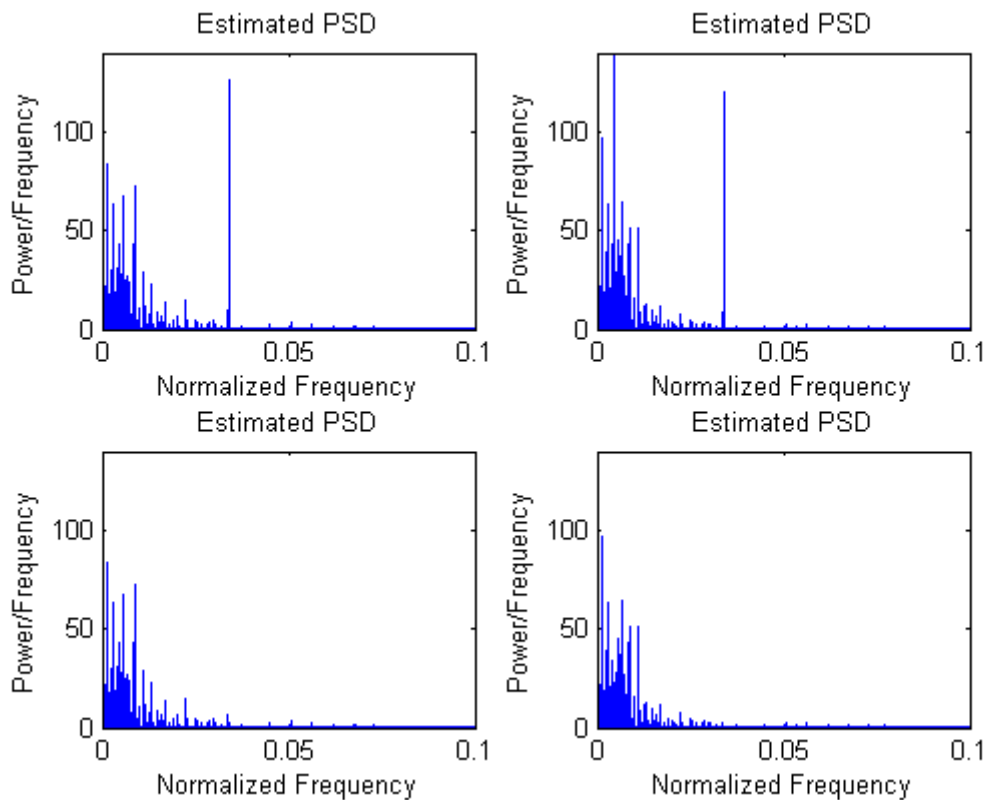


Fig 15. Periodogram of **Top Left:** Left Channel with noise **Top Right:** Right Channel with noise
Bottom Left: Left Channel original **Bottom Right:** Right Channel original

Next we can use the LMS adaptive filter to apply to the noise corrupted Universal Mind recording and to remove the whistling noise. (as we know that the LMS algorithm will produce less error than the butterworth filter) The LMS algorithm used is exactly the same as before with system order chosen to be 5 and the learning rate to be 0.2 after which the efficacy of the filter will then be analysed using the relative error equation:

$$\frac{||P_c - \hat{P}_c||}{||P_c||}$$

```

sysorder = 5 ; %filter order
ori = um_original(:,1); %the original music input
noisy = um(:,1); %the corrupted music input
totallength=size(noisy,1);
w = zeros ( sysorder , 1 ) ; %initialize w
for n = sysorder : totallength
    test = noisy(n:-1:n-sysorder+1) ; %pickup x(n) based on the filter
    order
    y(n)= w' * test; %estimated output
    e(n) = ori(n) - y(n) ; %compare estimated output with
original
    miu = 0.2 %define the learning rate
    out(n) = w' * test ; %calculate the output
    w = w + miu * test * e(n) ; %update the next gradient vector
end

```

After passing through the filter, the noisy version of the signal is compared with its filtered version in the time domain. We can see that large variations in the signal amplitude have been significantly reduced in both of the channels.

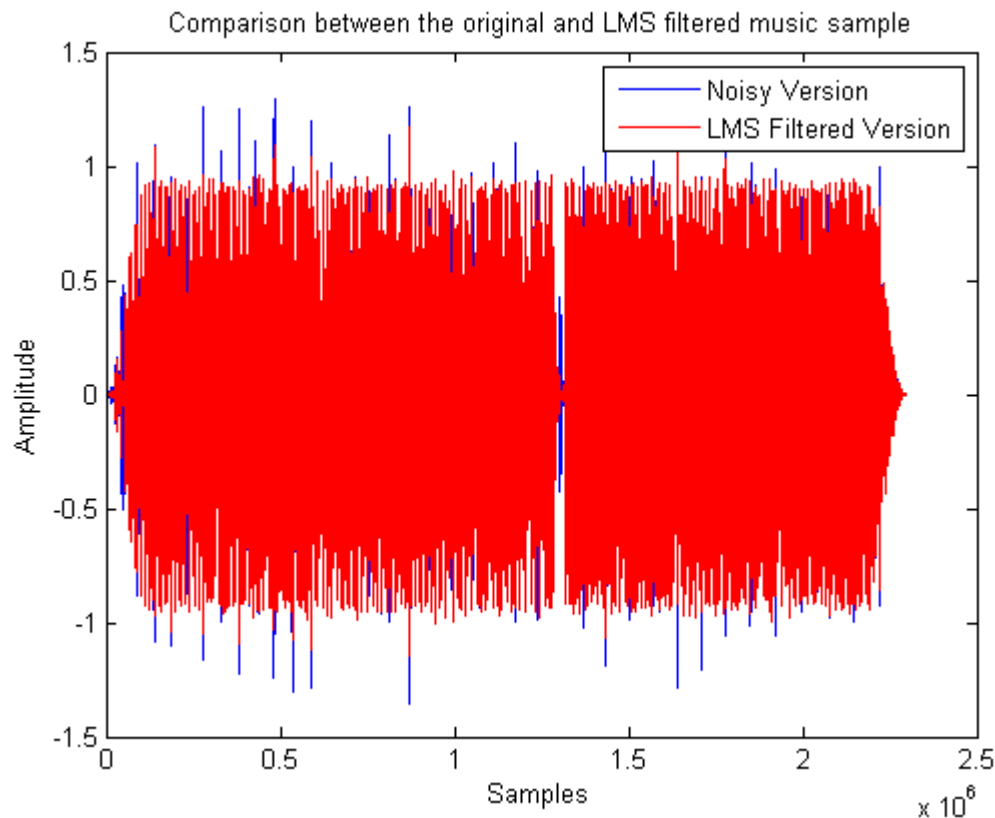


Fig 16. Comparison between the original and LMS filtered Music Sample, Left Channel.

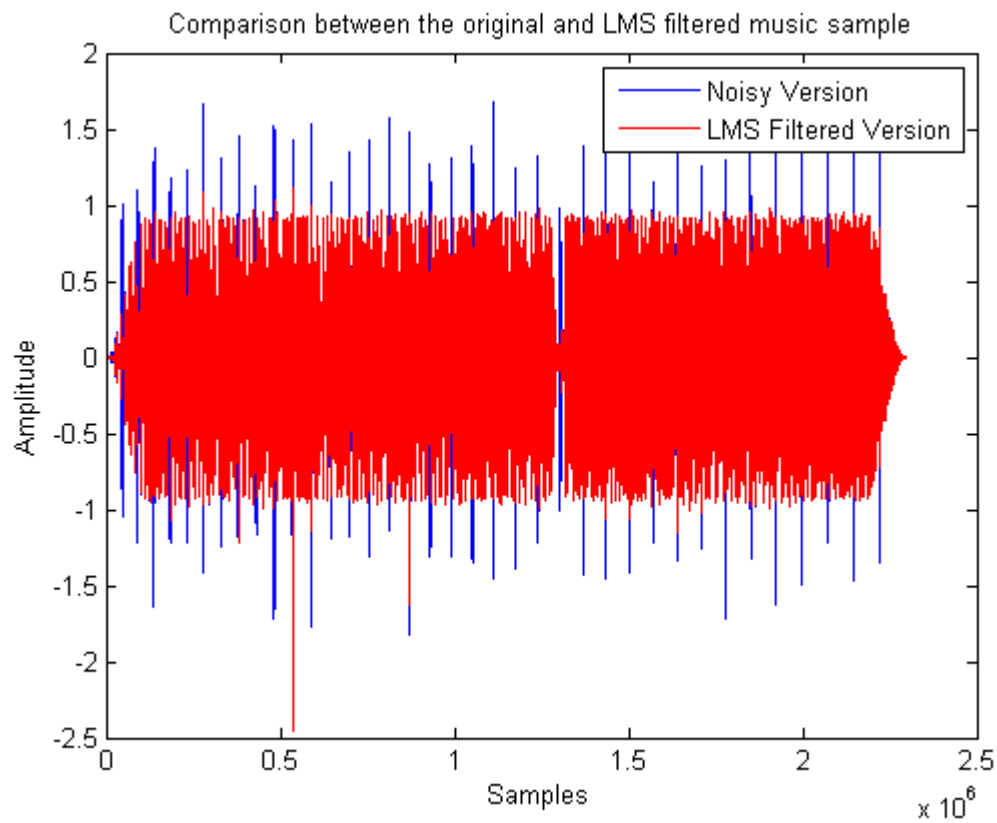


Fig 17. Comparison between the original and LMS filtered Music Sample, right channel.

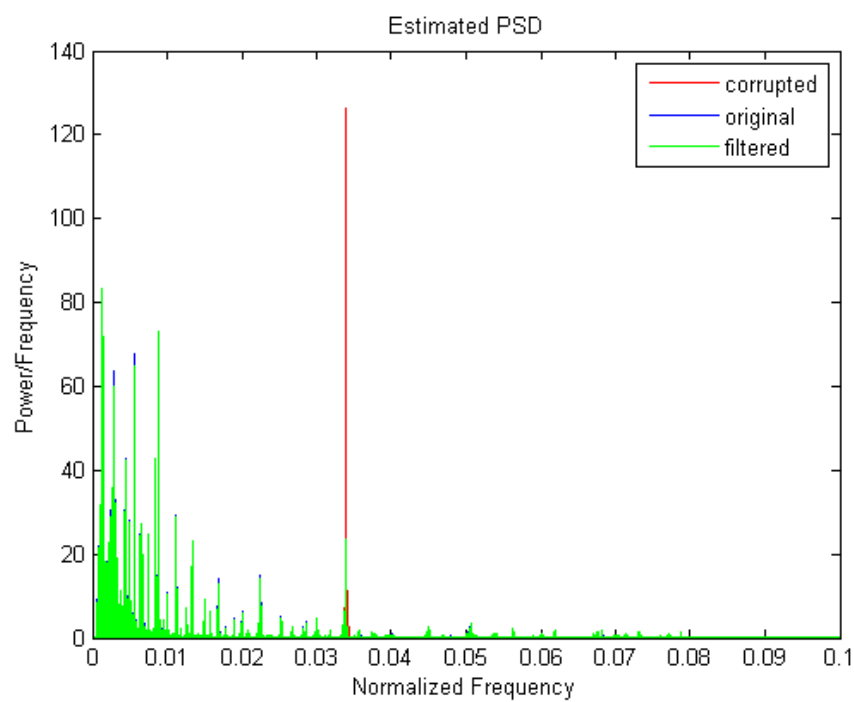


Fig 18. Periodogram of the left channel

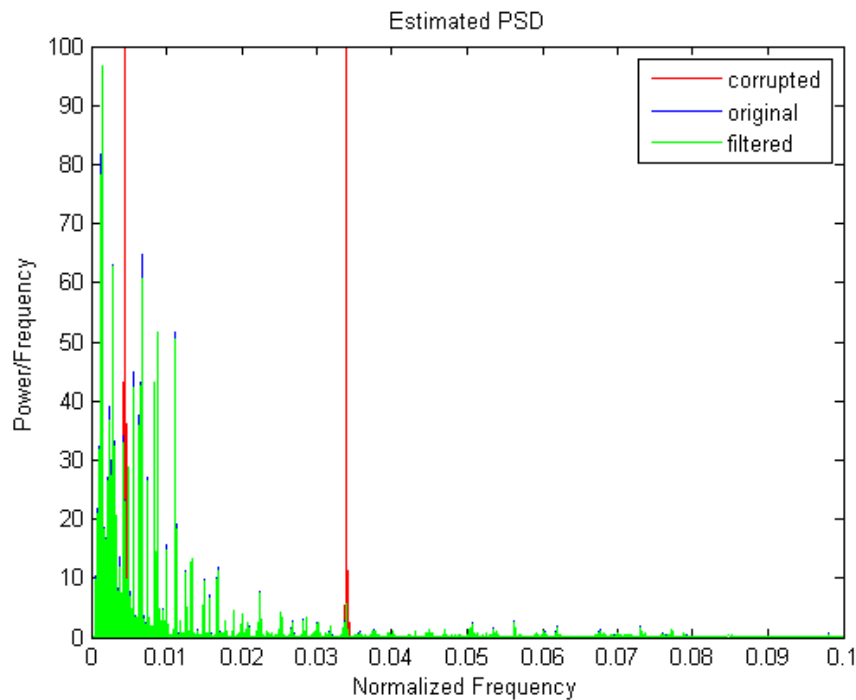


Fig 19. Periodogram of the right channel

When comparing the periodogram between the corrupted, original and filtered version of the music we can see that the noise components (Sharp red spikes) in both the left and the right channels are successfully suppressed by the LMS filter. Furthermore by comparing the filtered signal to the original signal, we can see that their PSDs are very similar. This shows that the LMS filter's performance is satisfactory as it managed to suppress the noise spectral component while still keeping the spectral component of the music.

Finally, the relative error of the LMS filter on the Universal Mind music file is evaluated. The error for this particular music file is smaller than the error when the LMS filter was used on the Stair to Heaven. This demonstrates the merit of the LMS filter which is capable to adapt the filter coefficients on a sample by sample basis despite the rich spectral component of the music file.

```
e_left = error_calc(pl,pl_est)
e =      0.1122
e_right = error_calc(pr,pr_est);
e =      0.1361
```

Finally, although both the butterworth and LMS algorithm resulted to some degree of spectral loss, the LMS adaptive filter offers a much improved performance.

Advanced Signal Processing (ASP)

Coursework 5

Name: Di Wu

CID: 00646091

Login: dw1310

Declaration: I confirm that this submission is my own work. In it, I give references and citations whenever I refer to or use the published, or unpublished, work of others. I am aware that this course is bound by penalties as set out in the College examination offenses policy.

Signed: Di Wu

Aim: The final piece of the coursework mainly deals with different forms of the adaptive filter. We will be reviewing the identification of adaptive filter systems such as the wiener, LMS and gear shifting filters. Finally, we will apply adaptive filters on a real world speech signal and discuss the advantage of such a system.

1. Wiener Filter

The Wiener Filter can be used to estimate/represent an unknown system by yielding an optimum filter that offers the closest match to the original filter system. To evaluate the performance of the Wiener filter an 'unknown' system is first created.

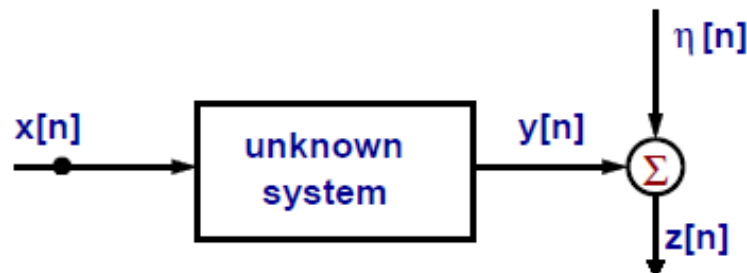


Fig 1. Illustration of the UNKNOWN system

The unknown system in this case is an FIR filter with order 5 (filter coefficient $b = [1, 2, 3, 2, 1]$ and $a = [1]$). The input signal $x[n]$ is a random white noise whereas the noise $\eta[n]$ used is also a random white noise with standard deviation of 0.1. The unknown system is simulated in MATLAB as follows:

```

x = rand(1000,1);           %input
b = [1 2 3 2 1]; a = [1];   %filter coeff
y = filtfilt(b,a,x);        %
y = y./sqrt(sum(b.*b));     %normalization
noise = 0.1*randn(1000,1);  %noise
z = y + noise;
SNR = 10*log10(sum(y.*y)/sum((y-z).*(y-z))); %SNR
  
```

The SNR of the system is calculated to be 19.734dB.

1.1 Estimation of the Cross Correlation and Auto Correlation Matrix

We first use the `xcorr` function to estimate the cross-correlation and auto-correlation vectors of the unknown system defined in the previous section.

To create the autocorrelation matrix R_{xx} we use the `toeplitz` function in MATLAB with the following command.

```

autocorr = xcorr(x);
c =
[autocorr(1000), autocorr(1001), autocorr(1002), autocorr(1003), autocorr(1004)];
r =
[autocorr(1000), autocorr(999), autocorr(998), autocorr(997), autocorr(996)];
acm = toeplitz(c,r);
  
```


And the cross-correlation vector is generated as:

```
crosscorr=xcorr(z,x);
pcm = crosscorr(1000:1004);
```

1.2 Estimation of the Cross Correlation and Auto Correlation Matrix

According to the lecture notes the optimal coefficients generated by the optimum wiener filter is given by the matrix product between the auto-correlation matrix and the cross-correlation matrix.

$$\mathbf{w}_{\text{opt}} = \mathbf{R}_{xx}^{-1} * \mathbf{p}_{xx}$$

This is implemented in MATLAB as follows:

```
w = inv(acm)*pcm;
w = w ./sqrt(sum(b.*b));
```

Which generated the following coefficient:

```
w =
0.9991
2.0027
2.9910
1.9865
0.9967
```

We can see that the filter coefficient predicted by the optimum wiener filter is almost the same as the coefficient of the unknown system for the given noise level.

1.3 Effect of noise on the Wiener solution

We now vary the scaling applied to the additive noise so that its variance takes 6 different values. We then will examine the effect of the level of noise on the Wiener solution

Noise Level (Variance)	SNR dB	Filter Coefficient					
0.2	6.7564	0.9797	1.9725	2.9756	1.9979	0.9554	
1	0.3879	1.0226	1.9355	2.9324	1.9404	1.2868	
2	-2.1293	1.0064	1.7131	2.7872	1.7204	0.8192	
5	-6.9102	1.6655	1.6716	2.5777	2.1598	0.9636	
8	-8.9988	0.9776	2.1860	3.3814	2.3251	1.0757	
10	-9.8694	0.6341	1.1781	3.0226	1.8520	0.9256	

Table 1. Filter coefficients with varying noise level

We can see that as the variance of the additive noise increases, the Wiener solution of the coefficient of the filter becomes less accurate.

If we assume N_w to be greater than 4 (taking 5 as an example) we will obtain the following results:

```
w =  
  
    0.9577  
    2.0242  
    3.0211  
    2.0299  
    1.0860  
   -0.0013
```

We can see that the additional filter coefficient generated is very close to 0. If we repeat this for an even higher $N_w = 7$:

```
W =  
  
    0.9447  
    1.9197  
    2.9608  
    2.0213  
    1.0980  
    0.1166  
    0.0190  
   -0.0253
```

The result is consistent as before.

1.4 Computational Complexity of the Wiener solution

Due to the block based nature of the Wiener solution, both the auto-correlation matrix and the cross-correlation vector needs to be calculated in order for the Wiener solution to be estimated. In the code segment shown above R_{xx} and p_{zx} are calculated using the MATLAB built in `xcorr` function.

According to the MATLAB documentation the `xcorr` function uses 2 ffts and 1 ifft and its complexity is of the order $N \log(N)$.

The computational complexity of finding the inverse of R_{xx} is of the order N^3 . For the multiplication between the auto-correlation matrix and the cross-correlation vector, N^2 multiplications and N^2-N additions are required. Overall, the computation of the Wiener solution is very computational intensive.

2. LMS Filter

The Wiener filter assumes that the unknown system is statistical stationary, however, in real life this is rarely the case. So we now investigate the LMS adaptive filter which has the capacity to estimate the Wiener solution on a sample by sample basis.

2.1 LMS algorithm

In this section we will develop a `lms` function which takes the input $x[n]$, noise corrupted output $z[n]$ (referring back to the Fig 1). The adaptation gain and the order of the adaptive filter as its input to calculate the LMS estimate $\hat{y}[n]$.

The output of this function gives the error vector and the matrix containing the evolution of \mathbf{w} (the Wiener solution).

In coursework 4, we have already designed a LMS adaptive filter system. Therefore in this lab, we will use a similar function as we have used in coursework 4. The MATLAB code is implemented as follows:

```
function [ e,w_out ] = lms( x,z,miu,order )
%LMS algorithm
%LMS script developed based on the sample script from MATLAB central
%available@
%http://www.mathworks.co.uk/matlabcentral/fileexchange/3649-lms-algorithm-
%demo
clear w;
b=[1 2 3 2 1]; %original filter coefficient
sysorder = order ; %filter order
sys_out = z; %z[n]
sys_in = x; %x[n]
totallength=size(sys_in,1);
for n = 1:(sysorder-1)
w(:,n) = zeros ( sysorder , 1 ) ; %initialize w filter coef all zeros
end
for n = sysorder : totallength
test = sys_in(n:-1:n-sysorder+1) ; %pickup x(n) based on the
filter order
y(n)= w(:,n-1)' * test; %estimated output
e(n) = sys_out(n) - y(n) ; %compare estimated output with
original
w(:,n) = w(:,n-1) + miu * test * e(n); %update the next gradient
vector
out(n) = w(:,n)' * test ; %calculate the output
end
plot(sys_in,'b'); hold on;
plot(out,'r');
title('Comparison between the original and LMS filtered music sample') ;
legend('Noisy Version','LMS Filtered Version');
xlabel('Samples')
ylabel('Amplitude')
w_out = w.*sqrt(sum(b.*b)); %normalization
end
```

Applying this function to the x and z signals used in section 1.1, we can track the evolution of w generated by this LMS adaptive filter.

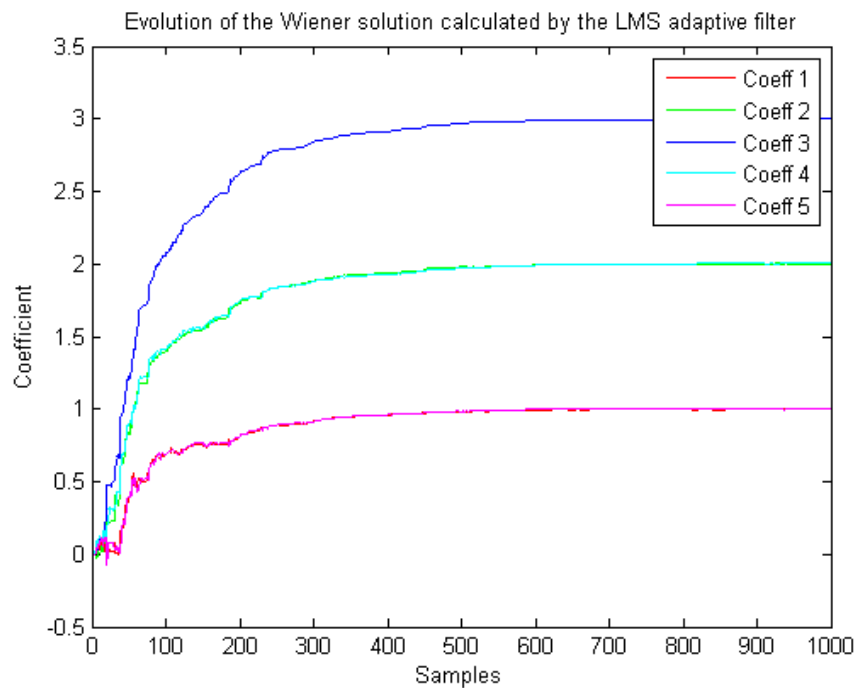


Fig 2. Evolution of the Wiener Solution as Calculated by the LMS filter

As we can see the value of w converges to the filter coefficient of the unknown system. i.e. $b = [1 \ 2 \ 3 \ 2 \ 1]$;

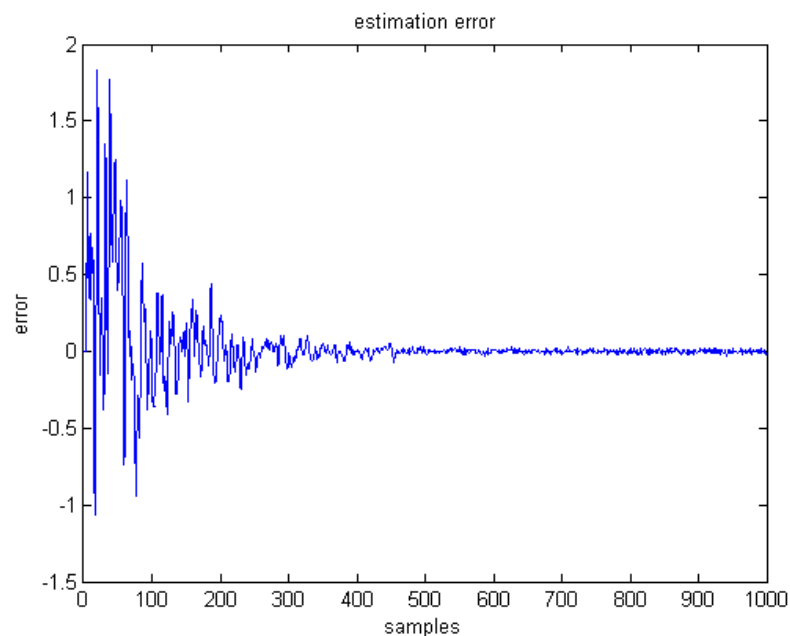


Fig 3. Evolution of the error term

We can also see that the estimation error decreases as the number of samples increase.

2.2 Effect of the Adaptation Gain on the Wiener Solution

We now investigate the effect of adaptation gain on the time evolution of the filter coefficient. We test the function that we have developed in the previous section by adjusting the adaptation gain μ .

The following graphs of w are plotted for $\mu = 0.002, 0.01, 0.1$ and 0.5

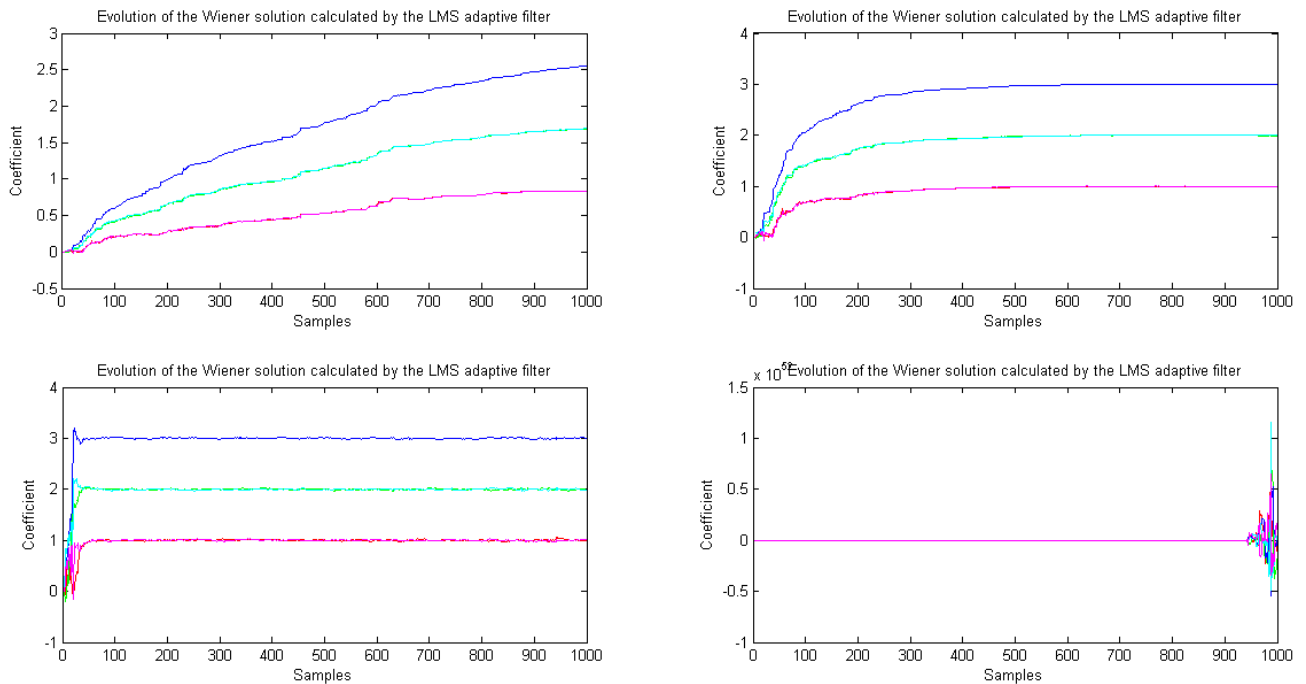


Fig 4. Effect of different adaptation gain on the coefficients

We can see that for larger μ , the filter coefficients converge faster to the coefficient defined for the unknown system.(Wiener Solution) However, if μ is too large as in the case of 0.5, then the filter coefficients will never converge.

2.3 Computational Complexity of the LMS Adaptive Filter

By observing the algorithm for the LMS adaptive filter written above, the LMS algorithm requires approximated $4N$ multiplications (N is the length of the input sequence) and $2N$ additions (subtractions) to calculate the filter coefficients of an unknown system ($6N$ computations in total) . Compared to the previous Wiener solution method which involves fft and matrix multiplication, the LMS adaptive algorithm is much more computationally efficient.

3. Gear Shifting

As we have seen from the section above, a large adaptation gain offers a quick adaptation, however a smaller adaptation gain offers a better tracking behavior at the steady state. Thus there exist a trade off between choosing different values of the adaptive gain for an LMS system. However, the gear shifting algorithm offers us an alternative. We can implement a time varying adaptive gain system so that both quick adaptation and small tracking error can be offered.

Method 1:

The adaptation gain is first set to a large value so that quick adaptation can be achieved. Then by analyzing the error of the n^{th} term, the adaptation gain is adjusted for the $n+1^{\text{th}}$ term. The criteria for the adjustment of the adaptation gain is so that if the error is greater than 15% of the original output, then the adaptation gain is reduced by 10%. This algorithm is implemented in MATLAB as follows:

```

miu = 0.25;           %initialization of miu

if abs(e(n)/sys_out(n)) > 0.15      %check if miu needs to be reduced

    miu = miu * 0.9;

```

When this method for adjusting the adaptation gain is used, MATLAB generated the following result:

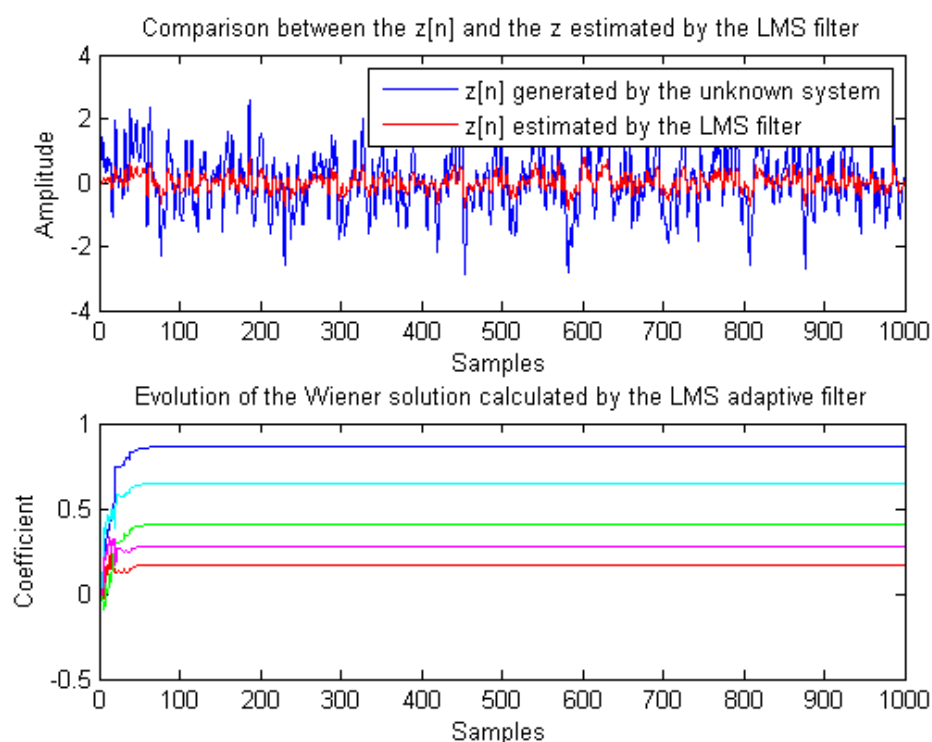


Fig 5. Results for Gear Shifting Method 1

We can see that the $z[n]$ estimated by the LMS adaptive filter did not track the original $z[n]$ produced by the unknown system, this is because despite the large starting value for the adaptation gain the μ decreased too fast causing the filter coefficients not to converge to the correct values. Hence, this method obviously does not work.

Method 2:

In method 1, the adaptation gain is decreased by 10% every time the error exceed the original by 15%. However, the speed of reduction causes μ to become too small which in turn causes the filter coefficients not to converge to the correct value. To improve this, we can change the reduction to 2% to prevent the adaptation gain from decreasing too fast.

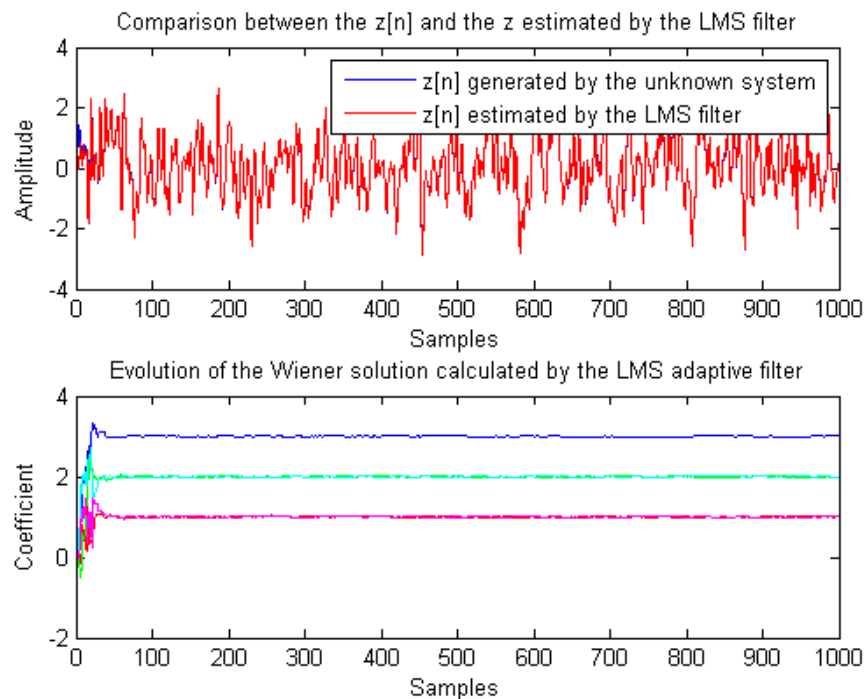


Fig 6 Results for Gear Shifting Method 2

We can see that with this algorithm implemented the estimated $z[n]$ is now able to correctly track the $z[n]$ generated by the unknown system. We can also see that due to the large value of μ used, the adaptation to the correct filter coefficients is very fast.

We now compare this to the original algorithm (no gear shifting) with the same μ .

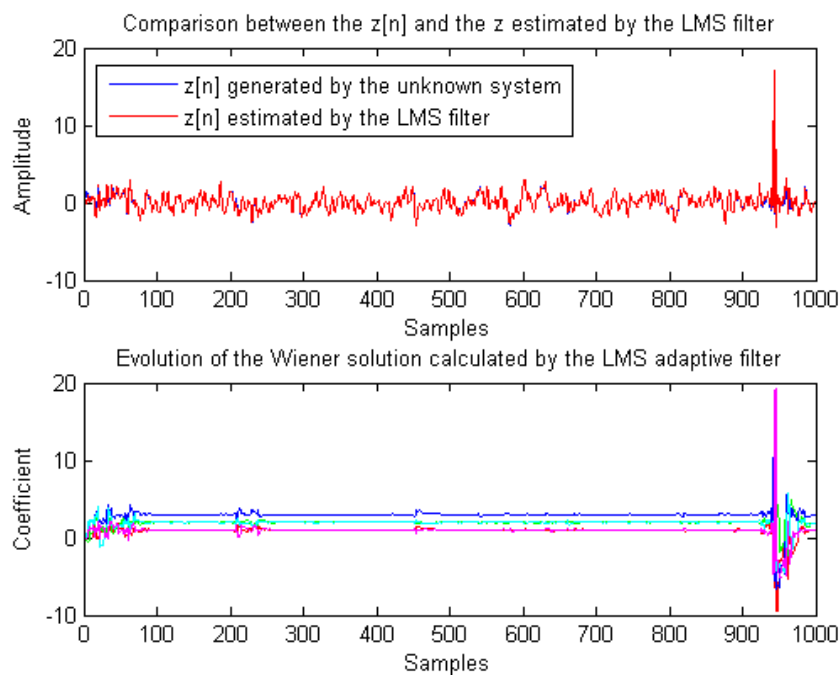


Fig 7 Results for Adaptation gain = 0.25 with no gear shifting

We can see that for the same adaptation gain but without gear shifting, the tracking of the output at the steady state is severely affected. This shows that the gear shift algorithm is indeed capable of producing both quick adaptation and a small steady state tracking error.

We now compare the performance of the two algorithms and demonstrate the merit of the gear shifting algorithm by further increasing μ :

For $\mu=0.5$:

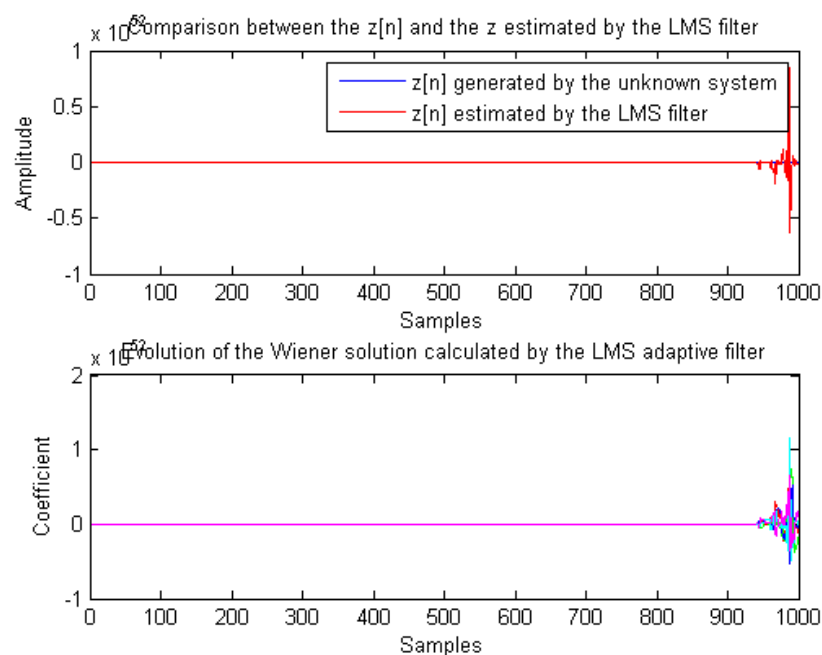


Fig 8 Results for Adaptation gain = 0.25 with no gear shifting

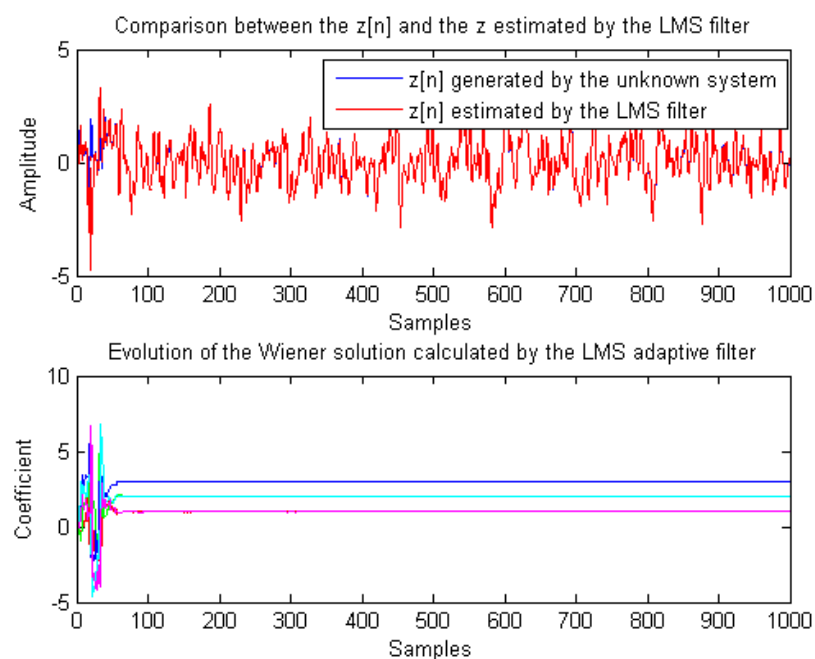


Fig 9 Results for Adaptation gain = 0.25 with gear shifting

4. Identification of AR processes

In this section we will implement the adaptive estimation of a speech recognition system according to the system illustrated below. The system consists of two parts synthesis and analysis. In the synthesis section $\eta[n]$ represents a white noise whereas $x[n]$ represents the result after $\eta[n]$ passes through an AR process with coefficients $a = [1 \ 0.9 \ 0.2]$ and $b = [1]$. In the analysis section there exists a LMS adaptive system trying to dynamically estimate the parameters of the AR system.

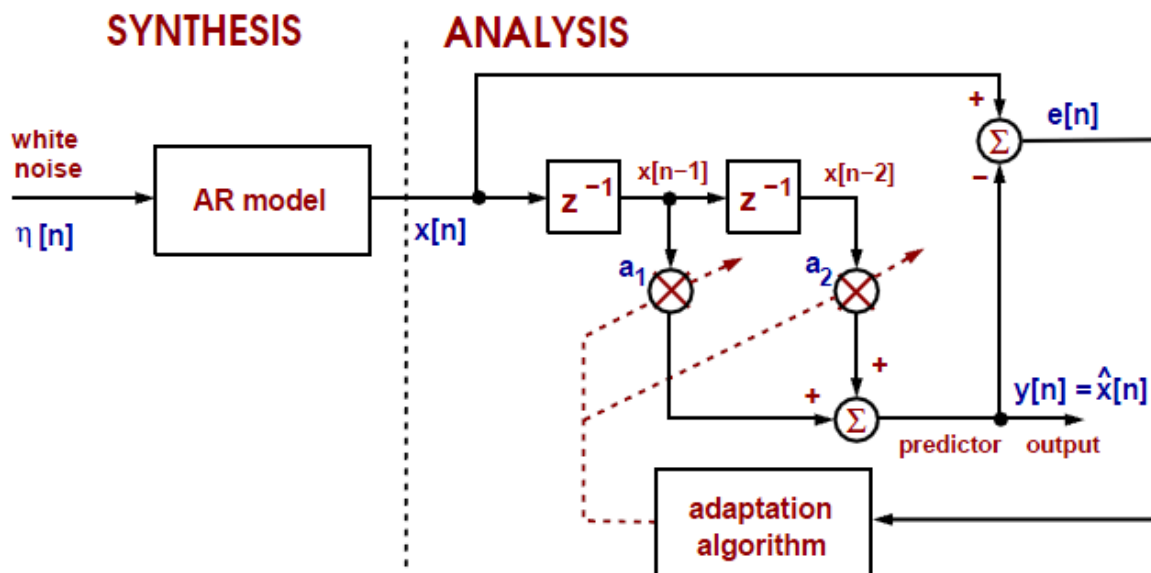


Fig 10 Illustration of a speech identification system

4.1 Estimation of the AR coefficients using the LMS Adaptive Filter

As we can observe from the above diagram, different from the LMS systems we have encountered before, in this particular system, both the input and the error calculation requires the same $x[n]$ sequence and the output of the LMS adaptive system is an estimate of the input. Thus we have to make some changes to our original LMS function to accommodate for these changes.

For our AR system $x[n]$ is given by

$$x[n] = 0.9x[n-1] + 0.2x[n-2] + \eta[n]$$

From the above diagram we can observe that $y[n]$

$$y[n] = 0.9x[n-1] + 0.2x[n-2] + \eta[n] + a_1x[n-1] + a_2x[n-2]$$

The condition for $y[n]$ to equal $x[n]$ is such that $a_1 = -0.9$ and $a_2 = -0.2$ in addition we have to remove the noise term $\eta[n]$. With this knowledge in mind, we can now implement the modified LMS estimator in MATLAB.

```

x = randn(1000,1);      %input
b = [1]; a = [1 0.9 0.2]; %filter coeff
y = filter(b,a,x);      %Pass through the AR system
[e,w] = lms3(y,y,x,2);   %use LMS to estimate coefficients

```

```

function [ e,w ] = lms3( x,z,whitenoise,order )
%LMS algorithm

clear w;
sysorder = order ;           %filter order
sys_out = z;                 %z[n]
sys_in = x;                  %x[n]
totallength=size(sys_in,1);
miu = 0.01;
for n = 1:(sysorder)
w(:,n) = zeros ( sysorder , 1 ) ; %initialize w filter coef all zeros
end
for n = sysorder+1 : totallength
    test = sys_in(n-1:-1:n-sysorder) ; %pickup x(n) based on the filter
    order
    y(n)= w(:,n-1)' * test; %estimated output
    e(n) = sys_out(n) - y(n) - whitenoise(n); %compare estimated output with
    original
    w(:,n) = w(:,n-1) + miu * test * e(n); %update the next gradient vector
    out(n) = w(:,n)' * test ; %calculate the output

end
subplot(2,1,1);
plot(sys_in-whitenoise,'b'); hold on;
plot(out,'r');
title('Comparison between the original and LMS filtered estimated x[n]') ;
legend('x[n]','estimated x[n]');
xlabel('Samples')
ylabel('Amplitude')

subplot(2,1,2);
plot(w(1,:), 'r');hold on;
plot(w(2,:), 'g');hold on;
legend('Coeff 1','Coeff 2');
title('Evolution of the AR coefficient calculated by the LMS adaptive
filter') ;
xlabel('Samples')
ylabel('Coefficient')
end

```

After executing the LMS function, we obtain the following results:

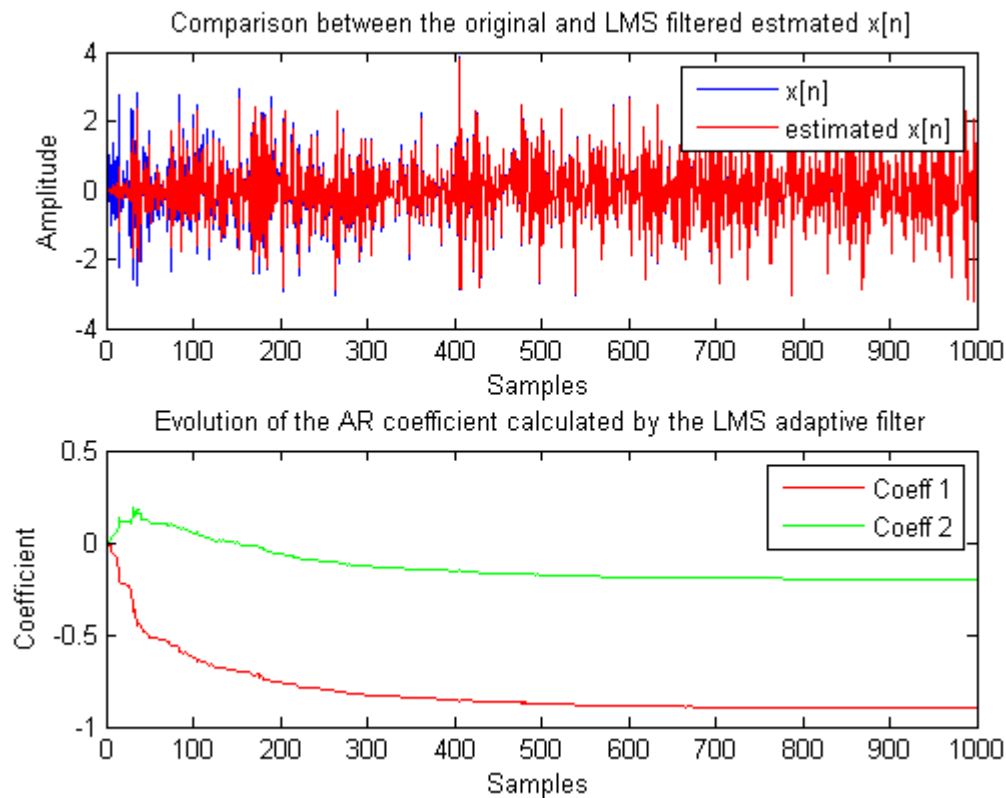
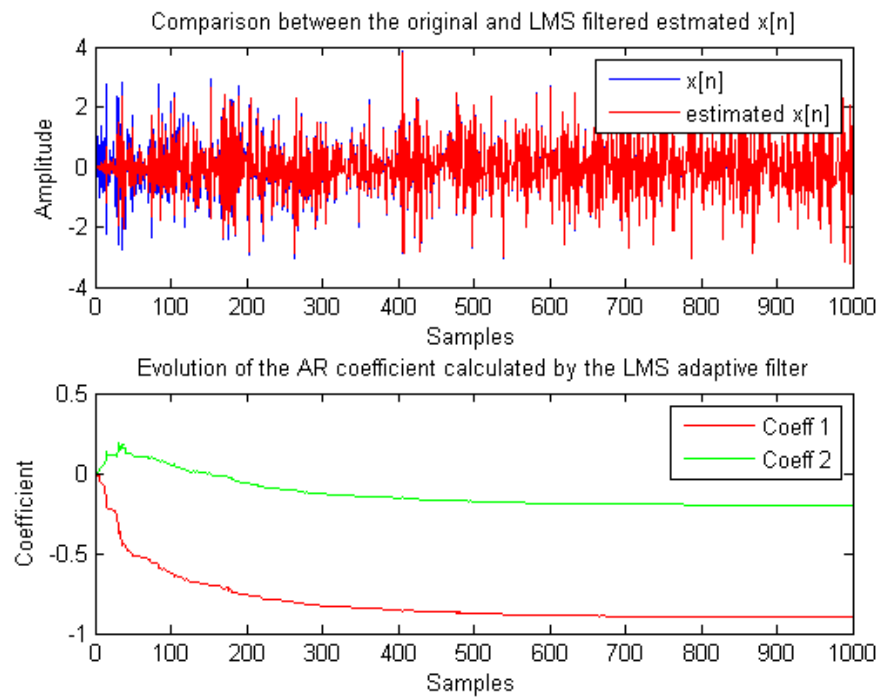
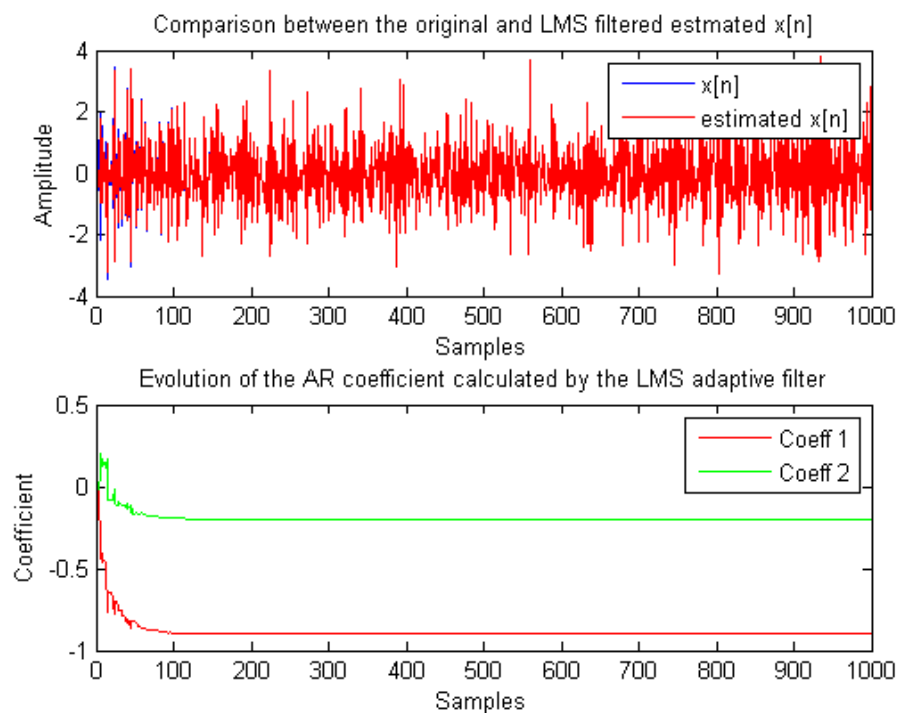


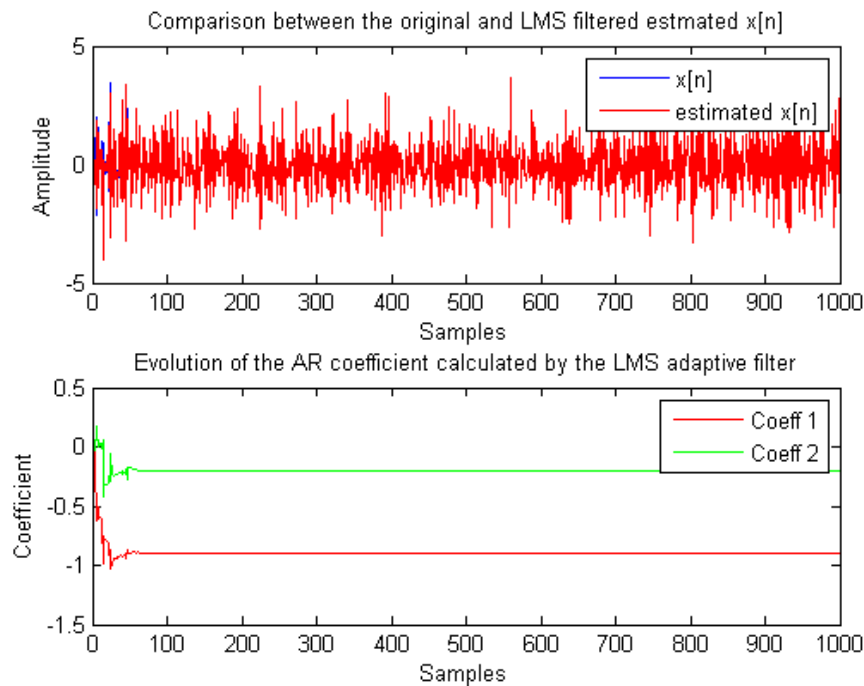
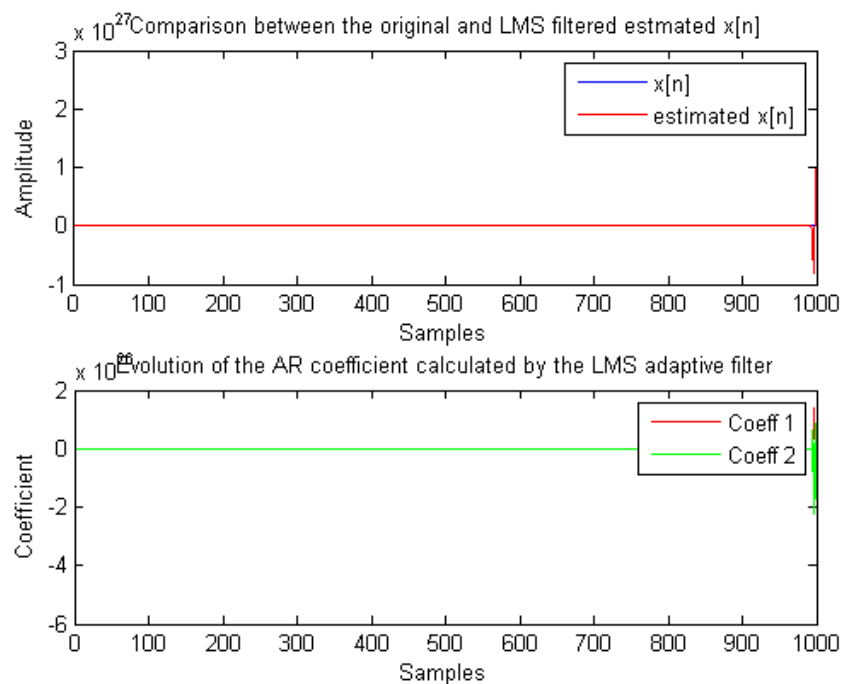
Fig 11. Evolution of the coefficients of the AR system

We can see that the coefficients of the AR system approaches -0.2 and -0.9 as what we have expected before. This demonstrates that the LMS estimator functions as we expected.

4.2 Effect of adaptive gain on the evolution of AR coefficients

Now, by varying the adaptive gain μ , we observe the effect of the adaptation gain on the evolution of the AR coefficients:

Fig 12. Evolution of AR coefficient for $\mu=0.01$ Fig 13. Evolution of AR coefficient for $\mu=0.05$

Fig 14. Evolution of AR coefficient for $\mu=0.1$ Fig 15. Evolution of AR coefficient for $\mu=0.4$

We can observe that as before a large adaptation gain offers quick adaptation, whereas a small adaptation gain offers better tracking in the steady state. The gearing shifting algorithm can also be applied here for an improved performance.

5. Speech recognition

5.1 Quality of the predictor

The right hand side analysis block of figure 10 will be used for speech recognition. The input speech signal will be of my own voice pronouncing the letters 'e' 'a' 's' 't' and 'x'. The sampling frequency will be set at 44100Hz and the initial recording will contain 100000 samples (about 2.26 seconds of recording). Then, by observing the waveform of samples 1000 samples will be taken as the test signal. (recoding is done using the wavrecord function i.e. `a = wavrecord(100000,Fs,'double');`)

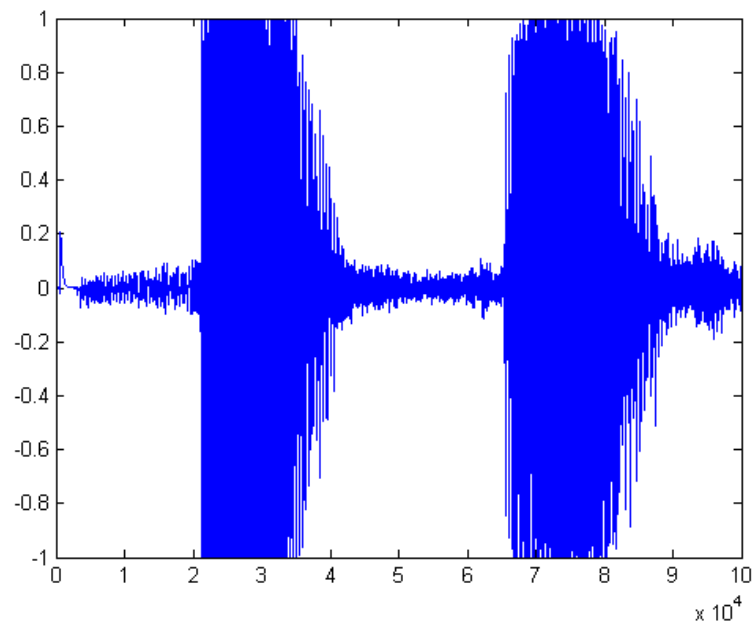


Fig 16. Recording of sound 'a'

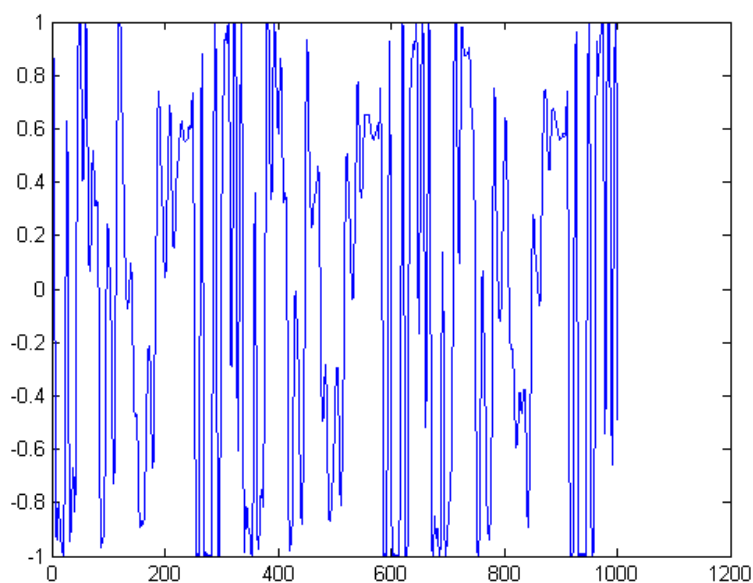


Fig 16. 1000 samples taken from the Recording of sound 'a'

To assess the prediction gain for the different sound files, the following MATLAB script is used:

```
[e, w] = lms4(e_sound,e_sound,2,0.05);

10*log10(var(e_sound)/var(e))    %assess the quality of sound e

[e, w] = lms4(a_sound,a_sound,2,0.05);

10*log10(var(a_sound)/var(e))    %assess the quality of sound a

[e, w] = lms4(s_sound,s_sound,2,0.05);

10*log10(var(s_sound)/var(e))    %assess the quality of sound s

[e, w] = lms4(t_sound,t_sound,2,0.05);

10*log10(var(t_sound)/var(e))    %assess the quality of sound t

[e, w] = lms4(x_sound,x_sound,2,0.05);

10*log10(var(x_sound)/var(e))    %assess the quality of sound x
```

Adaptation gain = 0.05 order = 2

And we obtained the following results:

Sounds	Prediction Gain
E	12.6396dB
A	10.2135dB
S	5.2332dB
T	7.2105dB
X	8.7124dB

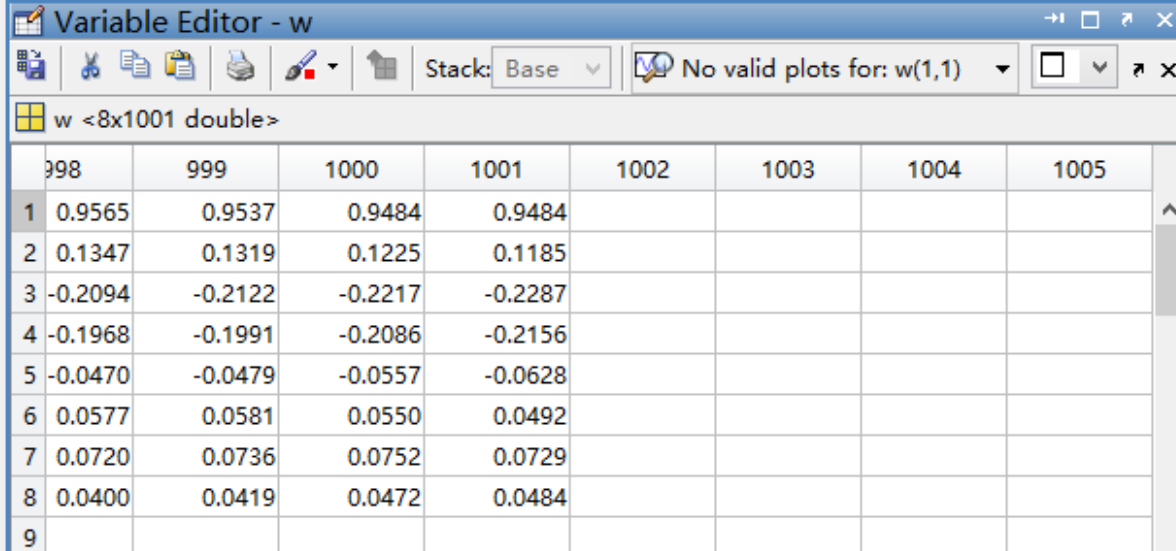
Table 2. Quality of the Predictor for different sounds

We observe that the Quality of the Predictor R_p is much higher for vowels than for consonants. we have also observed by changing the order of the predictor, there is little effect on the Quality of the Predictor.

Predictor Order	Prediction Gain
2	10.2135dB
4	10.6719dB
5	10.9724dB
8	10.9362dB

Table 3. Quality of the Predictor for sound 'a' at different orders

However, by observing the evolution of the filter coefficients using a large order, we can see that the optimum order for sound 'a' is 4. As the filter coefficients after the 4th are very small.



	998	999	1000	1001	1002	1003	1004	1005
1	0.9565	0.9537	0.9484	0.9484				
2	0.1347	0.1319	0.1225	0.1185				
3	-0.2094	-0.2122	-0.2217	-0.2287				
4	-0.1968	-0.1991	-0.2086	-0.2156				
5	-0.0470	-0.0479	-0.0557	-0.0628				
6	0.0577	0.0581	0.0550	0.0492				
7	0.0720	0.0736	0.0752	0.0729				
8	0.0400	0.0419	0.0472	0.0484				
9								

Fig 17. Evolution of filter coefficients for order 8

By varying the adaptation gain, we can also test the effect of the different adaptation gain on the Prediction Gain. If the adaptation gain is small, as we have discussed before, the filter coefficients don't converge fast enough. This will result to a low quality of the predictor. However, if the adaptation gain is too large, then there will be poor tracking in the steady state. This will also cause the prediction gain to decrease. To tackle this compromise, gearing algorithm can be used again. With gearing algorithm in place we can increase the adaption gain to up to 1 and obtain a large prediction gain.

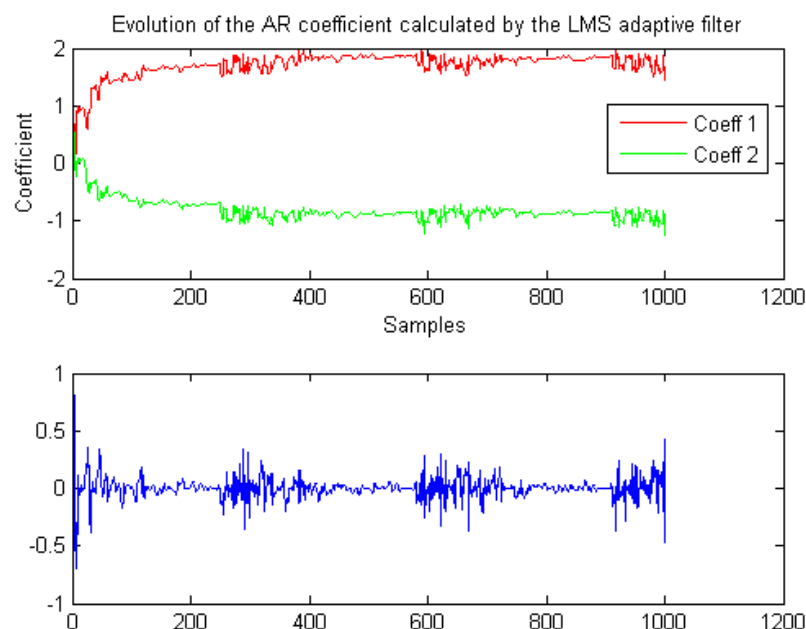


Fig 18. Evolution of filter coefficients and estimation error for adaptation gain = 0.9

In this particular implementation, the Prediction Gain obtained is 17.0813dB.

5.2 Prediction Gain for Sampling Frequency = 16000Hz

The sound samples are recorded again using an 16000Hz sampling rate. First 50000 samples are recorded and 1000 samples are taken from the 50000 samples

```
a = wavrecord(50000,Fs,'double');
plot(a)
a_sound = a(10000:11000);
```

Then by using the same MATLAB script we can test the Prediction Gain for the sound recorded at 16000Hz with the same parameters as in the previous case.

```
[e, w] = lms4(e_sound,e_sound,2,0.05);
10*log10(var(e_sound)/var(e))    %assess the quality of sound e

[e, w] = lms4(a_sound,a_sound,2,0.05);
10*log10(var(a_sound)/var(e))    %assess the quality of sound a

[e, w] = lms4(s_sound,s_sound,2,0.05);
10*log10(var(s_sound)/var(e))    %assess the quality of sound s

[e, w] = lms4(t_sound,t_sound,2,0.05);
10*log10(var(t_sound)/var(e))    %assess the quality of sound t

[e, w] = lms4(x_sound,x_sound,2,0.05);
10*log10(var(x_sound)/var(e))    %assess the quality of sound x
```

Sounds	Prediction Gain
E	8.4613dB
A	7.1445dB
S	3.5920dB
T	4.9063dB
X	5.8372dB

Table 4. Quality of the Predictor for different sounds

As we can see from the table of results, for a lower sampling frequency, the Prediction Gain decreased by a large proportion. However, the prediction gain for the vowels are still higher than the prediction gain for the consonants.

6. Dealing with computational complexity – sign algorithms

When estimating the adaptation weights we used the following algorithm

$$w(n+1) = w(n) + \mu * e(n) * x(n)$$

but for good hardware and high speed applications we can simplify the above equation and apply the sign algorithm.

There are three possible sign algorithms:

Signed error - $w(n+1) = w(n) + \mu * \text{sign}(e(n)) * x(n)$

Signed regressor - $w(n+1) = w(n) + \mu * \text{sign}(e(n)) * x(n)$

Sign sign - $w(n+1) = w(n) + \mu * \text{sign}(e(n)) * \text{sign}(x(n))$

We now evaluate these three algorithms for the AR parameter identification in section 4. We first start with an adaptation gain of 0.04.

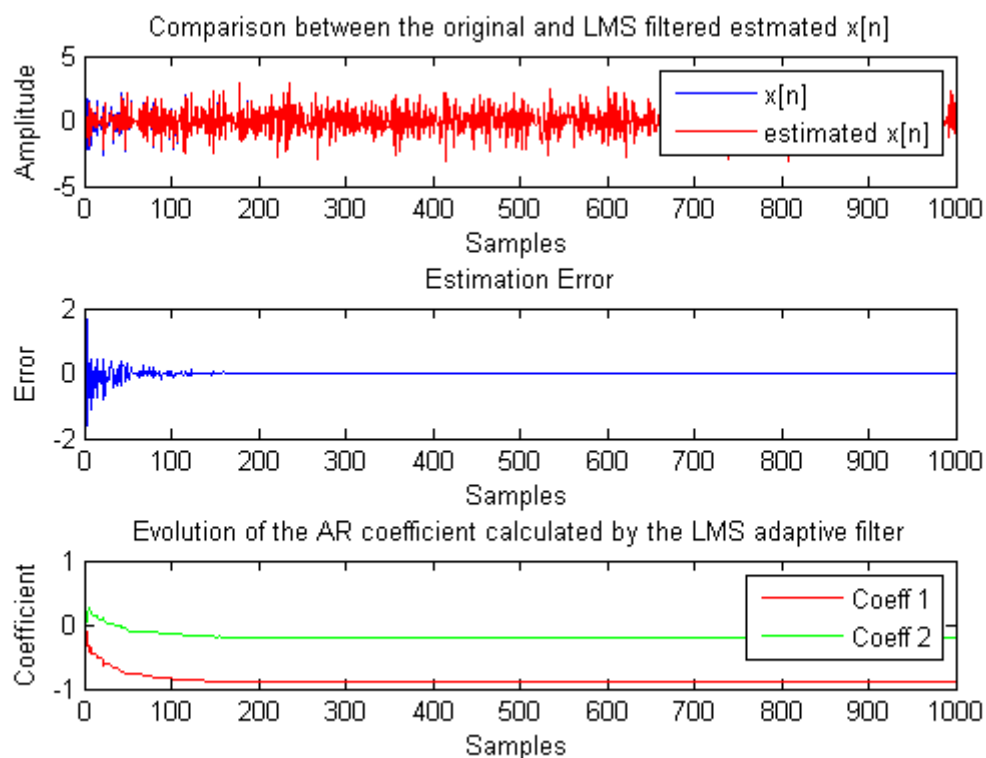


Fig 19. AR parameter identification using the original algorithm

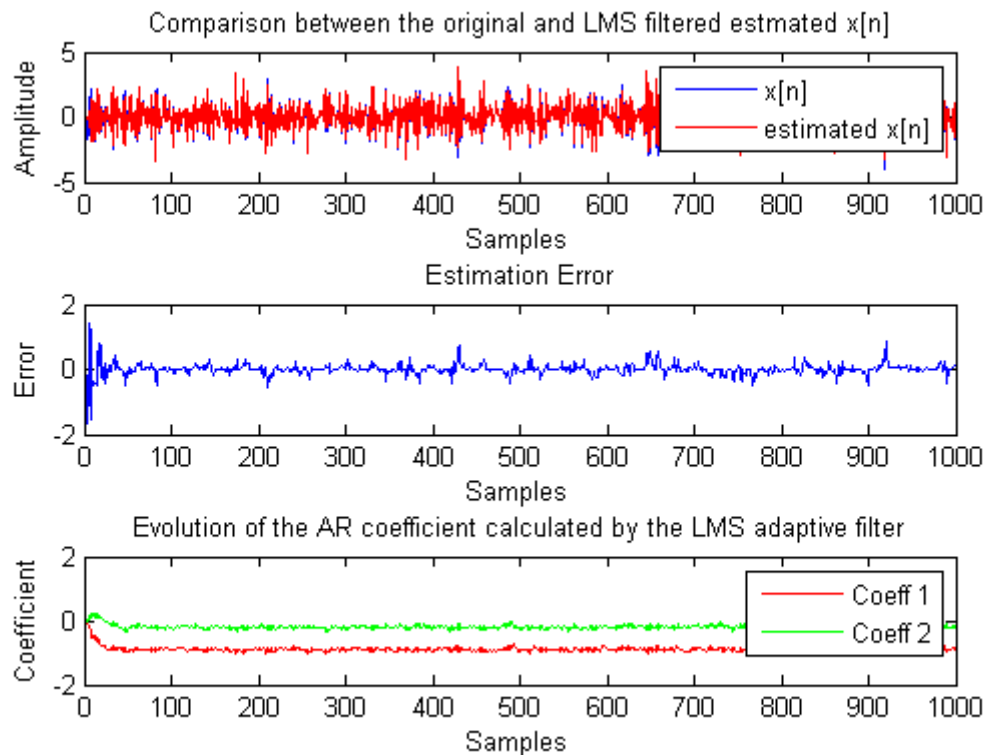


Fig 20. AR parameter identification using the signed error algorithm

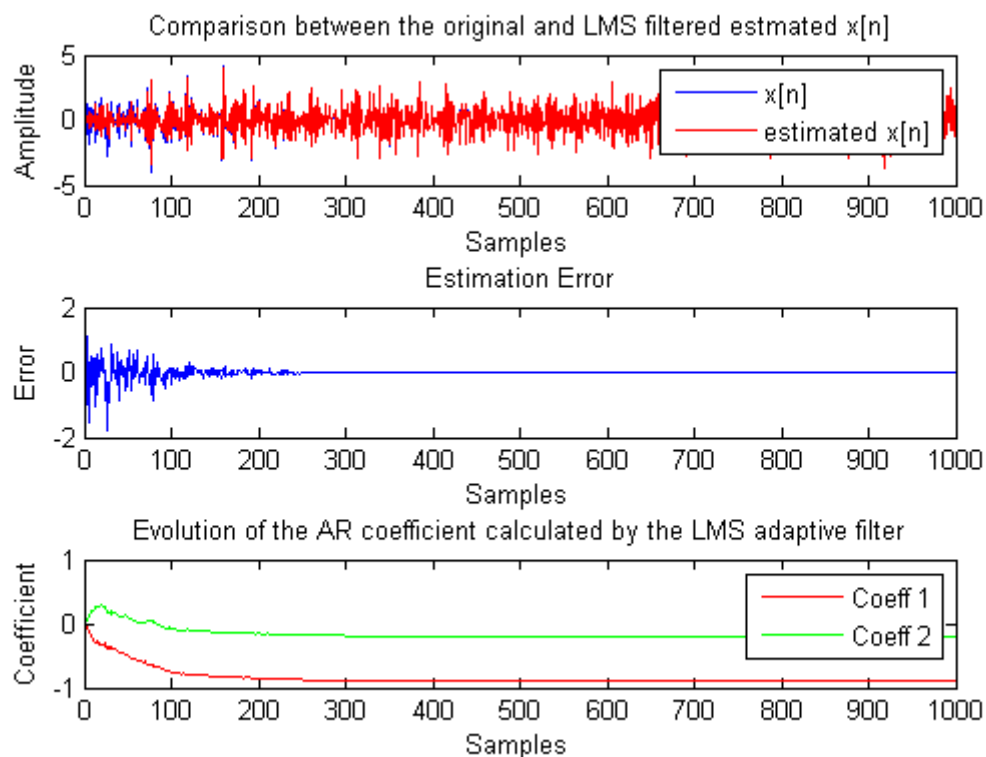


Fig 21. AR parameter identification using the signed regressor

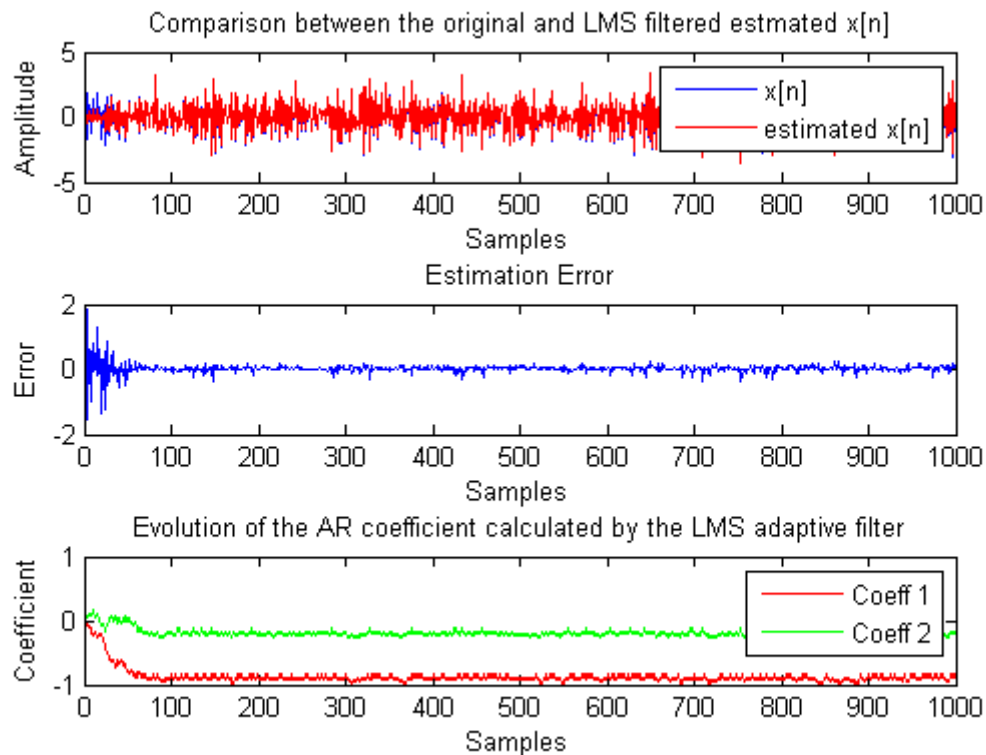


Fig 22. AR parameter identification using the sign sign algorithm

From the four different algorithms, we can observe the following. The Signed error algorithm does not offer a good tracking in the steady state this causes large errors in the output. The signed regressor algorithm offers a steady state tracking, but the speed of convergence of the algorithm is much slower when compared to the original. The sign sign algorithm combines the feature of the two other sign algorithms. It offers an improved speed of convergence and a lowered steady state error.

We now increase the adaptation gain to 0.08

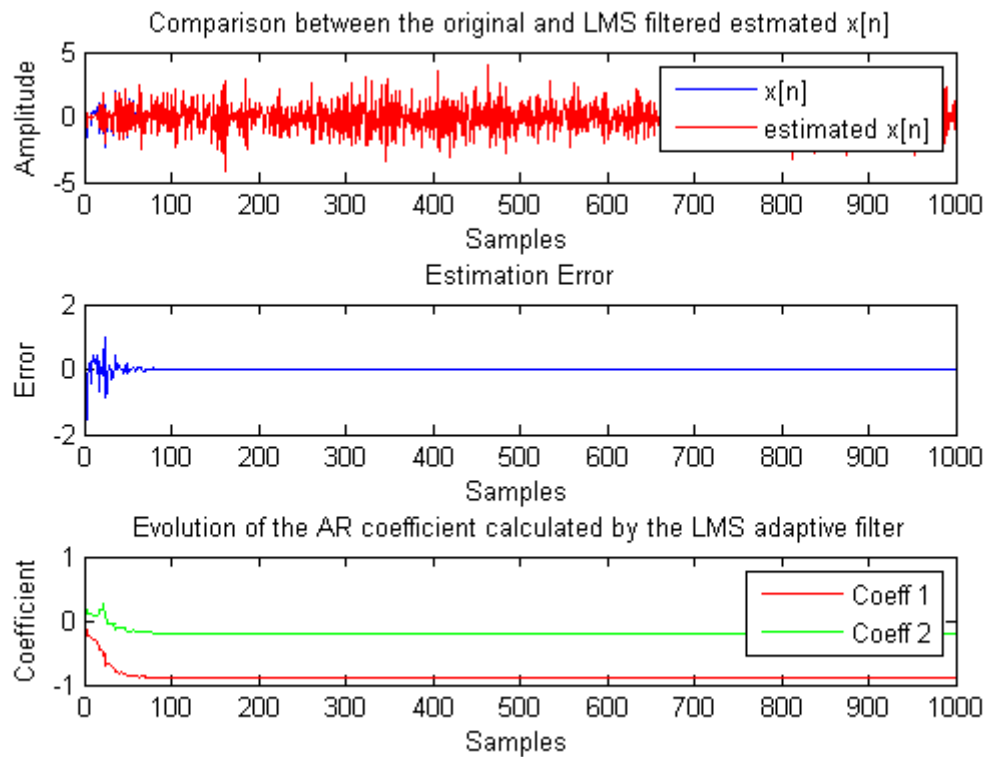


Fig 23. AR parameter identification using the original algorithm for adaptation gain = 0.08

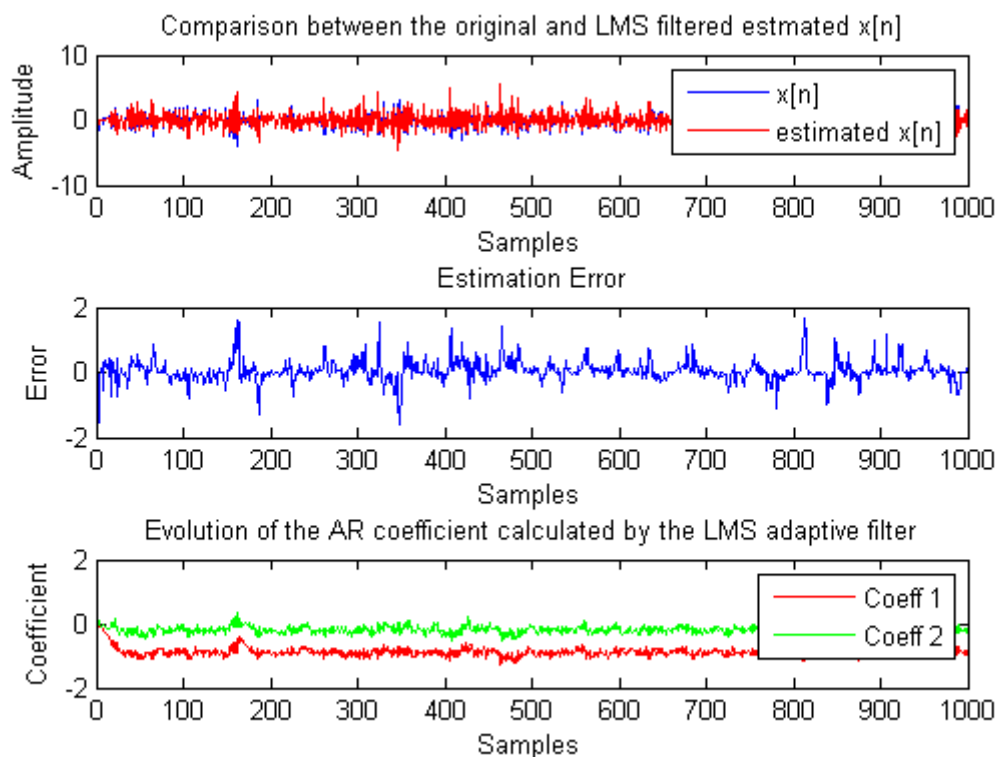


Fig 24. AR parameter identification using the signed error for adaptation gain = 0.08

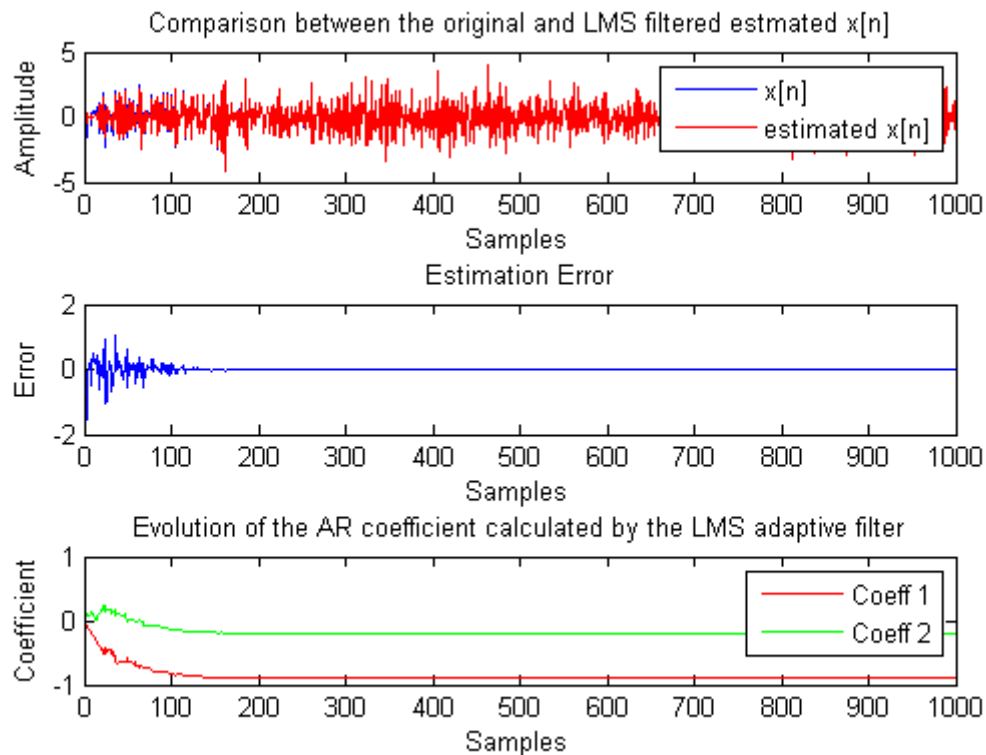


Fig 25. AR parameter identification using the signed regressor for adaptation gain = 0.08

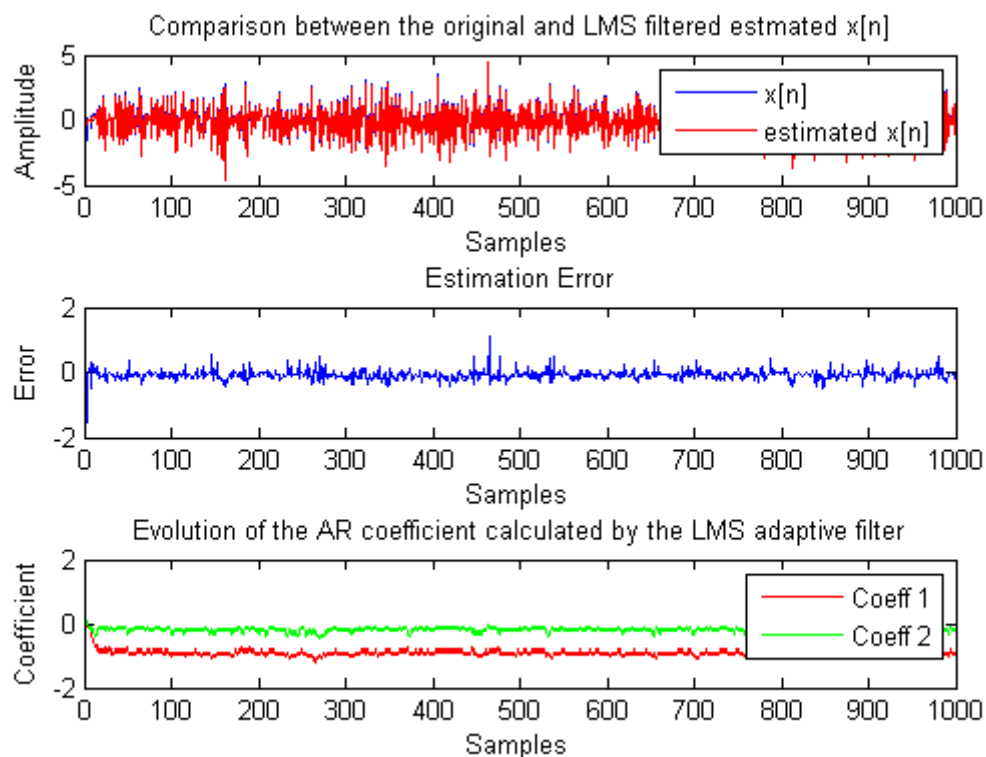


Fig 26. AR parameter identification using the signed signed for adaptation gain = 0.08

When the adaptation gain is increased, the speed of convergence for all algorithms also increased, however, for the signed error and signed error algorithm. The steady state error also increased significantly. If we are to increase the adaptation gain further, this trend would continue.

Now we apply the signed algorithms to two of the speech files we created in section 5.

The first example is the 'a' sound and the following parameters are used:

Adaptation gain: 0.2 Predictor order: 2 sampling frequency: 16kHz

Algorithm	Prediction Gain
Original	8.3096dB
Signed Error	7.4991dB
Signed Regressor	8.2132dB
Signed Signed	6.2595dB

Table 5. Performance comparison for sound a

The second example is the 't' sound and the following parameters are used:

Adaptation gain: 0.2 Predictor order: 2 sampling frequency: 16kHz

Algorithm	Prediction Gain
Original	1.4033dB
Signed Error	0.9830dB
Signed Regressor	1.3515dB
Signed Signed	0.0188dB

Table 6. Performance comparison for sound t

From the above comparisons, it can be clearly seen that the signed algorithm offers a degraded performance when compared to the original LMS algorithm. Out of all the signed algorithms, the signed regressor is believed to have the best performance when compared to the original algorithm. However, by varying the adaptation gain, we have also observed that the signed regressor algorithm has a slower rate of convergence. Hence, if there are no restraints on hardware/computational complexity the original LMS is recommended.

REFERENCES:

Professor D. Mandic, Advance Signal Processing Lecture Notes, 2013, Imperial College London.

Professor D. Mandic, Advance Signal Processing Coursework Handout, 2013, Imperial College London.

Monson H. Hayes, Statistical Signal Processing and Modeling, 1996 Edition

Mathworks, MATLAB Documentation 2011b, Cambridge MA.