Advanced Signal Processing

Sebastian Grubb
sg3510@ic.ac.uk

March 2013

Contents

Part 1

Random Signals and Stochastic Processes

1.1 Statistical estimation

1.1.1 Expected Value

To generate a 1000 random values and calculate de expectation in a vector in matlab we use the following code:

```
x = rand(1000,1); %generates 1000 uniformly distributed random ...
    values vector

a=0;
for i=1:1000
    a = a+x(i);
end
a=a/1000;
disp(a)
mean(x)
stem(x)
```

Some sample values obtained are:

mean(x)	Manually calculated
0.4958	0.4958
0.4948	0.4948
0.5118	0.5118
0.4984	0.4984
0.5127	0.5127
0.4867	0.4867

As we can see both values are consistent and to the same degree of accuracy. Increasing the decimal places to 20 also led to both functions to have the same accuracy and precision (i.e. they give out the same number).

1.1.2 Sample standard deviation

To calculate the standard deviation we use the following script:

```
x = rand(1000,1);

a=0;
m=mean(x);
for i=1:1000
    a = a+(x(i)-m)^2;
end
a=a/999;
a=sqrt(a);
sprintf('%.17f',a) %17 digits is maximum accuracy in matlab
sprintf('%.17f',std(x))
```

std(x)	calculated
0.284984	0.284984
0.279909	0.279909
0.289126	0.289126
0.288586	0.288586
0.287258	0.287258
0.286541	0.286541

Table 1.1: Calculated Standard Deviation vs obtained standard deviation

Some values calculated for various sets are found in Table 1.1.

Again both the custom defined function and the internal matlab function return the same values to the same decimal accuracy (with matlab giving a maximum of 17 decimal places).

1.1.3 Bias estimation

To calculate the bias we use the following script:

```
a=0;
for i=1:10
x = rand(1000,1);
m=mean(x);
a(i) = 0.5-m; %E(X) = 0.5 for N¬U(0,1)
end
stem(a) %plot the bias
```

Which resulted, for a certain run of it, in Figure 1.1.

It can be seen that the randomly generated samples all tend towards 0, clustering around their theoretical values.

1.1.4 Probability density function

A pdf was approximated using the following function:

```
a=0;
x = rand(1000,1);
a = hist(x)./1000;
b=.1:0.1:1
bar(b,a)
```

From this Figure 1.2 was obtained, which shows an expected bar graph, with each value having a probability of 0.1 of occurring.

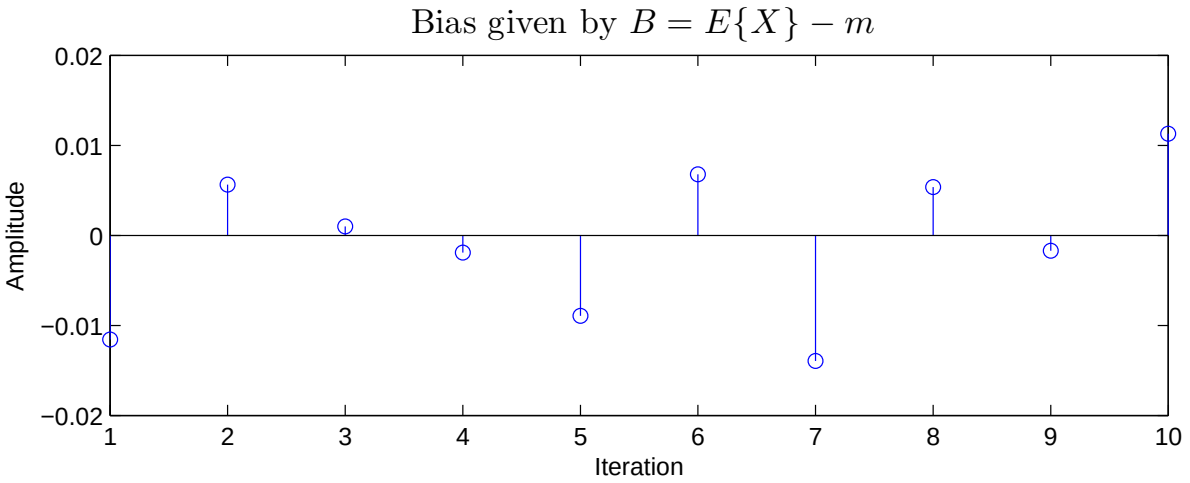


Figure 1.1: Bias of 10 different iterations

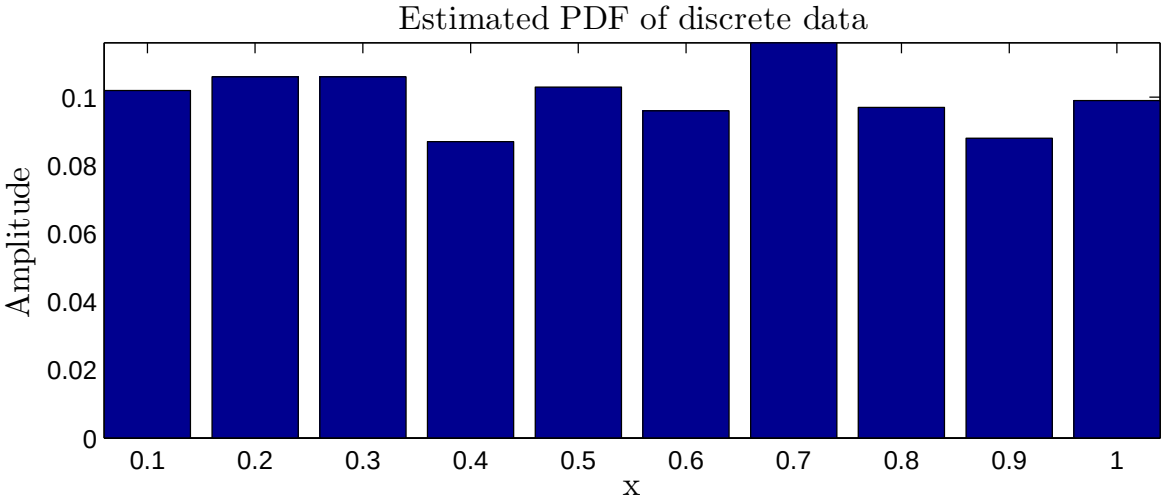


Figure 1.2: Discrete PDF of a set of 1000 uniformly generated samples

1.1.5 With Gaussian variable

Repeating steps 1-4 for a Gaussian distributed variable we find the mean to be around 0, with observed values including: -0.020069, -0.027927 and 0.011566. The standard deviation is, as expected around 1, with observed values including: 0.965608, 0.978326 and 1.041055.

Figure 1.3 provides an insight into bias estimation of various iterations. As we can see the values gravitate towards the expected value, which is 0.

A probability density function can be estimated using the hist function and gives us a distribution similar to that of its theoretical Gaussian distribution function. An example is provided in Figure 1.4.

The code used to calculate was the following:

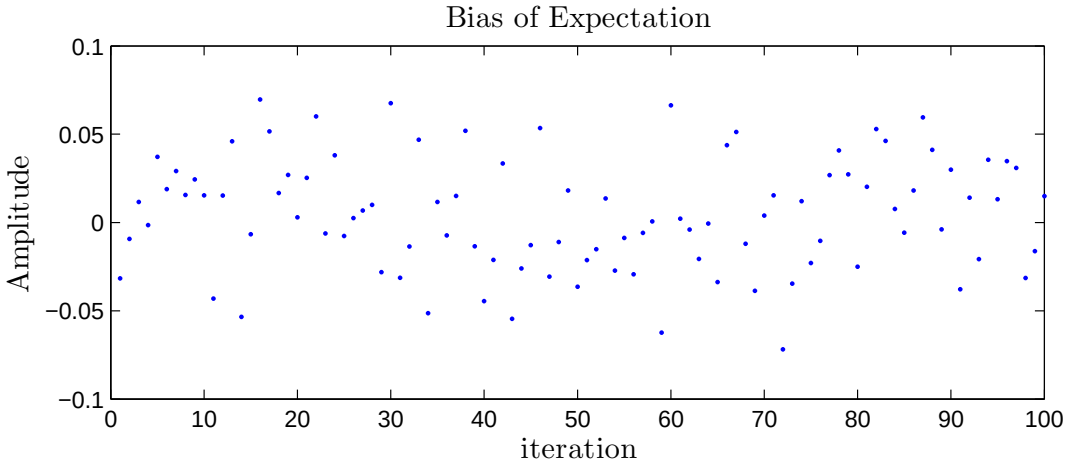


Figure 1.3: Bias of 100 iterations of discrete Gaussian data.

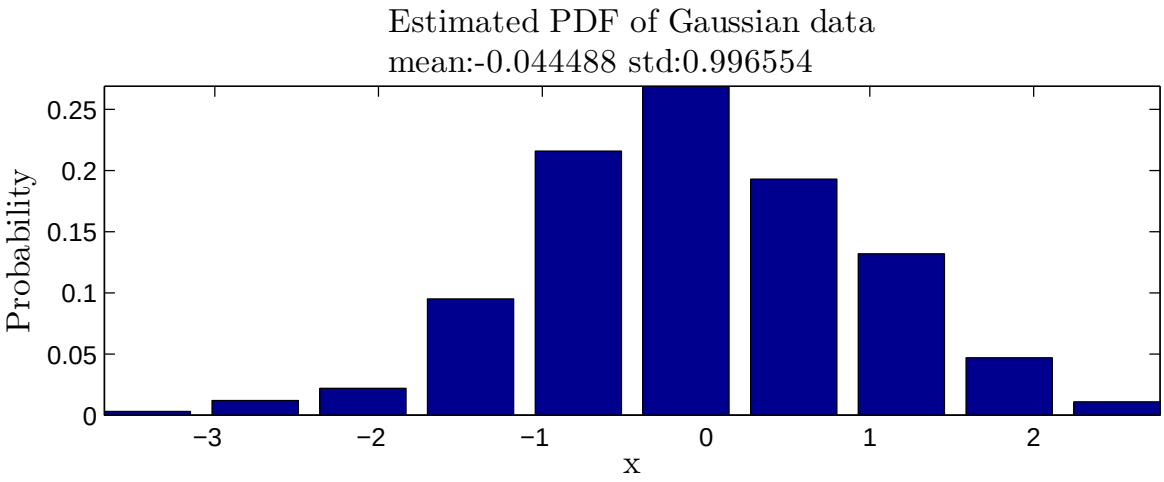


Figure 1.4: Estimated PDF of discrete Gaussian data.

```
%% 1.1.5 Gaussian Variable
x = randn(1000,1);
[a,b] = hist(x);
a = a./1000;
bar(b,a)
str = 'Estimated PDF of Gaussian data';
str = [str sprintf('\textbackslash nmean:%.6f',mean(x))];
str = [str sprintf('  std:%.6f',std(x))];
title(str,'interpreter','latex','fontsize',13)
ylabel('Probability','interpreter','latex','fontsize',13)
xlabel('x','interpreter','latex','fontsize',13)
axis tight;
for i=1:100
    x = randn(1000,1);
    a(i) = mean(x);
end
figure(2)
plot(a,'.')
title('Bias of Expectation','interpreter','latex','fontsize',13)
ylabel('Amplitude','interpreter','latex','fontsize',13)
xlabel('iteration','interpreter','latex','fontsize',13)
```

1.2 Stochastic processes

1.2.1 Ensemble mean and standard deviation for M,N = 100

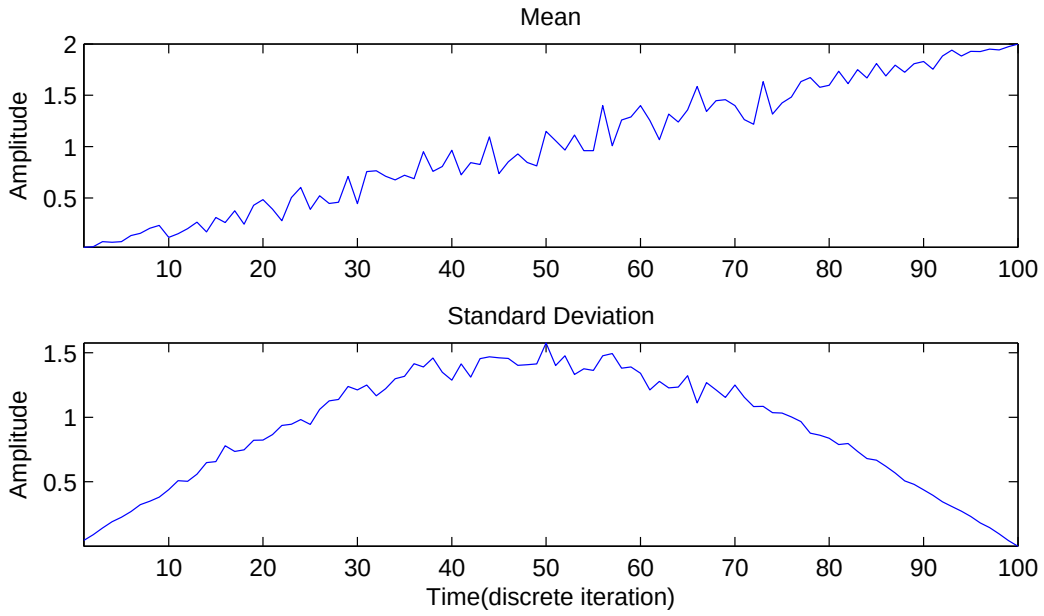


Figure 1.5: Mean and Standard Deviation over time for Process rp1

Figure 1.5 shows the evolution of the ensemble mean and standard deviation evolution over time for the rp1 stochastic process.

From this we can observe that the rp1 process is not stationary as its mean and standard deviation change when shifted in time. Indeed the mean increases linearly over time with a trend line $mean = \frac{\text{time sample \#}}{50}$ for this particular case. The standard deviation does not have a linear trend but rather a trend line best approximated by equation 1.1.

$$s.d. = 1.44 \times \sin(\frac{\text{time sample \#}}{\text{time sample total}} \times \pi) \tag{1.1}$$

Again this is not a constant and depends on time thus we conclude rp1 is a non-stationary process.

Figure 1.6 shows the ensemble mean and standard deviation plotted over time for the rp2 process.

We can notice that the rp2 process is stationary with a constant expected mean (in this case just above 0.5) and standard deviation of $\frac{1}{3}$.

Figure 1.7 shows the ensemble mean and standard deviation plotted over time for the rp3 process.

The rp3 process is also stationary with an expected mean of 0.5 and expected standard deviation of .86.

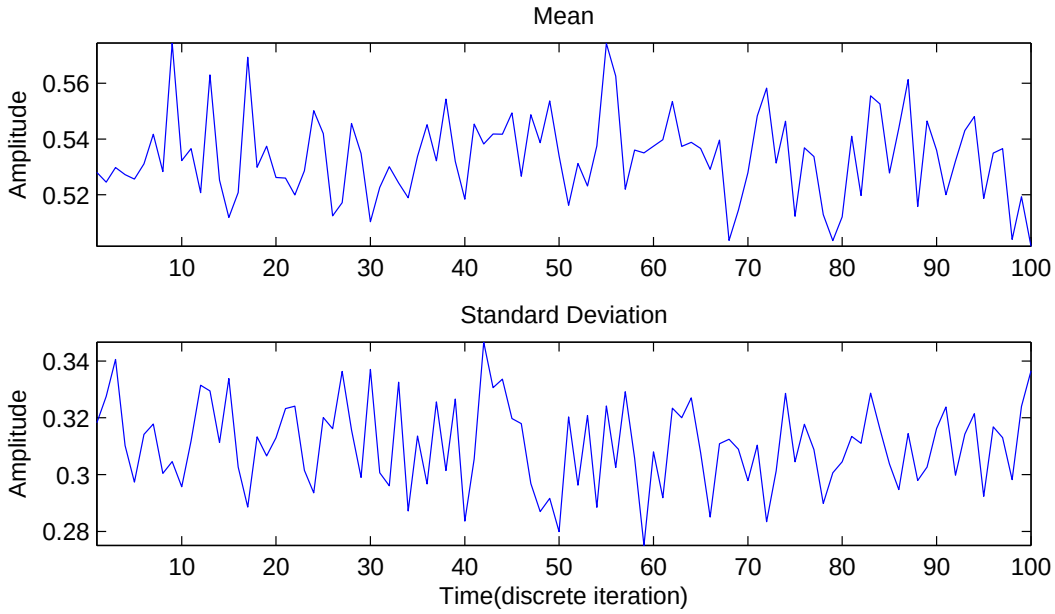


Figure 1.6: Mean and Standard Deviation over time for Process rp2

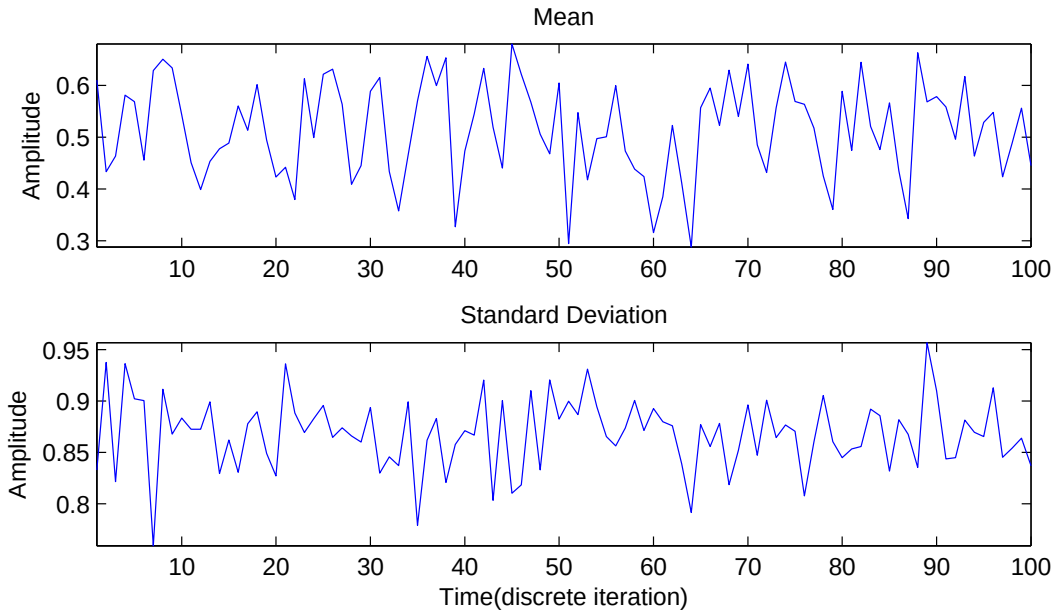


Figure 1.7: Mean and Standard Deviation over time for Process rp3

These graphs were made from the following code:

```
a = rp1(100,100);
figure(1)
subplot(2,1,1); plot(mean(a))
axis tight
title('Mean')
ylabel('Amplitude')
subplot(2,1,2); plot(std(a))
axis tight
title('Standard Deviation')
ylabel('Amplitude')
xlabel('Time(discrete iteration)')
a = rp2(100,100);
figure(2)
subplot(2,1,1); plot(mean(a))
axis tight
title('Mean')
ylabel('Amplitude')
subplot(2,1,2); plot(std(a))
axis tight
title('Standard Deviation')
ylabel('Amplitude')
xlabel('Time(discrete iteration)')
a = rp3(100,100);
figure(3)
subplot(2,1,1); plot(mean(a))
axis tight
title('Mean')
ylabel('Amplitude')
subplot(2,1,2); plot(std(a))
axis tight
title('Standard Deviation')
ylabel('Amplitude')
xlabel('Time(discrete iteration)')
```

1.2.2 Ergodicity of processes for M=4 and N=1000

In the following cases we are essentially taking 4 samples at 1000 different occurrences in time, i.e. we have 1000 different realisations of 4 samples.

For the rp1 process we can conclude that it is not ergodic due to the average of the ensembles not being the same as the time average, due to the nature of the trendline established earlier. Essentially a reading of many samples at one point in time will not help predict the full properties of this stochastic system.

For the rp2 process is not ergotic either. Indeed each set of occurrences and samples yield vastly different values. Thus observing one realisation of the system 1000 times will not provide us with all of the properties of the system, i.e. all the values that the system can take are not exhibited in one run and the starting value of the system determines the future instance of this system. Table 1.2 provides an example of this.

The rp3 process is ergotic as the systems expected mean and standard deviation is constant and does not change over different iterations. Thus in just one instance of many time samples the properties of the system can be determined.

Iteration	Expectation of Expectation of time	Expectation of Standard Deviation samples
1	0.1195	0.207
2	0.4367	0.2713
3	0.5635	0.3172
4	0.4527	0.2134
5	0.2346	0.3048
6	0.7103	0.2522

Table 1.2: Different iterations of rp2 to show how the expectation can change

1.2.3 Mathematical representation

1.2.3.1 Process rp1

Rp1 can be represented by the function:

$$v_t = cb \times \sin(\frac{t \times \pi}{T}) + at \tag{1.2}$$

Where:

a and b are constants and c is a uniformly distributed random variable between -0.5 and 0.5 with $\mathbb{E}(c) = 0$, $\sigma_c = \sqrt{Var(c)} = \frac{1}{2\sqrt{3}}$

t is also a constant representing the iteration number with T representing the total number of iterations.

From this we calculate the expected value:

$$\mathbb{E}(v_t) = \mathbb{E}(cb \times \sin(\frac{t \times \pi}{T}) + a \times t) = \mathbb{E}(c) \times \mathbb{E}(b \times \sin(\frac{t \times \pi}{T}) + \mathbb{E}(at) = 0 \times \mathbb{E}(b \times \sin(\frac{t \times \pi}{T})) + \mathbb{E}(a \times t) = at \tag{1.3}$$

And the standard deviation[1]:

$$\sigma_{v_t}^2 = Var(v_t) = \mathbb{E}(v_t^2) - \mathbb{E}(v_t)^2 = \mathbb{E}(c^2 b^2 \sin^2(\frac{t \times \pi}{T})) + 2 \times atcb \times \sin(\frac{t \times \pi}{T}) + a^2 t^2) - a^2 t^2, \tag{1.4a}$$

$$= \mathbb{E}\left(c^2 b^2 \sin^2\left(\frac{t \times \pi}{T}\right)\right) + \mathbb{E}\left(2 \times atcb \times \sin\left(\frac{t \times \pi}{T}\right)\right) + a^2 t^2 - a^2 t^2 \tag{1.4b}$$

$$= \mathbb{E}\left(c^2 b^2 \sin^2\left(\frac{t \times \pi}{T}\right)\right) + \mathbb{E}(c) \times \mathbb{E}\left(2 \times atb \times \sin\left(\frac{t \times \pi}{T}\right)\right) \tag{1.4c}$$

$$= \mathbb{E}(c^2) \times b^2 \sin^2\left(\frac{t \times \pi}{T}\right) \tag{1.4d}$$

$$= \frac{1}{12} \times b^2 \sin^2\left(\frac{t \times \pi}{T}\right) \tag{1.4e}$$

$$\therefore \sigma_{v_t} = \frac{1}{\sqrt{12}} b \times \sin\left(\frac{t \times \pi}{T}\right) \tag{1.5}$$

We know that the parameters where $a = \frac{1}{50}$ and $b = 5$, thus we can calculate the ideal values for expected value and standard deviation .

$\mathbb{E}(v_t) = \frac{t}{50}$, which fits in perfectly with the found values.

$\sigma_{v_t} = \frac{1}{\sqrt{12}} b \times \sin(\frac{t \times \pi}{T}) = \frac{5}{\sqrt{12}} \times \sin(\frac{t \times \pi}{T}) \approx 1.443 \times \sin(\frac{t \times \pi}{T})$, which also fits in perfectly with the observed standard deviation.

[1] $\mathbb{E}(c^2) = \frac{1}{12}$ due to it being taken from a uniform distribution.

1.2.3.2 Process rp2

Rp2 can be represented by equation 1.6.

$$v_t = X_n + Y_n \times Z_{t,n} \tag{1.6}$$

Where:

n and t represent what the random variables are random to, for example X_n and Y_n are both random with regards to the sample instance but not the number of realisations and $Z_{t,n}$ is random with respect to both sample instance and realisation.

X_n and Y_n are constants with respect to time uniformly distributed constants between 0 and 1 with $\mathbb{E}(X_n) = \mathbb{E}(Y_n) = 0.5$, $\sigma_{X_n} = \sigma_{Y_n} = \sqrt{Var(X_n)} = \sqrt{Var(Y_n)} = \frac{1}{2\sqrt{3}}$

$Z_{t,n}$ is a uniformly distributed random variable with $\mathbb{E}(z_n) = 0$, $\sigma_{z_n} = \sqrt{Var(Z_n)} = \frac{1}{2\sqrt{3}}$

Thus the expectation with respect to time.

$$\mathbb{E}(v_t) = \mathbb{E}(X_n) + \mathbb{E}(Y_n) \times \mathbb{E}(Z_{t,n}) = \mathbb{E}(X_n) + Y_n \times 0 = \mathbb{E}(X_n) = X_n \to 0.5 \tag{1.7}$$

Which backs up the findings that the rp2 process had a random expectation which in turn, over many realisations, has an average of 0.5. To illustrate this we have the graphs of the pdf of the mean of rp1 over many separate realisations. The code to make the following graphs is:

```
b=zeros(100,1);
for i = 1:100;
a = rp2(M,N);
b(i) = mean(mean(a));
end
figure(1)
gkdeb(b) %function written by Yi Cao (Cranfield University) to ...
    estimate pdf of input
%http://www.mathworks.co.uk/matlabcentral/fileexchange/
%19121-probability-density-function-pdf-estimator-v3-2/content/gkdeb.m
```

As we can see from Figure 1.8 that while the mean is always around 0.5 the distribution is very different according to the number of instances per time (M).

The standard deviation of rp2 is:

$$\begin{aligned} Var(v_t) = \mathbb{E}(v_t^2) - \mathbb{E}(v_t)^2 &= \mathbb{E}(X_n^2 + 2X_nY_n \times Z_{t,n} + Y_n^2 \times Z_{t,n}^2) - X_n^2 \\ &= \mathbb{E}(X_n^2) + \mathbb{E}(Y_n^2)\mathbb{E}(Z_{t,n}^2) - X_n^2 = \frac{1}{3} + \frac{1}{3} \cdot \frac{1}{12} - \frac{1}{4} = \frac{1}{9} \end{aligned} \tag{1.8}$$

Note: to find $\mathbb{E}(X_n^2)$ and $\mathbb{E}(Y_n^2)$ we can use $Var(X_n) = \mathbb{E}(X_n^2) - \mathbb{E}(X_n)^2 \to \frac{1}{12} + \frac{1}{4} = \frac{1}{3} = \mathbb{E}(X_n^2)$

$$\sigma_{v_t} = \sqrt{Var(v_t)} = \frac{1}{\sqrt{9}} = \frac{1}{3} \tag{1.9}$$

The result found in equation 1.9 is exactly what is observed.

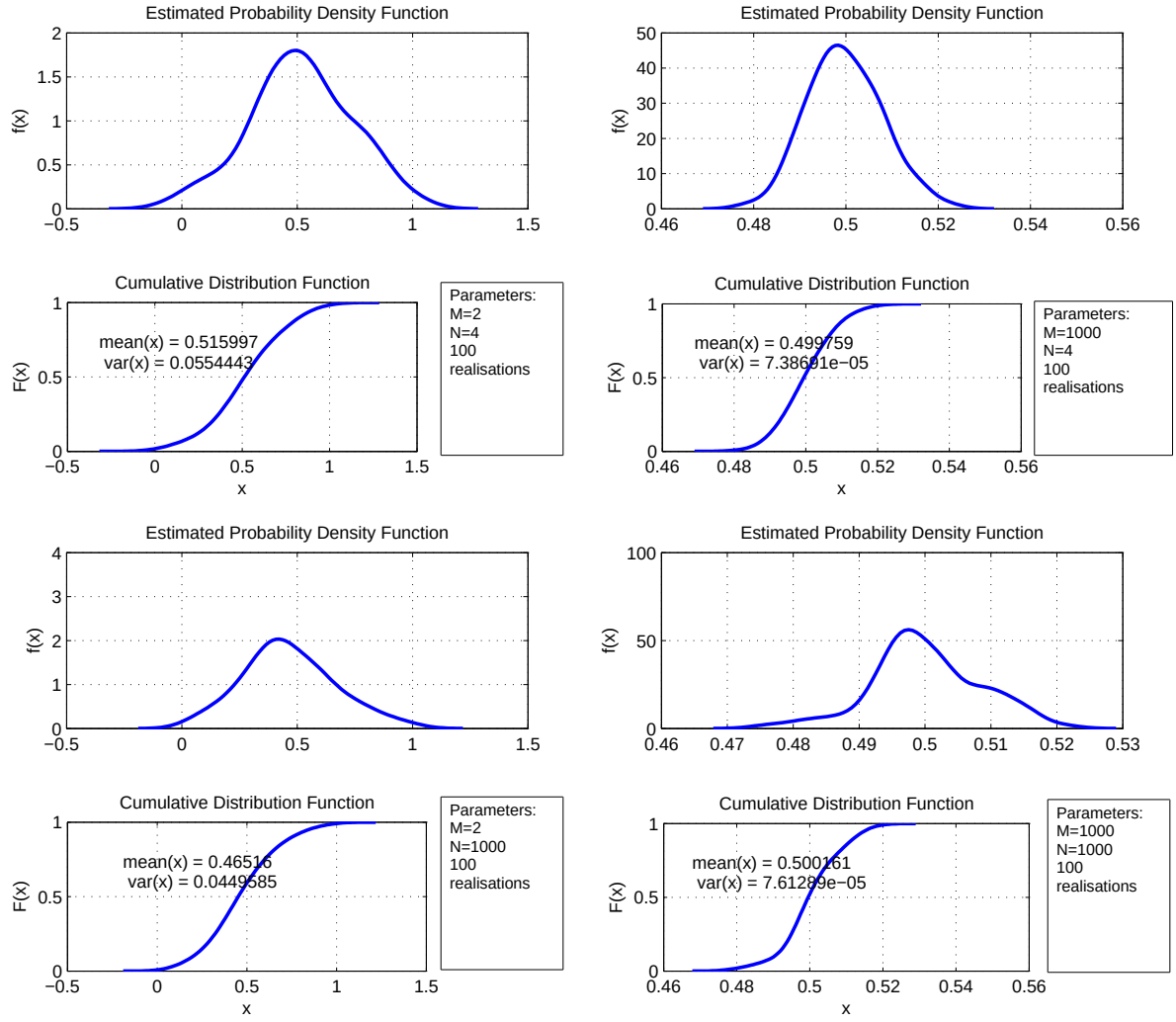


Figure 1.8: Estimated PDF of mean of Process rp2 over 100 iterations for different M and N values

1.2.3.3 Process rp3

We define the rp3 process as

$$v_t = m \cdot X + a, \text{where } X \sim \mathbb{U}(-\frac{1}{2}, \frac{1}{2}) \text{ and } m = 3 \text{ and } a = 0.5 \tag{1.10}$$

The expectation is

$$\mathbb{E}(v_t) = m \cdot \mathbb{E}(X) + a = a = \frac{1}{2} \tag{1.11}$$

And the standard deviation is:

$$\begin{aligned} Var(v_t) = \mathbb{E}(v_t^2) - \mathbb{E}(v_t)^2 = \mathbb{E}(m^2X^2 + a^2 + 2 \cdot amX) - \frac{1}{4} = \frac{9}{12} \\ \sigma_{v_t} = \sqrt{Var(v_t)} = \frac{3}{2\sqrt{3}} \approx 0.86 \end{aligned} \tag{1.12}$$

Which is what is observed.

1.3 PDF Estimation

1.3.1 Matlab PDF estimation

The following function was written to estimate the probability density function of input samples:

```
N=1000;
v=randn(1,N);
[a,b]=hist(v,50);
figure(1)
a=a/trapz(b,a);%more accurate than dividing by N
%as dividing by N is true only if the bin size is small
%relative to the variance of the data
bar(b,a);
xlabel('X')
ylabel('Probability')
title('Estimated PDF')
```

1.3.2 Comparison of theoretical to obtained PDFs

The only stationary and ergodic process is rp3 and as such will be the only investigated.

Figure 1.9 compares the theoretical PDF to the obtained, discrete, PDFs. As can be seen the higher the number of samples the closer the theoretical PDF is achieved.

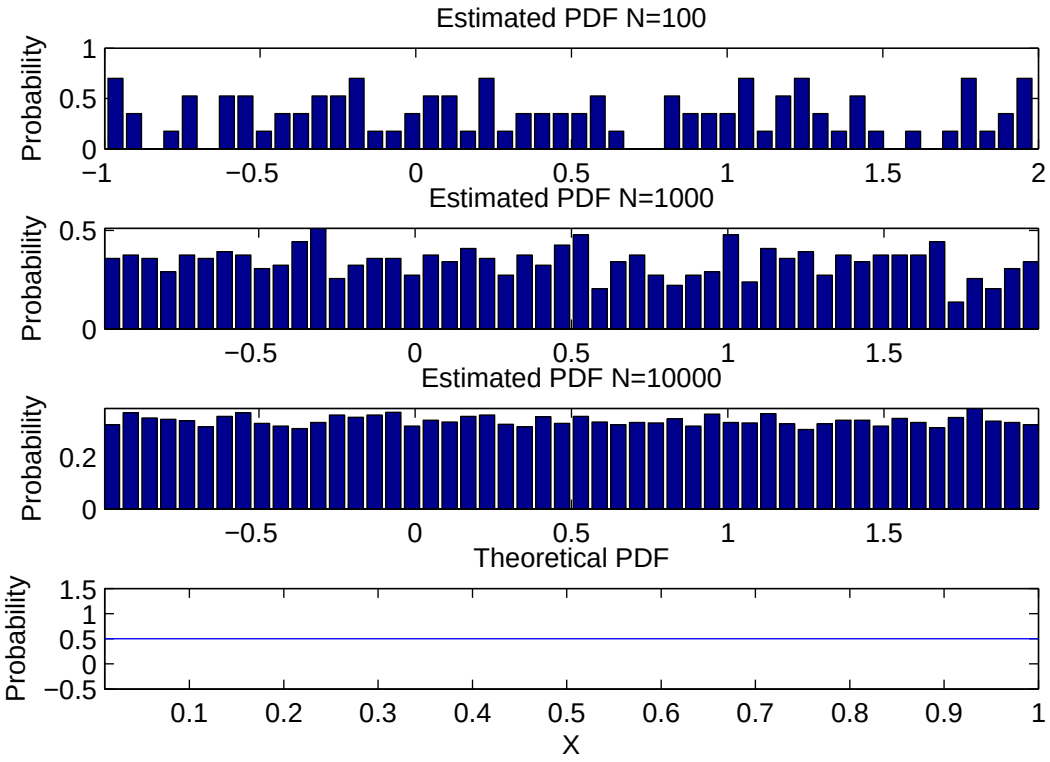


Figure 1.9: Estimated PDF of Process rp3 compared to its theoretical PDF

1.3.3 Estimation of non-stationary processes

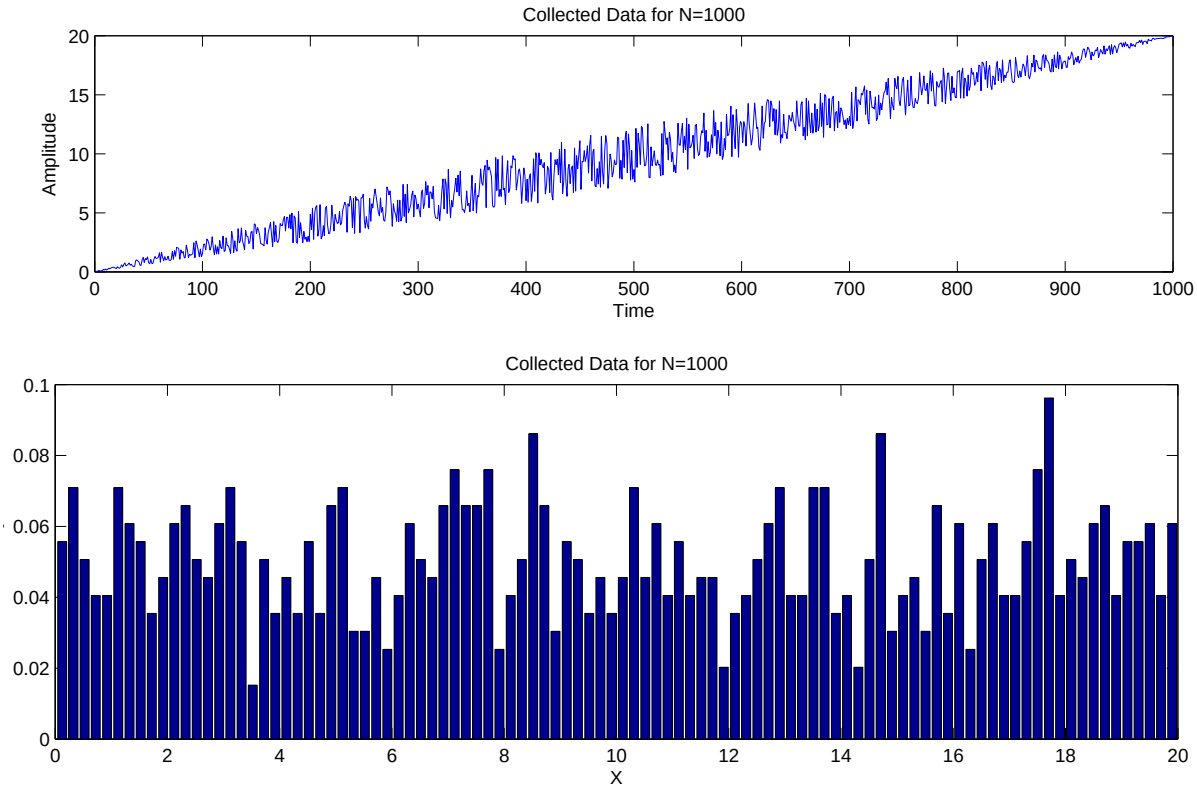


Figure 1.10: Estimated PDF of Process rp2 with its discrete values in time (above)

As we can see in Figure 1.10 the PDF implies that the function amplitude is randomly generated between 0 and 20. However from the graph above we know that this is not the case due to this not being a stationary process. As such using a hist function removes this critical information and causes a loss of information. (This is information which would not be contained in a stationary process explain why such processes are uniquely defined by a time-invariant pdf.).

1.4 Appendix

1.4.1 Matlab Code

1.4.1.1 Part 1

```
clc;
a=0;
x = randn(1000,1);
[a,b] = hist(x);
a = a./1000;
%b=-.9:0.2:.9;
sprintf('mean:%.6f',mean(x))
sprintf('std:%.6f',std(x))
bar(b,a)
```

1.4.1.2 Part 1.2

```
clc;
a = rp3(200,400);

subplot(2,1,1); plot(mean(a))
title('Mean')
subplot(2,1,2); plot(std(a))
title('Standard Deviation')
b=0;
c=0;
d=0;
for i = 1:1000;
a = rp3(200,400);
b(length(b)+1) = mean(mean(a));
c(length(c)+1) = mean(std(a));
end
figure(2)
gkdeb(c) %function written by Yi Cao (Cranfield University) to ...
    estimate pdf of input
%http://www.mathworks.co.uk/matlabcentral/fileexchange/
%19121-probability-density-function-pdf-estimator-v3-2/content/gkdeb.m
```

1.4.1.3 Part 1.3

```
clc;
N=100;
%v=randn(1,N);
v=rp3(1,N);
u=rp3(1,N*10);
w=rp3(1,N*100);
figure(1)
gkdeb(v);
[a,b]=hist(v,50);
figure(2)
a=a/trapz(b,a);%more accurate than dividing by N
```

```
%as dividing by N is true only if the bin size is small
%relative to the variance of the data
subplot(4,1,1)
axis tight
bar(b,a);
ylabel('Probability')
title('Estimated PDF N=100')
%%%%%%%%%%%
subplot(4,1,2)
[a,b]=hist(u,50);
a=a/trapz(b,a);
bar(b,a);
axis tight
ylabel('Probability')
title('Estimated PDF N=1000')
%%%%%%%%%%%
subplot(4,1,3)
[a,b]=hist(w,50);
a=a/trapz(b,a);
axis tight
bar(b,a);
axis tight
ylabel('Probability')
title('Estimated PDF N=10000')
%%%%%%%%%%%
subplot(4,1,4)
x=0.01:0.01:1;
y = ones(1,100).*1/2;
plot(x,y)
axis tight
xlabel('X')
ylabel('Probability')
title('Theoretical PDF')
clc;
%%%%%%%%%
v=rp1(1,1000);
plot(v)
[a,b] = hist(v,100);
a = a./trapz(b,a);
%bar(b,a);
xlabel('Time')
ylabel('Probability')
title('Collected Data for N=1000')
%%%%%%%%%
% subplot(2,1,2)
% v=rp1(1,1000);
% [a,b] = hist(v,100);
% a = a./trapz(b,a);
% bar(b,a);
% xlabel('X')
% ylabel('Probability')
% title('Collected Data for N=1000')
```

Part 2

Linear Stochastic Modeling

2.1 ACF of uncorrelated sequences

2.1.1 Matlab xcorr function

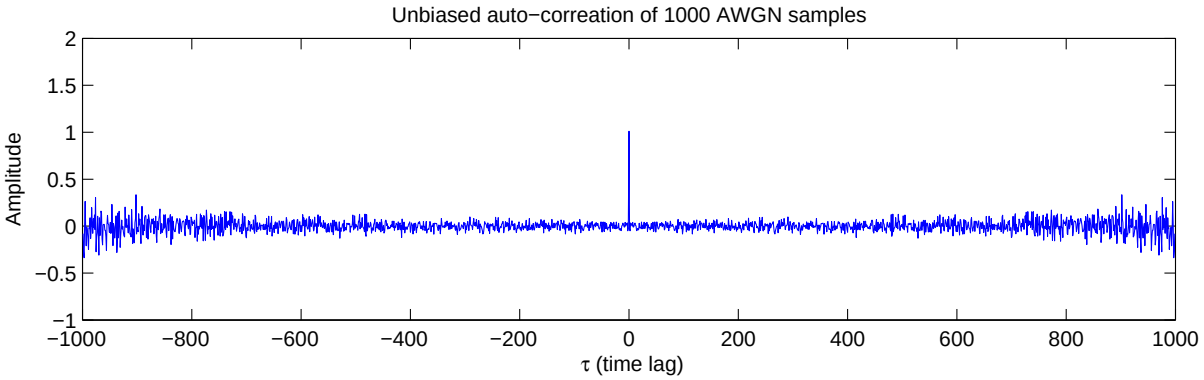


Figure 2.1: Unbiased autocorrelation of white Gaussian noise. The equation for unbiased autocorrelation is $\hat{R}_x[l] = \frac{1}{N-|l|} \sum_{n=1}^{N-|l|} x[n]x[n+l]$

Figure 2.1 shows the autocorrelation function of 1000 AWGN samples. We notice the discrete delta at $\tau = 0$ (the peak of 1) but more importantly the cross-correlation estimate increases as $\tau = \pm 999$ is reached starting from around $\tau = \pm 500$. This is because past this limit less than half the values of x (the vector storing the instance of the 1000 AWGN samples) overlap due to being of length 1000. Thus once ± 999 is reached it is understandable that an amplitude increase may increase due to only one sample being available for comparison. We also notice that the function is symmetric around $\tau = 0$. This is due to the autocorrelation being the correlation of a signal with itself thus at equal distances from $\tau = 0$ the same values of x will be compared.

2.1.2 Zoomed in instance

In Figure 2.2 it can be noticed that the amplitude is reasonably constant for values other than $\tau = 0$. This is, as explained above, due to the samples always overlapping and canceling each other out giving out a more reasonable estimate than for larger τ's.

The more discrete appearance of this graph is simply due to less samples being present.

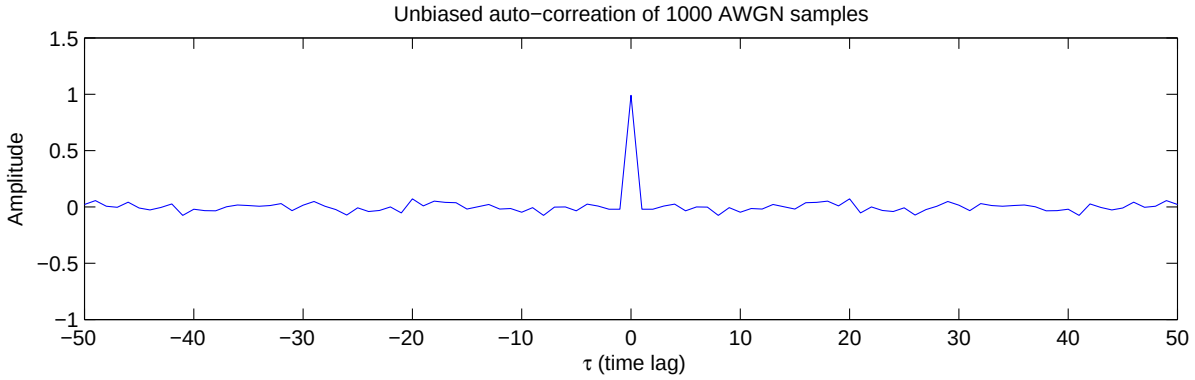


Figure 2.2: Unbiased autocorrelation of white Gaussian noise from $\tau = -50$ to $\tau = 50$

2.1.3 Empirical bound on τ

We notice that for $\hat{R}_x[\tau] = \frac{1}{N-|\tau|}\sum_{n=1}^{N-|\tau|} x[n]x[n+\tau]$ values past a certain limit start to diverge and suggest that for large lags the signal is in fact similar to itself samples before. As the samples taken are AWGN this is not expected (while it is possible the probability is extremely low).

In the previous case for 1000 AWGN samples a value of $\tau = \pm 500$ was a good estimate i.e. $\pm\frac{N_{samples}}{2}$. This is because past this value more than half of the samples will be available for comparison and the probability of the series resembling itself increases. Looking at $\hat{R}_X$ we notice that the factor $\frac{1}{N-|\tau|}$ also suggests that as τ gets larger the accuracy decreases due to less samples being available, confirming that estimates do get worse as the lag increases.

A crude explanation is that this due to the probability of values having the same sign (i.e. adding up) for a randomly generated process increasing when the number of samples decrease. For example if only 1 sample pair is used there is a 50% chance of both of those have the same sign, for 2 sample pairs there is a 25% chance etc. It is obvious why the maximum amplitude is reached for $\tau = 0$ and decreases as τ gets larger but then starts increasing again when $\tau \to \pm$limit due to the sample size decreasing.

2.2 ACF of filtered sequences

2.2.1 Moving Average Filter

Figure 2.3 and 2.4 show use the effects of passing AWGN samples through an nth order MA filter. In this case orders of 4 and 9 were chosen. As we can see the ideal ACF function for AWGN passed through a MA filter of order N is $ACF_{ideal}(\tau) = N \cdot \bigwedge(\frac{\tau}{N})$, where $\bigwedge(t)$ is the triangle function. This is due to the way the moving average filter works which is by taking the coefficients it has and recursively adding up past elements (the equation for FIR filter of order N is $y[n] = \sum_{i=0}^{N} b_i x[n-i]$). Thus by taking the average of elements at relative index 1 to N will be correlated to each other, with a higher correlation for being closer to the original element, explaining the triangle shape of the ACF.

The effect of the signal on its output is thus to smooth it out, we can see this from the fact that the 4^{th} order MA signal looks more jagged than the 9^{th} order one.

If we have an MA of order equal to the number of samples (N) then the value of average could be calculated by $y[N] = \sum_{i=0}^{N-1} b_i x[N-i]$ if $b_i = \frac{1}{N}$ as this would essentially become the equation of average $y[N] = mean(y) = \frac{1}{N}\sum_{i=0}^{N-1} x[N-i]$.

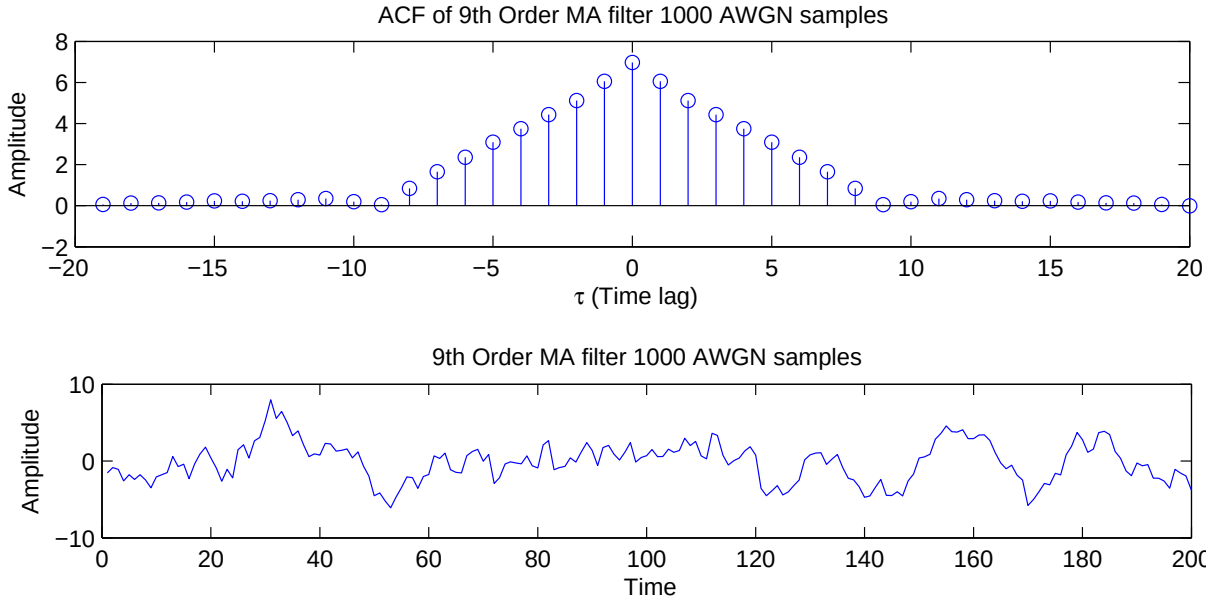


Figure 2.3: Autocorrelation of a 9^{th} order MA filter on 1000 samples with its time domain plot.

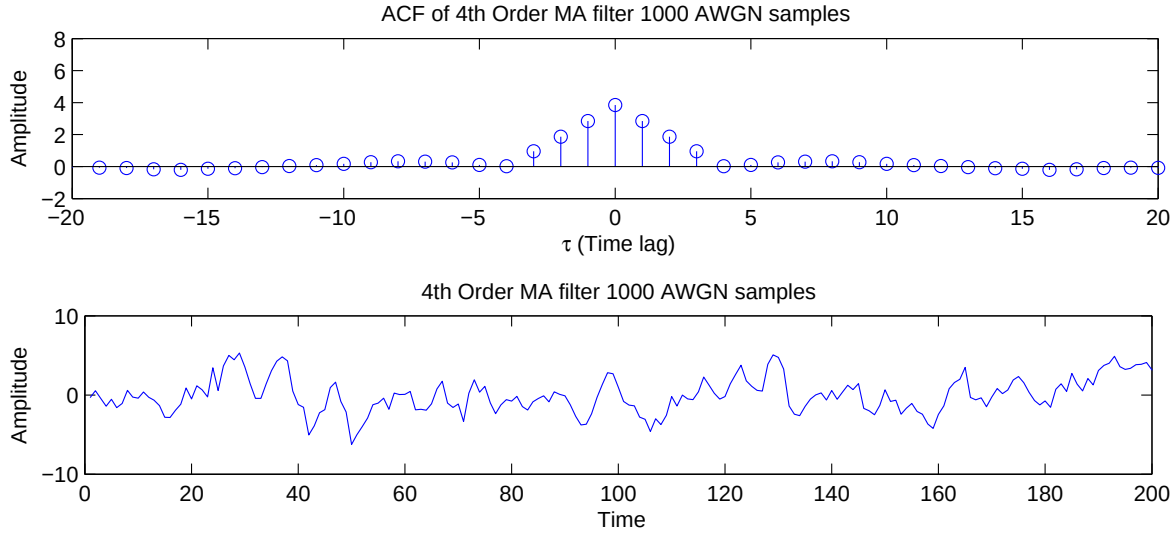


Figure 2.4: Autocorrelation of a 4^{th} order MA filter on 1000 samples with its time domain plot.

2.2.2 Stochastic process correlation

If Y_n is the result of a stochastic process, X_n is obtained from an uncorrelated process and we have

$$R_Y(\tau) = R_X(\tau) * R_h(\tau) \tag{2.1}$$

Then we have that $R_Y(\tau) = R_h(\tau)$ due to $R_X(\tau) = \delta(\tau)$, which is the result of the autocorrelation of an uncorrelated process (and any function convoluted by $\delta(\tau)$ is equal to itself). Thus the autocorrelation of Y represents the autocorrelation of the impulse response. In the case of processes taken from 2.2.1 the result is, as expected, the function $N \cdot \bigwedge(\frac{\tau}{N})$.

2.3 Cross-correlation function

2.3.1 Cross correlation of X and Y from 2.2

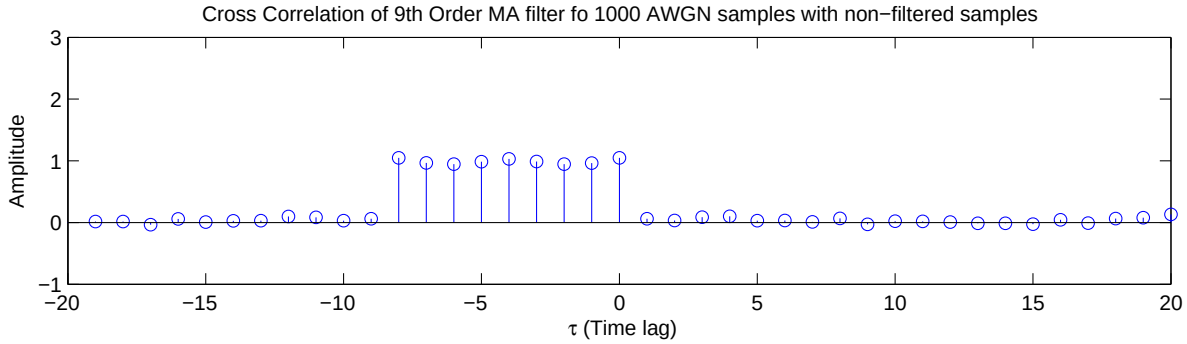


Figure 2.5: Cross correlation of X and Y process from 2.2

Figure 2.5 shows the plot of the cross correlation of X and Y. We notice that the result is the impulse response (here it is reversed in time but this depends on if `xcorr(y,x, 'unbiased')` or `xcorr(x,y, 'unbiased')` is used).

This is the expected response as the cross correlation X and Y is equal to $R_{XY}(\tau) = h(\tau) * R_X(\tau)$ which, by using the fact that X is AWGN which is uncorrelated, is equal to $R_{XY}(\tau) = h(\tau)$.

In this case $h(\tau) = \sum_{i=0}^{N} \delta(-i)$, which is what we have for the observed instance.

2.3.2 System estimation from cross correlation function

By applying the CCF on two functions, one which we know is the result of a FIR (IIR or any other LTI system would also work) filter and the other function being the original function, the impulse response of the system can be obtained though this is only possible if the original process is uncorrelated (i.e. we must have that $R_X(\tau) = \delta(\tau)$).

From the impulse response we can deduce the order by the number of delta function and if the system is an FIR the coefficients can be estimated.

2.4 Autoregressive modelling

2.4.1 Sunspot Data

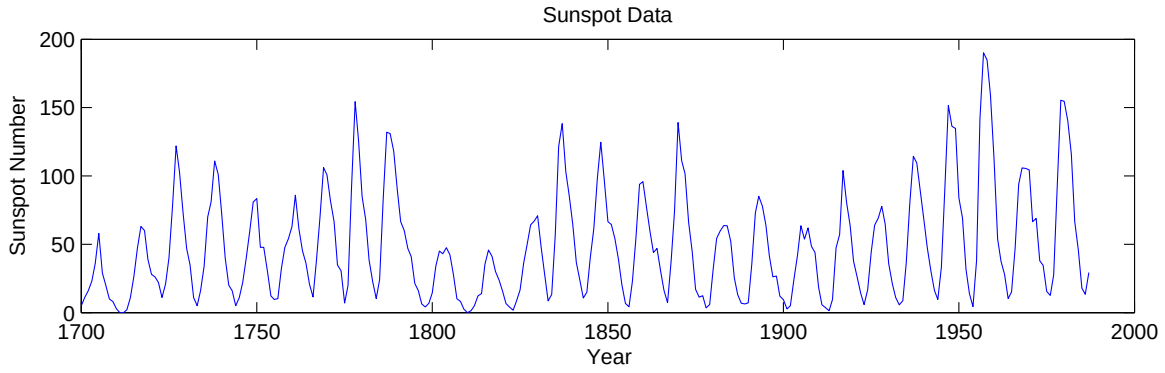


Figure 2.6: Plot over time of the Zürich sunspot relative number data.

Figure 2.7 shows the sunspot data plotted in time as well as the ACF of the data for various N. We can notice that for very small N (N=5) it is difficult to see that the data is recurring every few years, whereas the shape for N=20 shows how for every 13 years or so sunspot data repeats. When contrasting the ACF for N=20 to N=250 it can be noticed that the peaks of the data are higher originally but once a longer N is applied these apparent peaks do not exist. This is due to the issues with observing/interpreting data past $\pm\frac{N}{2}$ as discussed in section 2.1.

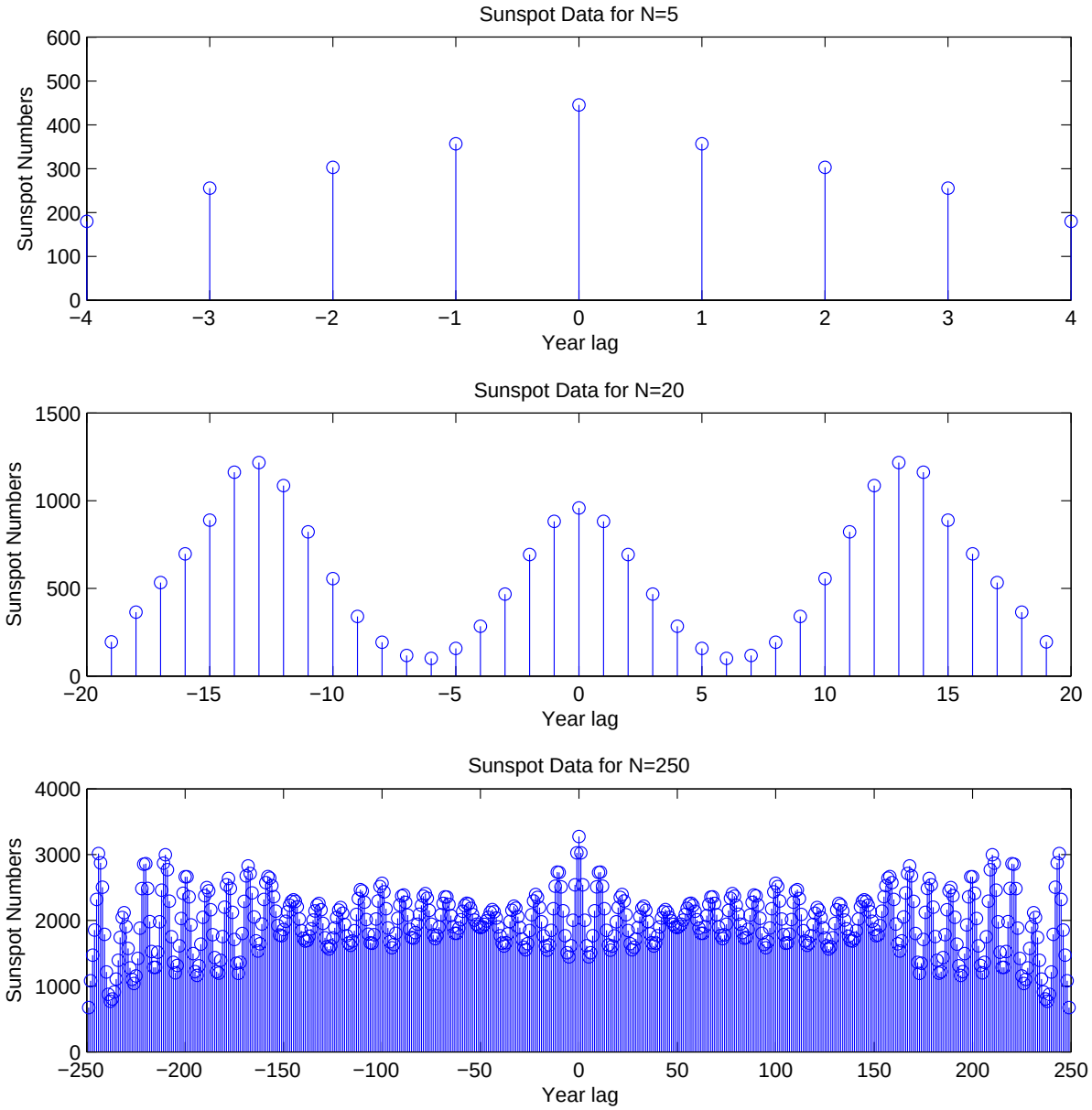


Figure 2.7: Autocorrelation of Zürich sunspot relative number data for various N values.

2.4.2 Zero mean data ACF

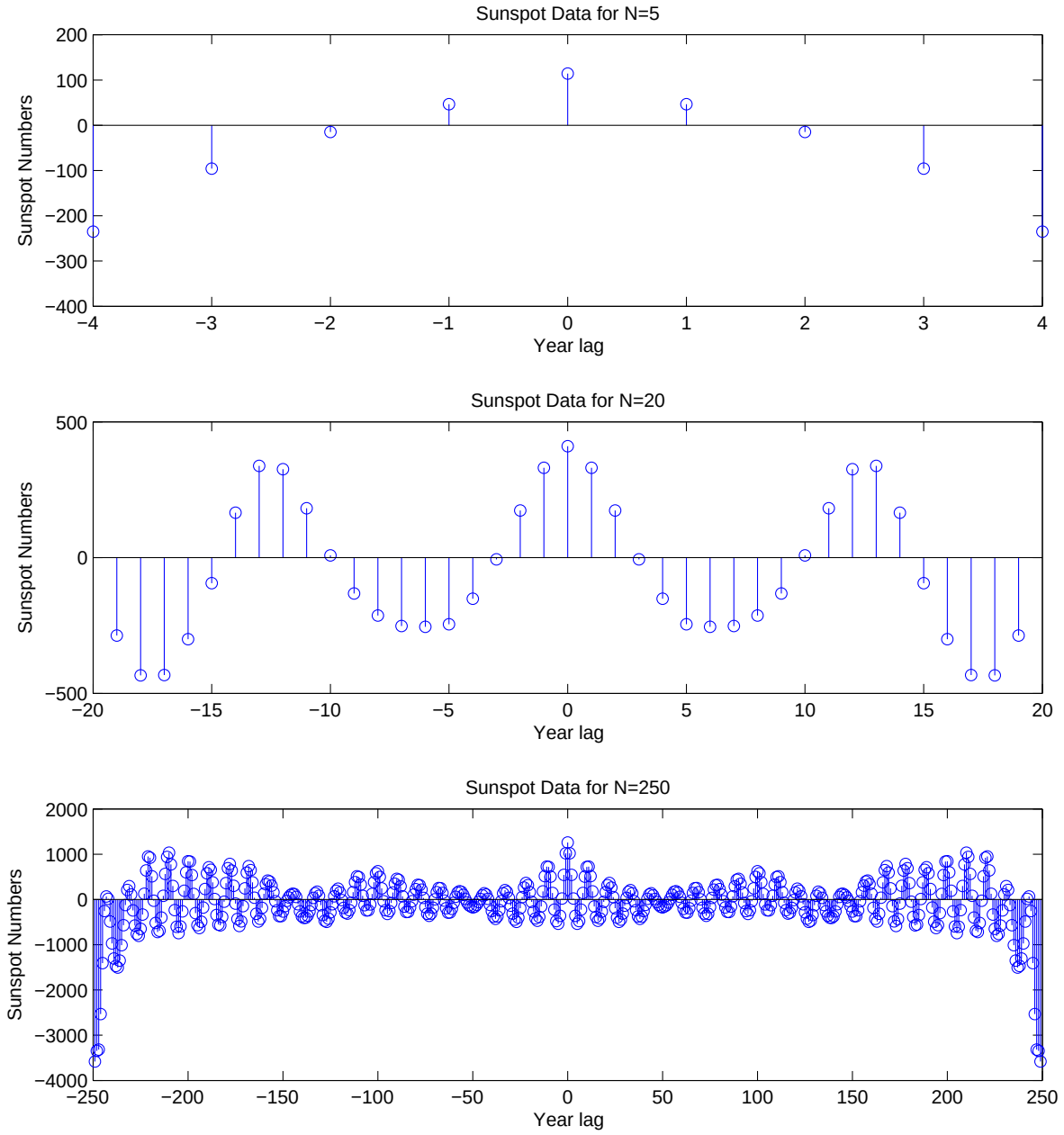


Figure 2.8: Autocorrelation of zero-meaned Zürich sunspot relative number data for various N values.

This time zero meaning the data has fixed a few problems observed earlier. For example the problem with the peaks being higher for N=20 (which implied the signal was more similar to itself when shifted than when it wasnt) is no longer the case. Finally for N=250 the data is somewhat more clear to observe. This is due to the way the ACF works, multiplying each shifted element by another, thus by placing a zero mean we ensure than there is an equal representation in positive as well as negative magnitudes meaning data does not simply accumulate throughout the ACF. This allows the periodicity of the data to be better seen and understood, with the peaks representing periods of repetition.

In Figure 2.8 we are operating on data which sample set has been zero-meaned in the sense that the mean has been removed after collecting the data sets. It is also possible to remove the

mean of the sunspot data before taking the samples. In this this case the graphs in Figure 2.9 would be obtained. It can be noticed that the start amplitude is always the same. However

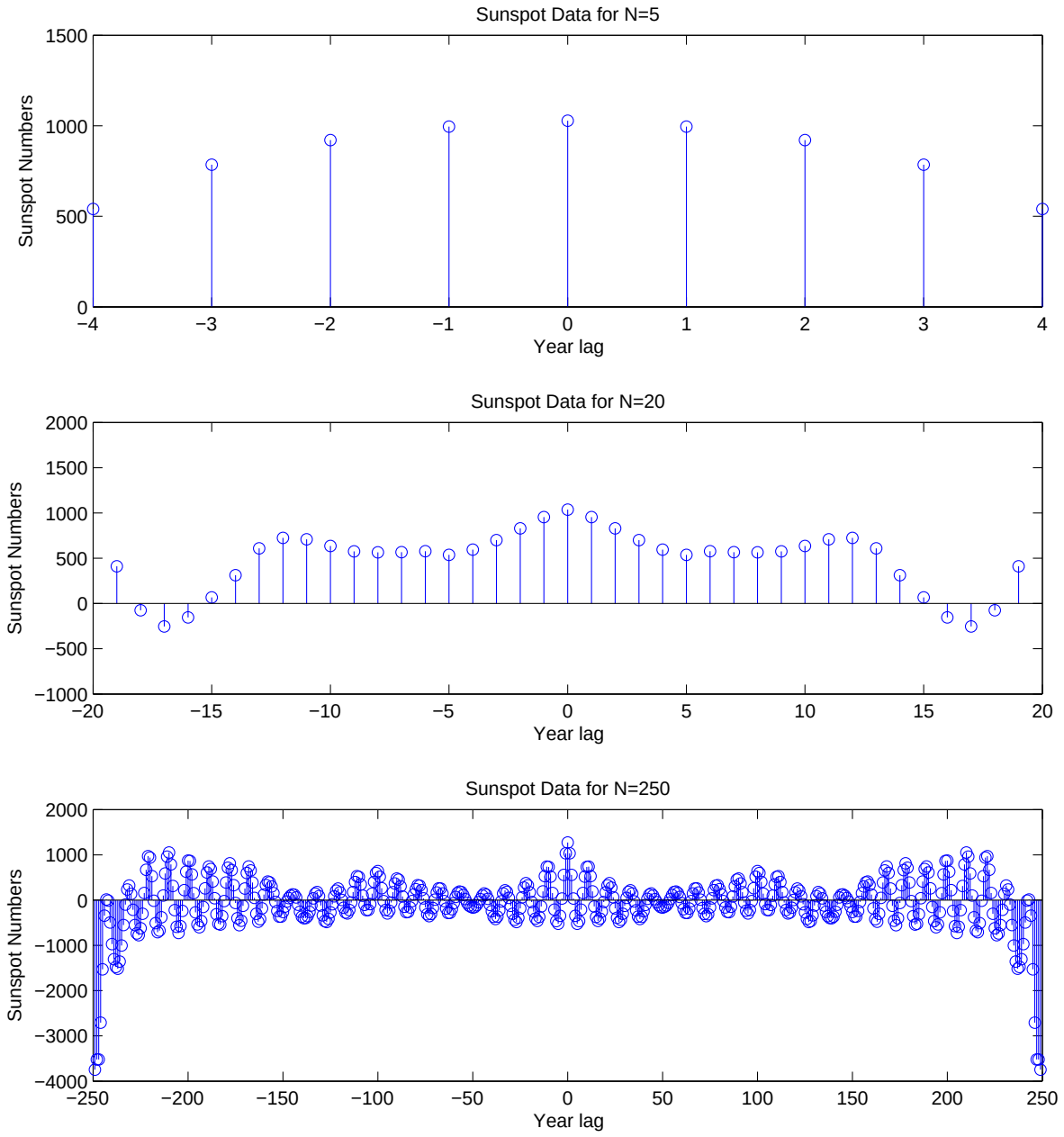


Figure 2.9: Autocorrelation of pre-zero-meaned Zürich sunspot relative number data for various N values.

it is still difficult to draw conclusions about the datas periodicity thus the previous method is preferred.

2.4.3 AR2 Stability

Figure 2.10 has blue stars and red dots to represent stable and unstable processes. In red are represented all the coefficients for stable process and in blue we can see the coefficient pairs for unstable processes. The stability of a process was detected by checking whether or not the final value of x (x[1000]) overflowed or not in matlab.

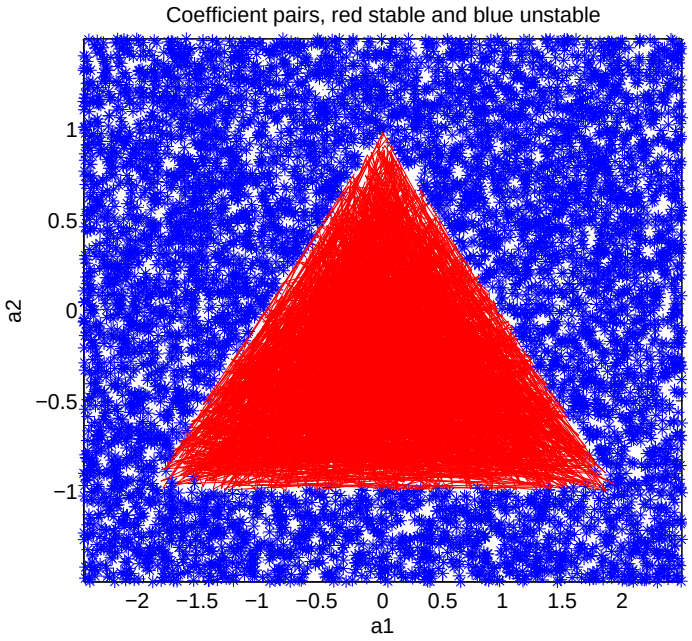


Figure 2.10: Stability triangle for an AR2 process.

This graph basically replicated the stability triangle as mentioned in the notes and to be stable a process must satisfy, all at the same time:

$$\begin{aligned} a1 + a2 &< 1 \\ a2 - a1 &< 1 \\ -1 &< a2 < 1 \end{aligned} \tag{2.2}$$

These equations arise from having roots inside the unit circle which implies a stable process.

2.4.4 Yule-Walker equations

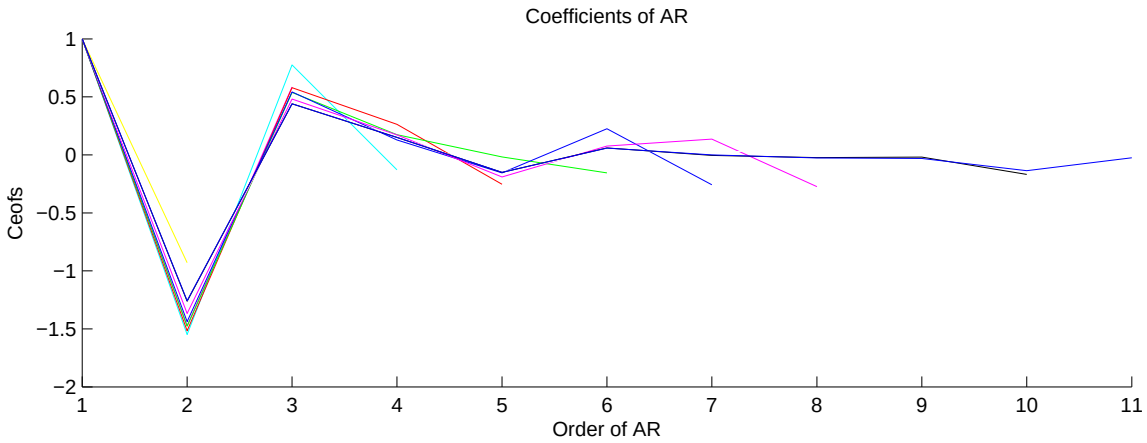


Figure 2.11: Plot of coefficients across various AR orders. AS can be seen past 2 most coefficients are around 0 and thus from this graph the optimal order would be 2.

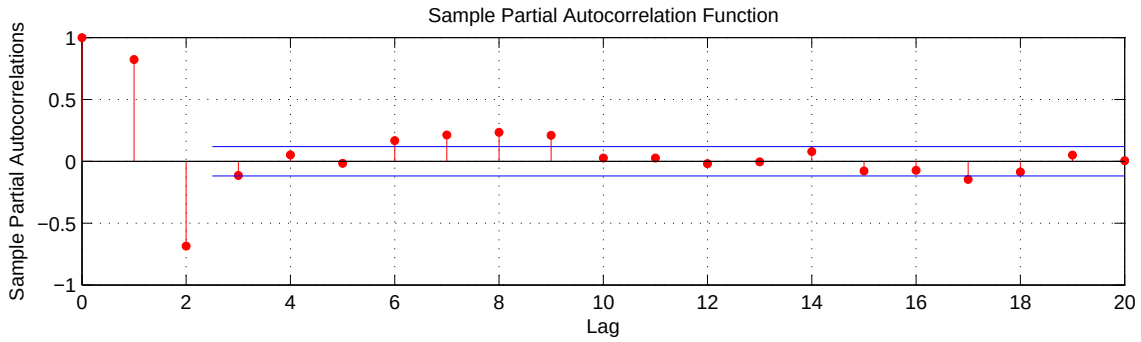


Figure 2.12: These lines were obtained from the partial autocorrelation function which calculates the autocorrelation of a series up to a certain time difference ignoring the other lags (thus its name partial autocorrelation) - meaning linear dependence of samples on each other is removed. This allows the determination of whether or not a sample is worth relative to another one and can help in the identification of the order of an AR process.

From the sample partial autocorrelation function in Figure 2.12 we can estimate the most likely order of the series which seems to be 2 as all subsequent coefficients are much smaller than the previous ones. Additionally the guidance lines the partial autocorrelation provides also point towards the sunspot process being best approximated by an AR(2) process.

2.4.5 Determining the correct model order

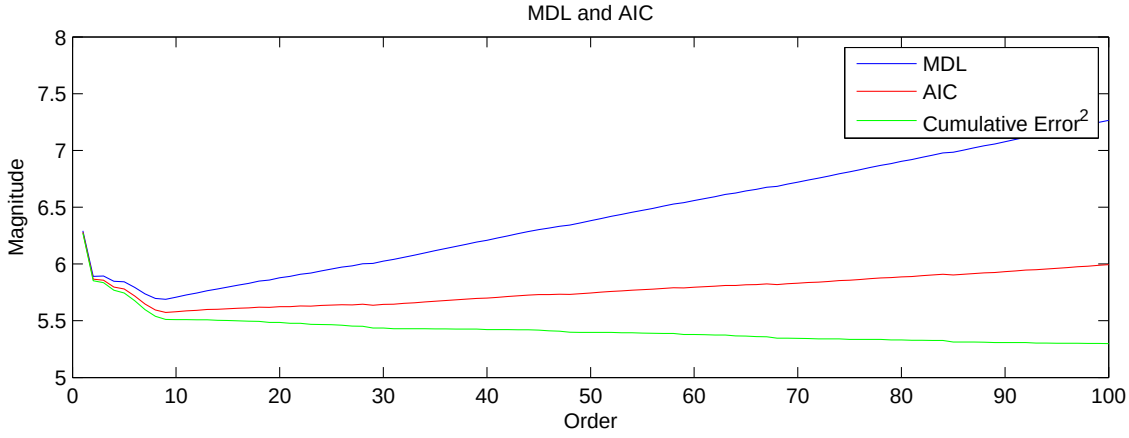


Figure 2.13: MDL, AIC and cumulative error squared between the orders of 0 to 100. The first minimum is found to be at 2 but both global minima occur at 9 for both AIC and MDL.

Here we use the Minimum Description Length (MDL) and Akaike Information Criteria (AIC) to help us determine the most optimal order. They work by calculating the expected error which leaves us to choose the minimum value which will guide us to choosing the order. The equations are:

$$\begin{aligned} \mathrm{MDL}(p) &= \log\left(E_p\right) + \frac{p\log(N)}{N} \\ \mathrm{AIC}(p) &= \log\left(E_p\right) + \frac{2p}{N} \end{aligned} \tag{2.3}$$

P is the order of the model , N being the number of samples and E is the cumulative squared error.

Figure 2.13 shows how order 9 is the most optimal model for the sunspot data. This means that for an order of 9 the amount of information lost is minimized however this does not predict whether or not the data will be predicted correctly. This is due to MDL and AIC only being relative comparators the best estimated order could still return a faulty model.

2.4.6 AR Modeling

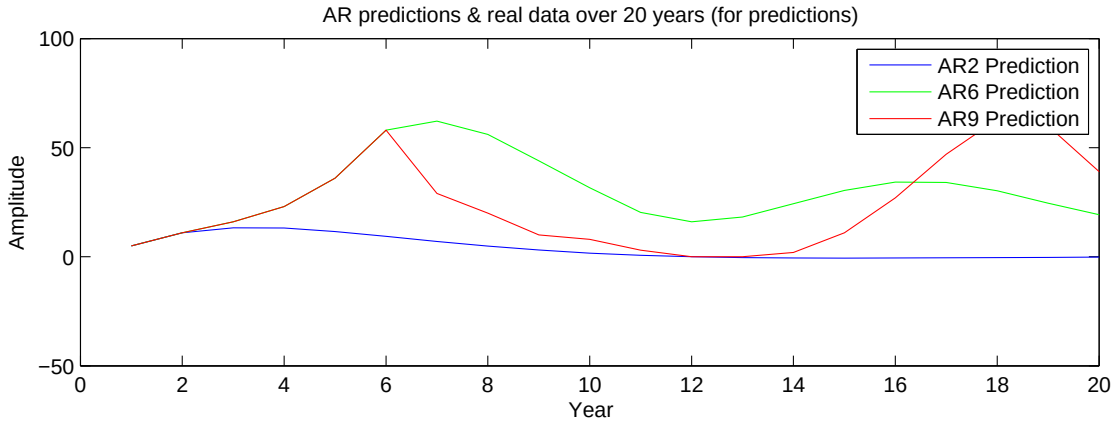


Figure 2.14: Prediction for an AR2 and AR6 process of future sunspot data over a period of 20 years. This means that 18 years are predicted by the AR2 process and 14 years predicted by the AR6 process.

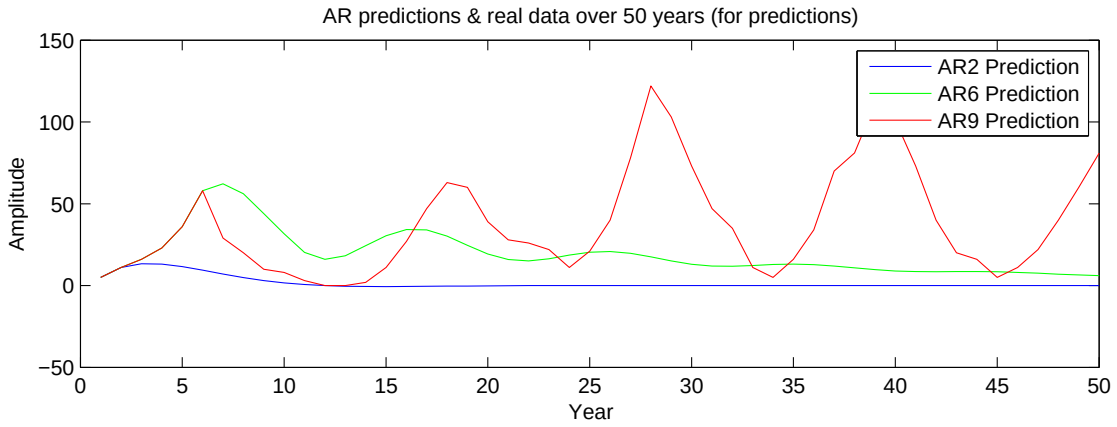


Figure 2.15: Prediction for an AR2 and AR6 process of future sunspot data over a period of 50 years. As can be seen the AR predicted data decays reasonably fast and would need more data input points to continue predicting.

Using the coefficients found from the Yule-Walker equations we can construct models to predict future sunspot data. From the Y-W coefficients we can use the following equation $x[n] = \sum_{k=1}^{p} a[k]x[n-k]$ to predict future values by first filtering the p previous values (p is the order of the AR process).

From Figure 2.14 and Figure 2.15 we can see that the AR2 and AR6 process can predict data to a certain extent with the AR6 having a better success due to being of higher order. AR2 badly performing is not too surprising due to the data having a repeating pattern in the order of 10 years.

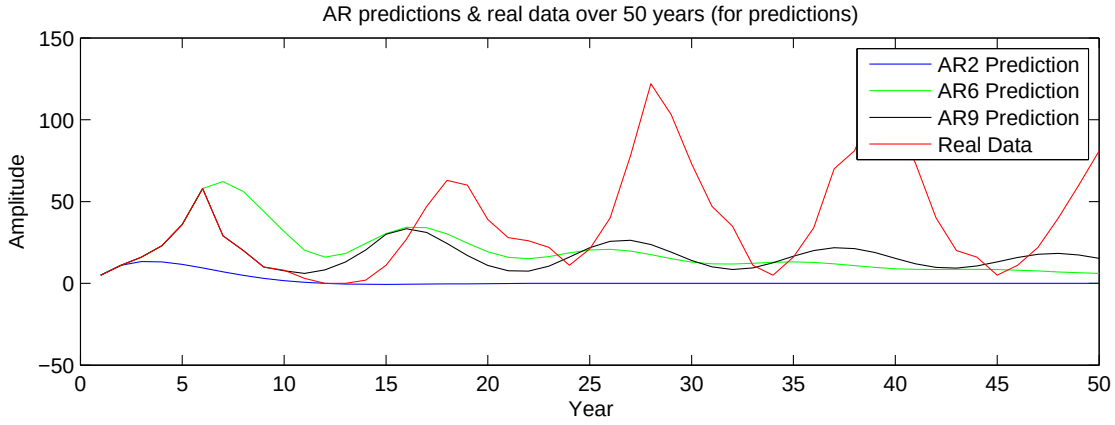


Figure 2.16: Prediction for an AR2, AR6 and AR9 process of future sunspot data over a period of 50 years. The AR9 process can be seen to oscillate much longer without need of further input however the values start to get out of phase and have other issues such as having a very low magnitude.

Figure 2.16 looks at how these predictions do over a very long period of time. The result is not surprising all the AR models tend to 0 meaning that more input values are needed to re-energize them.

2.5 Appendix

2.5.1 Matlab Code

2.5.1.1 Part 2.1

```
clc;
x = randn(1,1000);
[a,b] = xcorr(x, 'unbiased');

plot([-999:1:999],a)
xlabel(' \tau (time lag)')
ylabel('Amplitude')
title('Unbiased auto-correation of 1000 AWGN samples')
axis([-50 50 -1 1.5])
```

2.5.1.2 Part 2.2

```
clc;
Or = 4;
x = randn(1,1000);
y=filter(ones(Or,1),[1],x);
[a,b] = xcorr(y, 'unbiased');
subplot(2,1,1)
stem(b,a)
xlabel(' \tau (Time lag)')
ylabel('Amplitude')
str = sprintf('ACF of %dth Order MA filter 1000 AWGN samples',Or);
title(str)
axis([-20 20 -2 8])
subplot(2,1,2)
plot(y)
xlabel(' Time')
ylabel('Amplitude')
str = sprintf('%dth Order MA filter 1000 AWGN samples',Or);
title(str)
y(1000)
mean(y)
axis([0 200 -10 10])
```

2.5.1.3 Part 2.3

```
clc;
Or = 9;
x = randn(1,1000);
y=filter(ones(Or,1),[1],x);
[a,b] = xcorr(x,y, 'unbiased');
stem(b,a)
xlabel(' \tau (Time lag)')
ylabel('Amplitude')
str = sprintf('Cross Correlation of %dth Order MA filter fo 1000 ...
    AWGN samples with non-filtered samples',Or)
```

```
10 title(str)
11 axis ([-20 20 -1 3])
```

2.5.1.4 Part 2.4

```
1  clc;
2  % close all;
3  clear;
4  N = 250;
5  load sunspot.dat
6  year=sunspot(1:N,1);
7  x =sunspot(:,2);
8  %% plot sunspot data
9  figure(1)
10 plot(year(1:N),x(1:N));
11 title('Sunspot Data')
12 xlabel('Year')
13 ylabel('Sunspot Number')
14 %% Plot autocorrelation
15 x=x-mean(x); %normalize data
16 x = x(1:N);
17 [a,b] = xcorr(x, 'unbiased');
18 figure(2)
19 stem(b,a)
20 xlabel('Year lag')
21 ylabel('Sunspot Numbers')
22 str = sprintf('Sunspot Data for N=%d',N);
23 title(str)
```

2.5.1.5 Part 2.4.3

```
1  clc;
2  clear;
3  %plots the stability triangle if called mutliple times
4  N = 1000;
5  a1=5.*rand(100,1)-2.5;
6  a2=3.*rand(100,1)-1.5;
7  w=randn(N,1);
8  % a1_2 = zeros(100,1)
9  % for j=1:100
10 %     a1_2(i) = a1(i)^i;
11 %     a2_2(i) = a2(i)^i;
12 % end
13
14 x = zeros(N,N);
15 x(:,1) = w(1);
16 x(:,2) = w(2)+a1(1)*x(:,1);
17 for j=1:100
18     for i=3:N
19         x(j,i) = w(i) + a1(j)*x(j,i-1) + a2(j)*x(j,i-2);
20     end
21 end
22 a = zeros(100,1);
```

```
b = zeros(100,1);
for j=1:100
    %cehck for overshoot
    if (abs(x(j,N))>10)||(isnan(x(j,N)))
        disp(j)
        a(j) = a1(j);
        b(j) = a2(j);
        a1(j)=0;
        a2(j)=0;
    else
        str = sprintf('--------------\nfor j=%d\na1:%f\na2:%f ...
           \nx_end:%f\na1+a2:%f\n',j,a1(j),a2(j),x(j,N),a1(j)+a2(j));
        disp(str);
    end
end
%clear coef list
a1(a1==0)=[];
a2(a2==0)=[];
a(a==0)=[];
b(b==0)=[];
plot(a1,a2,'r-');hold on
grid on
plot(a,b,'*');
xlabel('a1')
ylabel('a2')
str = sprintf('Coefficient pairs, red stable and blue unstable');
title(str)
```

2.5.1.6 Part 2.4.4

```
clc;
clear;
N = 288;
load sunspot.dat
year=sunspot(1:N,1);
x =sunspot(1:N,2);
% for i=1:10
% str = sprintf('ar_%d = aryule(x,%d);',i,i);
% disp(str)
% end
%get coefs
ar_1 = aryule(x,1);
ar_2 = aryule(x,2);
ar_3 = aryule(x,3);
ar_4 = aryule(x,4);
ar_5 = aryule(x,5);
ar_6 = aryule(x,6);
ar_7 = aryule(x,7);
ar_8 = aryule(x,8);
ar_9 = aryule(x,9);
ar_10 = aryule(x,10);
% for i=1:10
% str = sprintf('plot(ar_%d);',i);
% disp(str)
% end
%plot them
```

```
figure(1)
hold on
plot(ar_1,'y');
plot(ar_2,'m');
plot(ar_3,'c');
plot(ar_4,'r');
plot(ar_5,'g');
plot(ar_6,'b');
plot(ar_7,'m');
plot(ar_8,'w');
plot(ar_9,'k');
plot(ar_10);
hold off
xlabel('Order of AR')
ylabel('Ceofs')
str = sprintf('Coefficients of AR');
title(str)
figure(2)
% parcorr(x,[],2)
pyulear(x,2)
```

2.5.1.7 Part 2.4.5 & 2.4.6

```
clc;
close all;
clear all;
% PART 2.4.5 & 2.5.6
load sunspot.dat;
year = sunspot(:,1);
data = sunspot(:,2);
N = length(data);
years = 50;
%Process the MDL and AIC for 100 orders
for i = 1:100
    [coefficients,E] = aryule(data, i);
    ord = length(coefficients);
    y(:,i) = filter(-1*coefficients(2:end),1,data(:));%cut off first ...
        value and negate the rest due to the way the equation works
    MDL(i,1) = log(sum(E)) + (i*log(N))/N;%Calculate MDL
    AIC(i,1) = log(sum(E)) +(2*i)/N;%Calculate AIC
    Er(i) = log(sum(E));
end
 %Generate coefficients for AR processes
coeffs_order2 = -1*aryule(data, 2);
coeffs_order6 = -1*aryule(data, 6);
coeffs_order9 = -1*aryule(data, 9);
ar2(1) = data(1);
ar2(2) = data(2);

%Initialise all
for i = 1:20
 ar2(i) = data(i);
 ar6(i) = data(i);
 ar9(i) = data(i);
end

```

```
%AR2 prediction
for i = 1:years-2,
    ar2(2+i) = coeffs_order2(2)*ar2(1+i) + coeffs_order2(3)*ar2(i);
end
%AR6 prediction
for i = 1:years-6,
   ar6(6+i) = coeffs_order6(2)*ar6(5+i) + coeffs_order6(3)*ar6(4+i) ...
       + coeffs_order6(4)*ar6(3+i) + coeffs_order6(5)*ar6(2+i) + ...
       coeffs_order6(6)*ar6(1+i) + coeffs_order6(7)*ar6(i);
end

%AR9 future
for i = 1:years-9,
    ar9(9+i) = 0;
    for j = 1:9
        ar9(9+i) = ar9(9+i) + coeffs_order9(1+j)*ar9(9+i-j);
    end
end

%Plot all
figure(1)
plot(MDL), title('MDL and AIC'), xlabel('Order'), ylabel('Magnitude');
hold on;
plot(AIC,'r');
plot(Er,'g');
legend('MDL','AIC','Cumulative Error^2');

figure(2)
str = sprintf('AR predictions & real data over %d years (for ...
   predictions)', years);
plot(ar2), title(str), xlabel('Year'), ylabel('Amplitude');
hold on;
plot(ar6, 'g');
plot(ar9, 'k');

plot(data(1:years),'r');
legend('AR2 Prediction','AR6 Prediction','AR9 Prediction','Real Data')
hold off;
```

List of Figures

3 Spectral estimation and modelling

Contents

3.0 Spectral Estimation introduction

PSD calculation gives a representation of the power present in each frequency. However in real applications it is difficult to correctly estimate it as all the samples of the signal cannot be obtained. Instead we use a periodogram which estimates the PSD by pacing values in each frequency bin. A periodogram function was written:

```
function [P, k] = pgm(data)
%returns PSD estimate in P with the normalized frequencies in k
%input is a set of input values which form the signal in time domain
%implements the function: P_X(k) = 1/N |Σ_{n=0}^{N-1} x[n] e^{-j2πkn/N}|
N = length(data);
k = (0:1/N:(N-1)/N);
P=zeros(N,1);
for n = 1:N
    e = exp((-1i*2*pi*(n-1)).*(0:1:N-1)'./N)';
    P(n) = (abs(sum((data.*e)))).^2./N;
end
end
```

The code was used to generate Figure 1 – PSD Estimates of a AWGN input for different samples sizes which shows the psd estimate for various sample sizes.

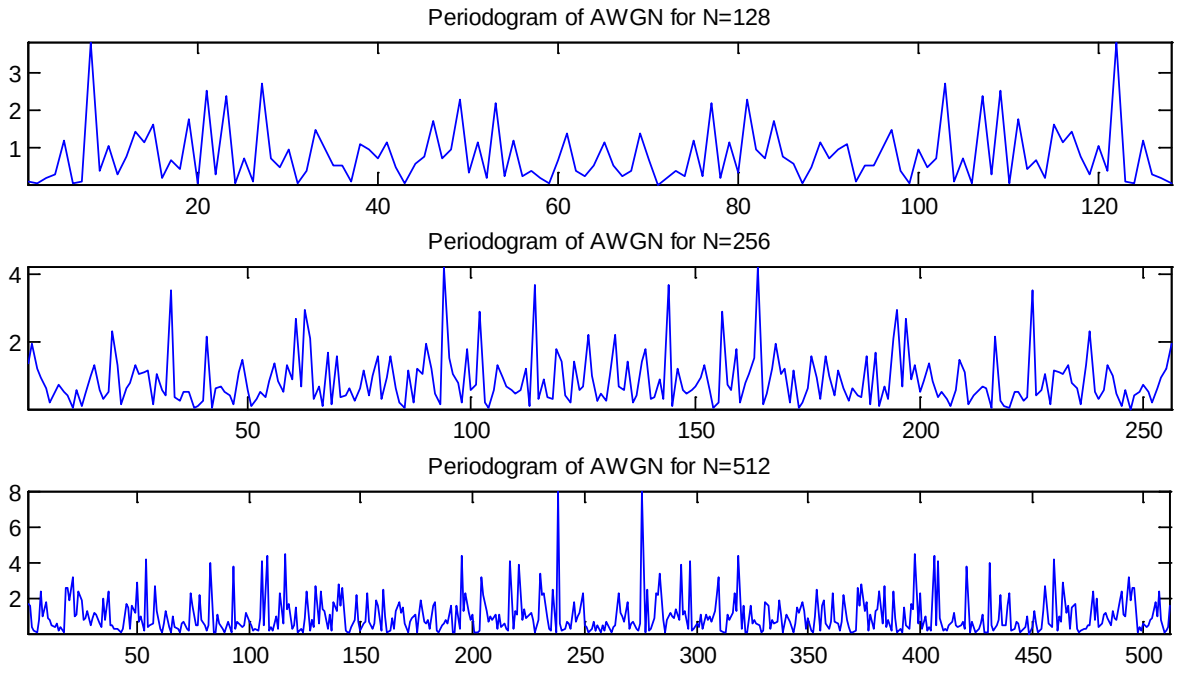


Figure 1 – PSD Estimates of a AWGN input for different samples sizes

These results seem reasonable if we remember that the PSD if the Fourier transform of the autocorrelation[1]. We have that $PSD_x(f) = \int_{-\infty}^{\infty} R_x(\tau) e^{-2\pi i f \tau}\, d\tau$ and due to having a Gaussian process we have $R_x(t) = {\sigma_x}^2 \delta(t)$ which means the expected $PSD_x(f)$ is

[1] http://mcise.uri.edu/chelidze/courses/mce567/handouts/psdtheory.pdf

$PSD_x(f) = \int_{-\infty}^{\infty} {\sigma_x}^2 \delta(t)\, e^{-2\pi i f \tau}\, d\tau = {\sigma_x}^2$. As the periodograms in Figure 1 are for very small sample sizes we can see that while we expect a uniform distribution which is somewhat visible –the main drawback is the very low number of samples.

3.1 Averaged Periodogram estimates

3.1.1 PSD estimate smoothing

As we expected a uniform distribution from the previous result it can be investigated in whether or not this is possible to obtain a 'better' uniform distribution by smoothing out the PSD estimate. This is what Figure 2 achieves. From it, it can be seen that the output does indeed look more like the expected uniform distribution. Additionally the result gets closer to the expected result the higher number of samples.

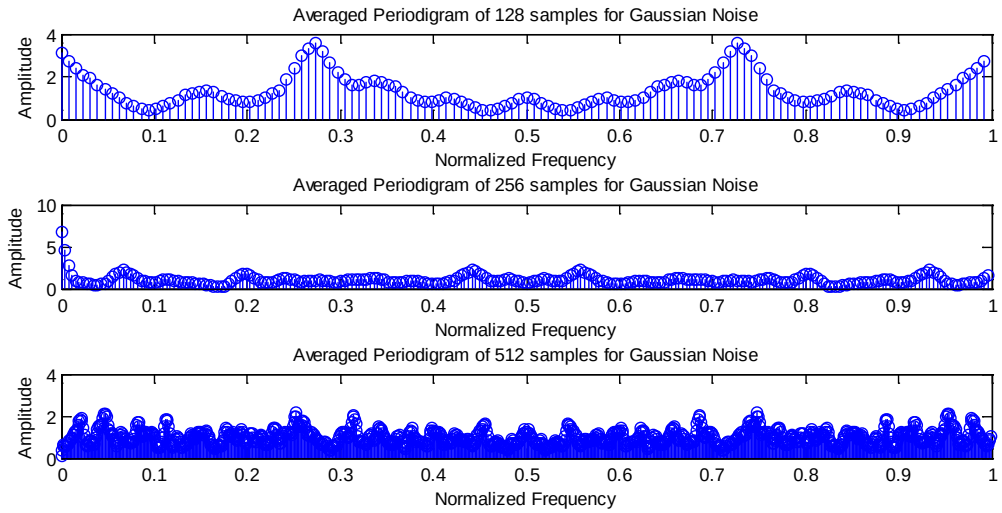


Figure 2 -Averaged Periodogram of Gaussian noise notice the contrast with Figure 1 and how this graph looks more like a uniform distribution – the expected result.

3.1.2 PSD estimate splitting

A 1024 sample sequence was generated using randn and divided into 8 sample segments as seen from Figure 3. As all values are independent splitting it up into portions of N=128 has not changed the overall shape compared to the 128 sample instance found in Figure 1. This means that the same inaccuracy remains and that the large variation is caused by the small sample size.

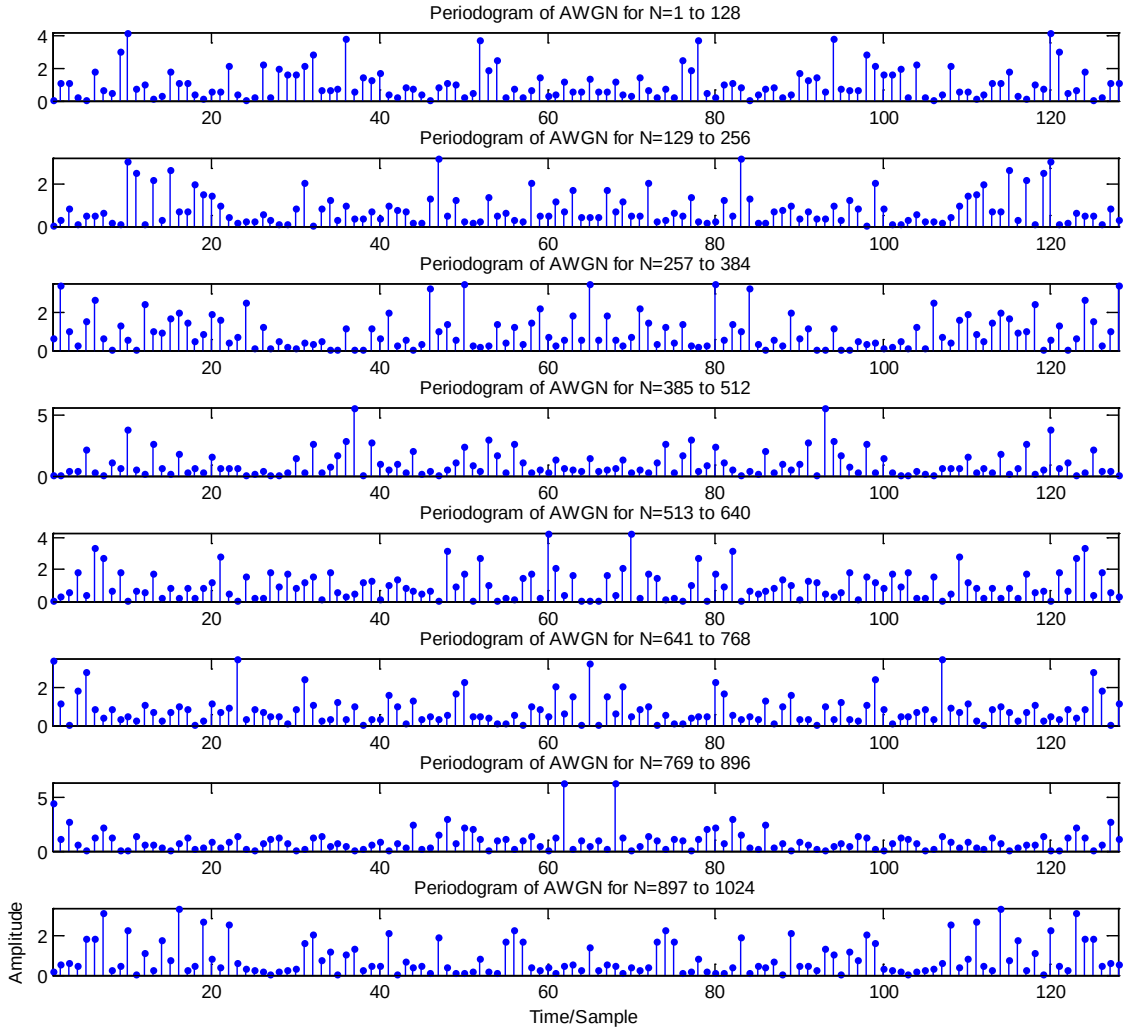


Figure 3 – 8-split of 1024 sample Gaussian noise data.

```
x=randn(1024,1);
for i=1:8
    subplot(8,1,i)
    [a,b]=pgm(x((i-1)*128+1:128*i)');
    plot(a)
    axis tight;
    str = sprintf('Periodogram of AWGN for N=%d to %d',(i-1)*128+1,128*i);
    title(str);
end
```

3.1.2 PSD estimate splitting and smoothing

In this part we take the average of all individual bins of Figure 4 and take the spectrogram. We see that the mean and standard deviation are now much closer to what is expected. Thus we can infer that the PSD splitting and smoothing leads to a better PSD estimate than the previous periodograms.

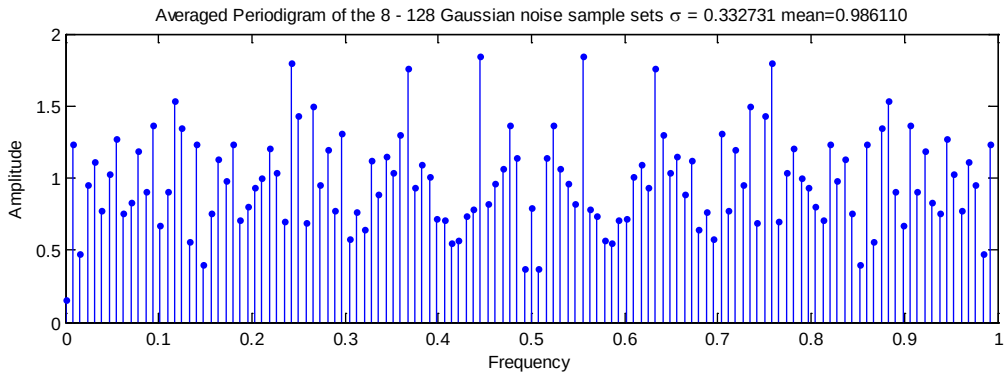


Figure 4 - Periodogram resulting from the average of the 8 periodograms of Figure 3.

3.2 Spectrum of autoregressive processes

In this section we look at a 1024-sample (technically 1064 with 40 subtracted) random noise passed through an AR1 process. This essentially acts as a high pass filter of the input and is apparent from Figure 5 where the output appears to be more "spiky" a feature of high frequency signals.

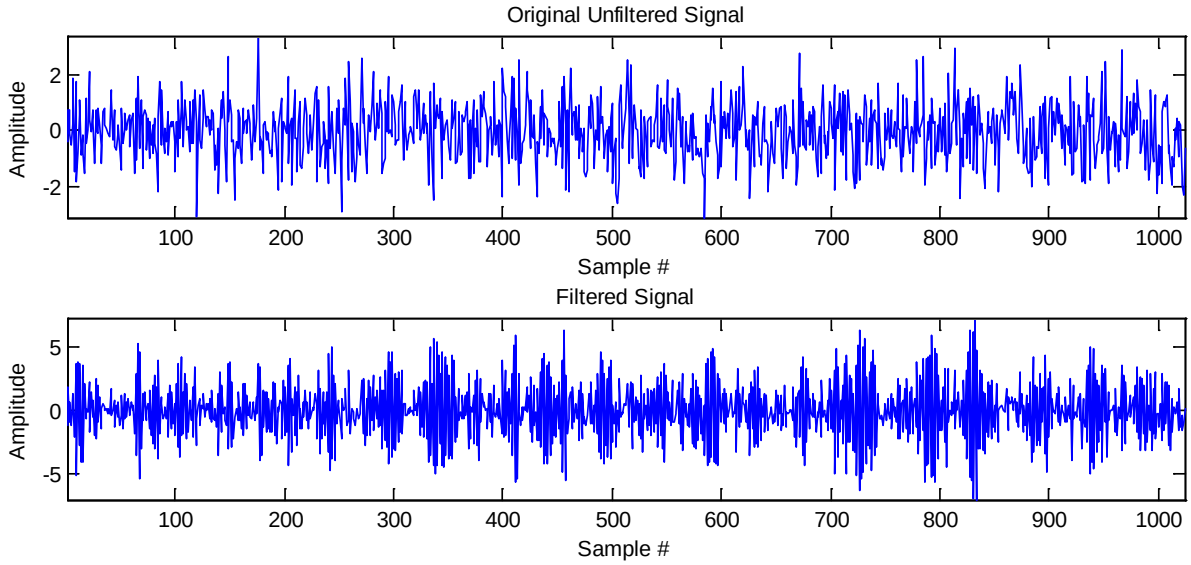


Figure 5 – Gaussian Noise data passed through a filter y=filter([1],[1 0.9],x)

The reason for the increase is due to the filter features. Indeed the relation $\sigma_y{}^2 = \frac{\sigma_x{}^2}{1-a_1{}^2}$ shows how the variances are related to each other. From this we can derive the expected variance of $\sigma_y{}^2 = \frac{1}{1-0.9^2} \cong 5.26$ which agrees with our findings.

3.2.1 Ideal PSD response

We investigate the nature of the PSD ideal estimate.

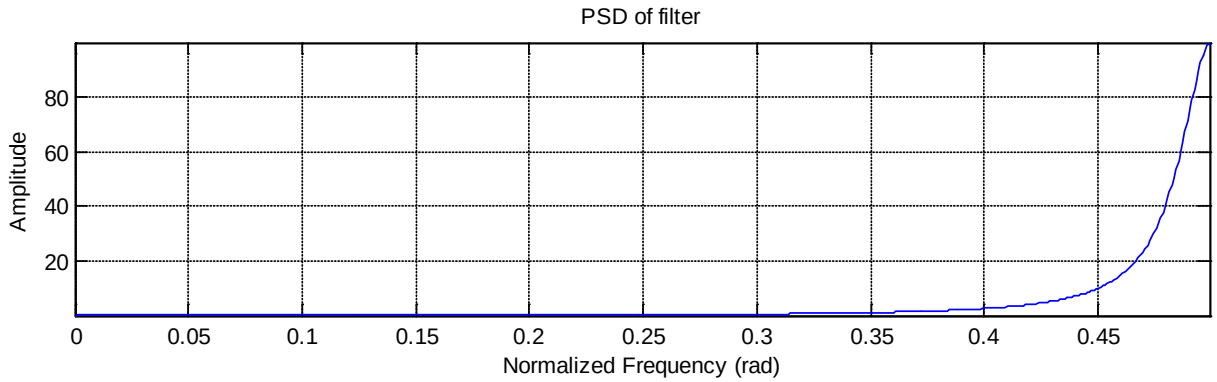


Figure 6 – PSD estimate of Y, generated by an AR1 process.

From the above graph of the PSD of Y ($P_Y(f) = \frac{1}{|1+0.9e^{-j2\pi f}|^2}$) , generated by an AR(1) process, it can be noted that most power is present in higher frequencies and that virtually

none exists in frequencies under 0.3 rad (normalized frequency) and most is present past 0.45 rad.

This due to the nature of $P_Y(f)$ which has a max for f (normalized) = 0.5.

3.2.2 Comparison to obtained data

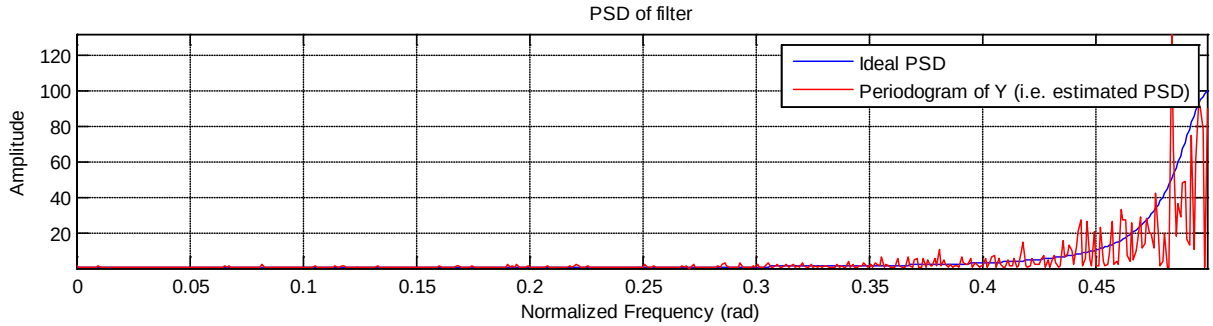


Figure 7 – Periodogram of Y plotted against the ideal PSD. The output periodogram can be compared to its input which would be a Gaussian signal similar to Figure 1.

The above graph compares the ideal PSD to the periodogram of Y. The first observation is that the AR1 does succeed in exaggerating high frequency components while minimizing the components in lower frequencies.

The differences are due to the collected data being performed on AWGN and that the data was simply taken as a whole to make the periodogram. Previous methods such as segmenting then averaging would give better estimates – this assigns the power more accurately to the correct frequencies (as a trade-off less frequency resolution is available).

The periodogram and the ideal while somewhat similar disagree to a certain extent.

This is due to 2 inter-related reasons:

- The periodogram is only an approximate of PSD
- The input is a discrete and limited in sample size whereas for an ideal PSD an infinite number of samples would be needed.

3.2.3 Effect of rectangular windowing

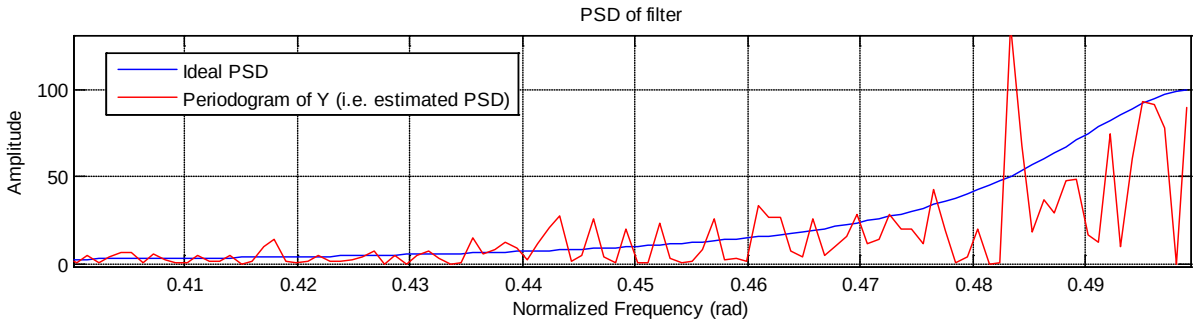


Figure 8 – Periodogram of Y plotted against the ideal PSD for a normalized frequency range of 0.4 to 0.5

Around 0.4 to 0.5 normalized frequency it can be seen that the filter is far from ideal. This would be due to the way that the periodogram calculates the output. This is first done by

windowing the data in time domain by a rectangular window. However in frequency domain this is equivalent to convoluting with a sinc function as $\mathcal{F}_{rect\left(\frac{t}{a}\right)}(f) = asinc(af)$. This causes the peak, theoretically at 0.5 normalized frequency , to be more spread out For example in Figure 8 it can be see that the peak it at around 0.48 – this is due to the combination of the sample size being small and the effect of rectangular windowing in time domain.

Additionally we can see that a small rect in time leads to a very large sinc in frequency – which is another reason larger sample sizes are needed

3.2.4 Model Estimation

Calculating the PSD by periodogram is possibly not the best estimation and another model based method is attempted. The idea of this model is to use observed values of the sampled data. This is because the auto-correlations can be expressed from the Yule-Walker equations as

$$r_{xx}(0) = -a_1 r_{xx}(1) + {\sigma_x}^2$$

$$r_{xx}(1) = -a_1 r_{xx}(0)$$

Rearranging these equations we get

$$\widehat{a_1} = \frac{-\hat{r}_{xx}(1)}{\hat{r}_{xx}(0)}$$

$$\hat{\sigma}_x^2 = \hat{r}_{xx}(0) + \widehat{a_1}\hat{r}_{xx}(1)$$

These values can be used to obtain and estimate of P_y i.e. $\hat{P}_y$.

This led to the following code:

```
corry = xcorr(y,'unbiased'); %% plot estimate
corry = xcorr(y,'unbiased');
%calculate a and sigma
a1 = -corry(2)/corry(1);
sigma_x = corry(1)+a1*corry(2);
%sigma_x = var(x); %alternative to line above
figure(3);
%get the data
[h,w]=freqz(sigma_x,[1 a1],512);
%plot estimate with model
plot(w/(2*pi),abs(h).^2); hold on;
plot(w/(2*pi),py(1:512),'r');hold off;
legend('Estimated','Periodogram of Y (i.e. estimated PSD)')
xlabel('Normalized Frequency (rad)')
ylabel('Amplitude')
axis tight;
grid on;
str = sprintf('Estimated PSD of AR(1)\n \\sigma_x = %f a_1=%f',sigma_x,a1);
title(str);
```

3.2.5 Model Comparison

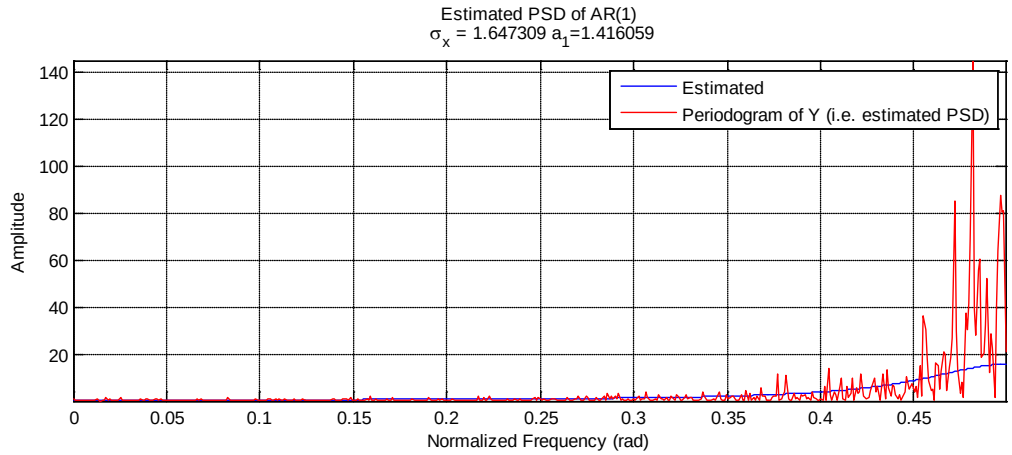


Figure 9 - Periodogram of Y plotted against the estimated PSD

As can be seen the estimated PSD from $\hat{P}_Y(f) = \frac{\hat{\sigma}_x{}^2}{|1+\hat{a}_1 e^{-j2\pi f}|^2}$ in Figure 9 the estimated model has the same shape as the periodogram but with a much reduced amplitude. This is due to the scaling heavily relying on the variance estimate $\hat{\sigma}_x{}^2$ which is extremely sensitive due to the way it relies on a set of data with very small values. Thus this model while good at estimating the shape (due to a_1 being reasonable accurate) does not work very well in estimating the amplitude.

3.3 Spectrogram example: dial tone pad

Here we generate the sounds generated when using old-style phones which use the DTMF (Dual Tone Multi-Frequency) dialling system. In this system each number is assigned pair of frequencies as seen in Figure 10. After generation there is hope of being able to decode the frequencies and correctly determine which symbol was being represented.

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

Figure 10 – DTMF Frequency pairs for different keys

3.3.1 Number generation

N random London phone numbers were generated by the following code in matlab:

```
l_phone = floor(10*rand(N,11));
l_phone(:,1:3) = ones(N,1)*[0 2 0];
```

From this it was possible to generate the discrete time samples of a London phone number:

```
N = 1;
Fs = 32768; %set sampling frequency
```

```
%generate phone numbers
l_phone = floor(10*rand(N,11));
l_phone(:,1:3) = ones(N,1)*[0 2 0];
f1 = zeros(11,1);
f2 = zeros(11,1);
% assign frequencies
for i=1:11
    switch mod(l_phone(1,i),3)
        case 0
            f1(i) = 1477;
        case 1
            f1(i) = 1209;
        case 2
            f1(i) = 1366;
    end
    switch floor((l_phone(1,i)-1)/3)
        case 0
            f2(i) = 697;
        case 1
            f2(i) = 770;
        case 2
            f2(i) = 852;
    end
    %handle exception
    if l_phone(1,i) == 0
        f1(i) = 1336;
        f2(i) = 941;
    end
end
tone = zeros(1,180224);
%generate sound using y[n] = sin(2πf1n) + sin(2πf2n)

for i=1:11

    for t=(1+Fs/2*(i-1)):(Fs/4*i + Fs*(i-1)/4)
        tone(t) = (sin(2*pi*f1(i)*(t)/Fs)+sin(2*pi*f2(i)*t/Fs))/2;
    end
    %silent section for 0.25 seconds
    for t=((1+Fs/4*i)+ Fs*(i-1)/4):Fs/2*i
        tone(t) = 0;
    end
end
```

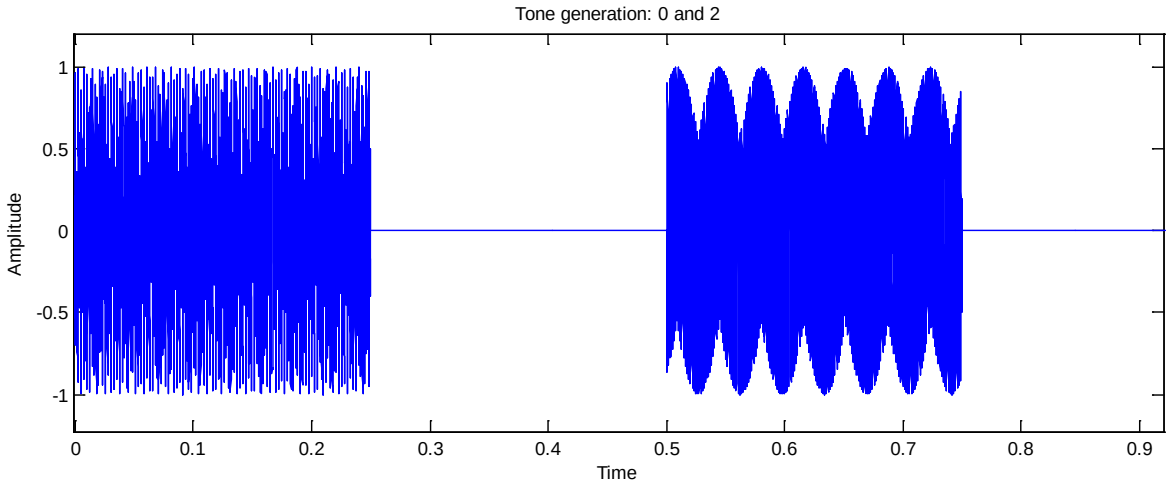


Figure 11 – Time plot of first two key presses of a London number, 0 and 2

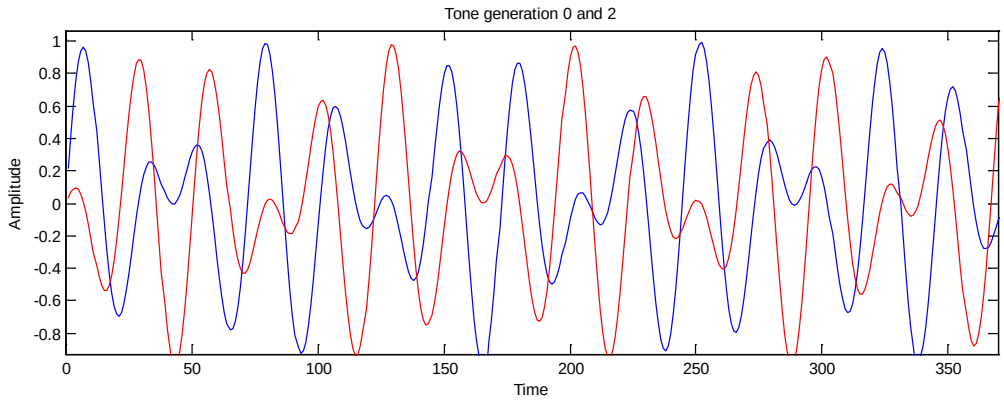


Figure 12 – In blue is the time domain plot of the tone for 0(941 and 1336Hz) and in red is the plot for 2(697 and 1336Hz). It can be seen that they have different frequency components but identification of which is which is not simple.

3.3.2 & 3.3.3 Frequency – Spectogram analysis

Note – while matlab provides a spectrogram function Mike Brookes has written a possibly more difficult to use but way more configurable spectrogram function as part of his voicebox toolbox[2].

Figure 13 and Figure 14 show the spectrogram of 2 random London telephone numbers with different sensitivity settings. From these we can see that the frequencies can be accurately separated and identified, leading to key identification being possible. Indeed the first frequency pair can be seen in Figure 14 to be just under 950hz and just above 1300 Hz, clearly indicating that this was a 0.

Thus key classification in frequency domain is a definite possibility and explains why this standard is so widespread.

[2] http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html

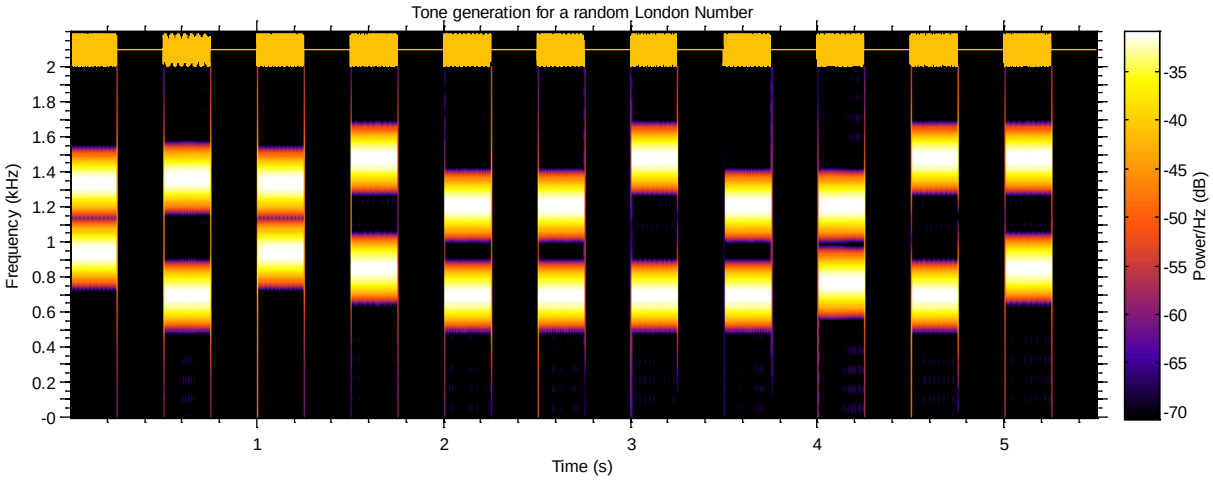


Figure 13 – Spectrogram of a random London telephone number using amplitude sensitive settings.

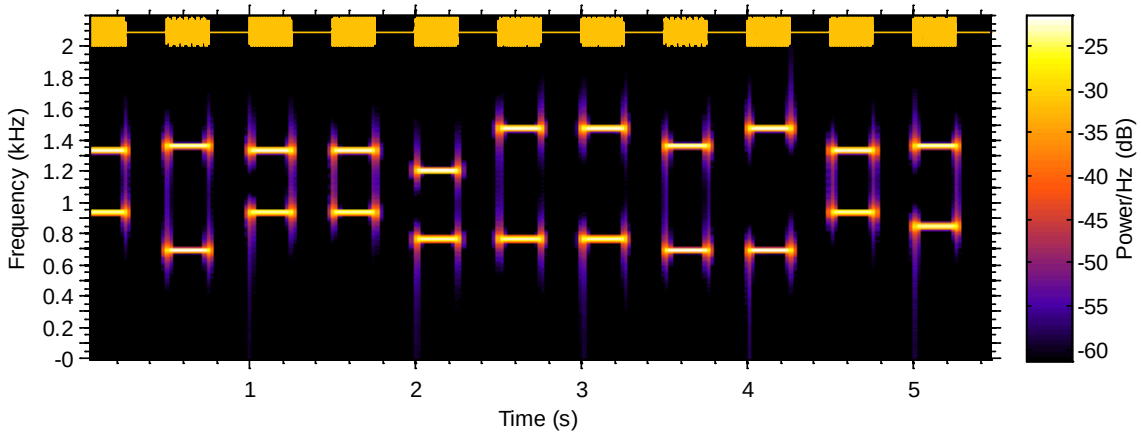


Figure 14 - Spectrogram of a random London telephone number using less amplitude sensitive settings to more clearly distinguish the frequencies between each other.

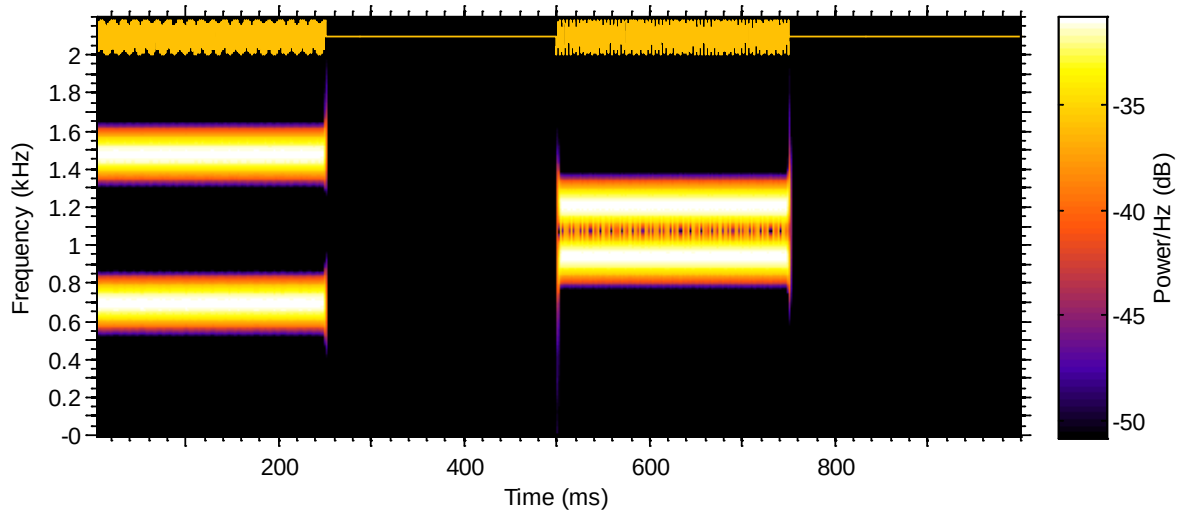


Figure 15 – Spectrogram of first two keys – 0 and 2

3.3.4 Analysis in noisy environment

In the previous tests the signal was not corrupted by noise – however this is not representative of real life scenarios. In this scenario we investigate what happens when the received signal is[3] $y[n] = \sin(2\pi f_1 n) + \sin(2\pi f_2 n) + \sigma_n n[n]$ and we will determine what the minimum workable SNR could be.

In Figure 16 the SNR is 30 and the traces of the noise can be seen lightly but overall the tone frequencies are easily distinguishable .

Figure 17, with an SNR of 11dB, the noise while very visible does not interfere with visual identification of the sounds still possible.

Figure 18, with a negative SNR of -3dB the signal can still be identified however the noise starts to get visibly worse. The time domain plot (just above the spectrogram) starts showing the noise to be able to be confused with a tone period.

From Figure 19 which has an SNR of -18dB the signal is nearly indistinguishable from the noise with only faint traces of the signal remaining. By looking at the time domain plot it is very difficult to distinguish a pause from a tone.

Overall it is possible to determine that identifying the tones by spectral analysis is a modestly robust method due to the amount of noise coped with. This is due to the dual tone nature of this system where most power is concentrated exclusively at 2 frequencies.

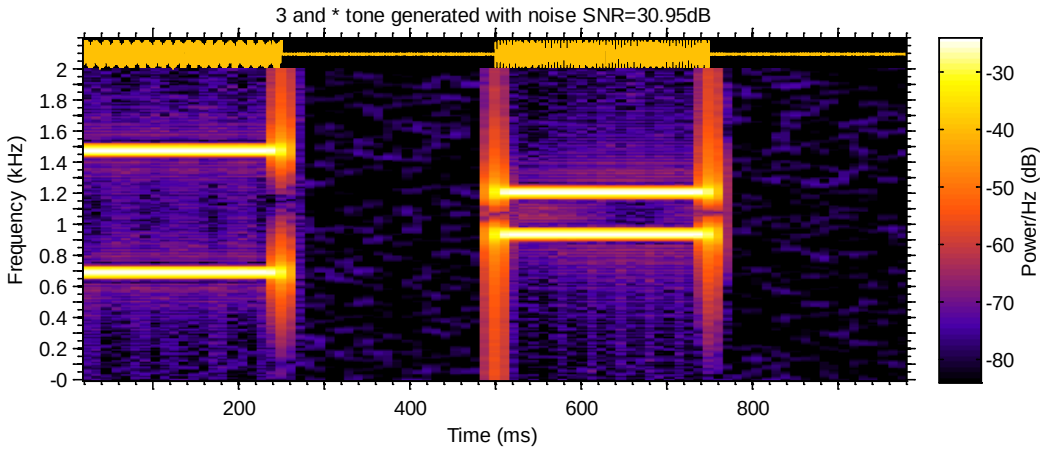


Figure 16– 3 and * tones generated with AWGN noise of $\sigma_n = 0.01$. Above is a small time domain version of the signal.

[3] $n[n]$ is zero mean Gaussian

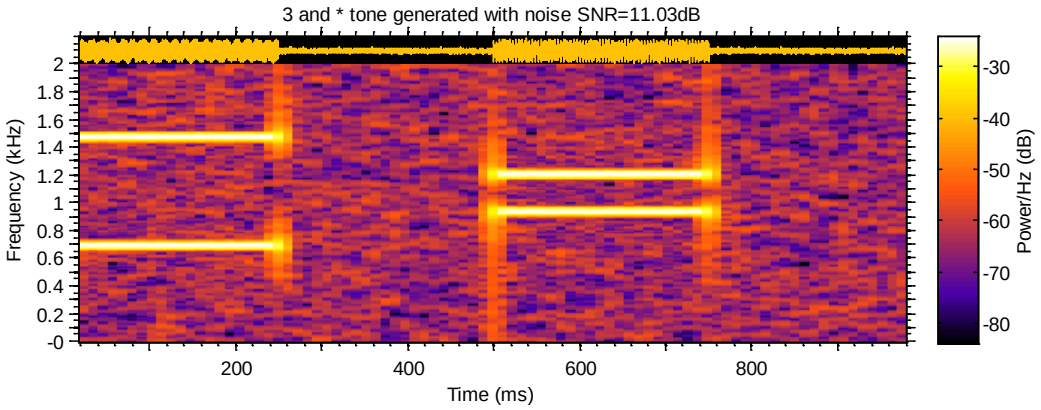


Figure 17– 3 and * tones generated with AWGN noise of $\sigma_n = 0.1$

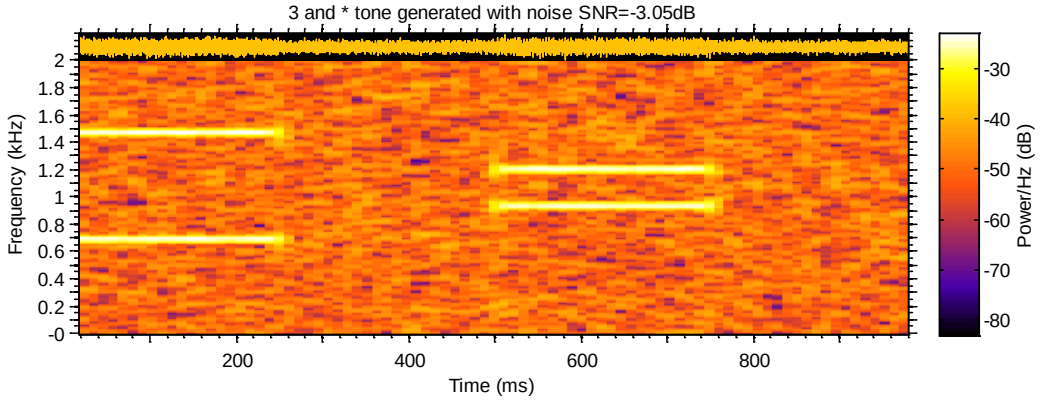


Figure 18– 3 and * tones generated with AWGN noise of $\sigma_n = 0.5$

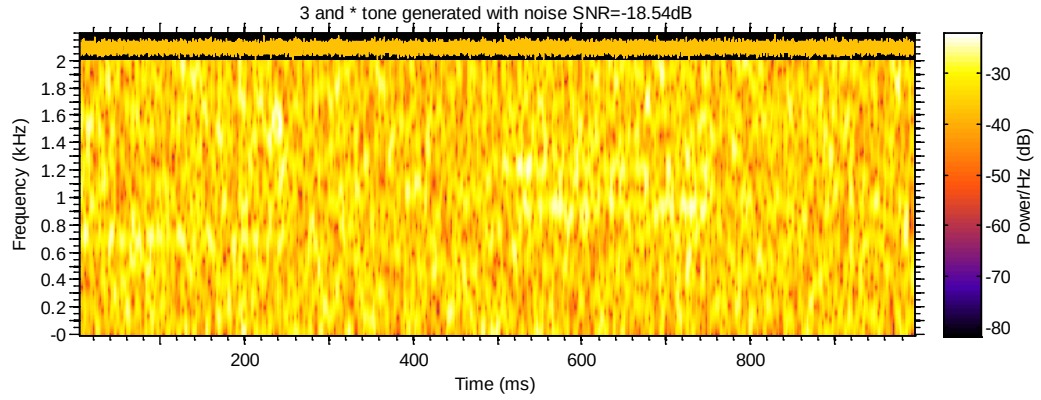


Figure 19 – 3 and * tones generated with AWGN noise of $\sigma_n = 3$

3.4 Spectrum of dialled signals using AR modelling

As seen in section 3.2 the PSD of a signal can be estimated by trying to model it to an AR process. This allows us to model the spectrum by the function

$$P_y(f) = \frac{{\sigma_x}^2}{\left|1 - \sum_{k=1}^{p} a[k]e^{-\mathrm{j}2\pi\mathrm{f}k}\right|^2}$$

Which in matlab gives us:

```
[h,w] = freqz(σ_x^2,a_k,512,Fs)
plot(w/(2*pi),abs(h).^2)
```

This essentially plots the estimate of the AR process. This is possible due to the way an AR process can be modelled. Consider the following AR2 process[4]:

$P_y(f) = \frac{{\sigma_x}^2}{\left|1-a_1e^{-j2\pi f}-a_2e^{-j4\pi f}\right|^2}$ with poles respectively at $p_1, p_2 = \frac{1}{2}\left(a_1 \pm \sqrt{a_1^2 + 4a_2}\right)$

When the poles form a complex conjugate i.e. $0 < a_1^2 + 4a_2$ then we get $\cos(2\pi f) = \frac{(1-a_2^2)(4a_2+a_1^2)}{4a_2}$, which when rearranged gives $f = \frac{1}{2\pi}cos^{-1}\left(\frac{(1-a_2^2)(4a_2+a_1^2)}{4a_2}\right)$.

The reason we need an AR4 process at minimum for the 3 and * dial tones draws from the reasoning above. Indeed to create an oscillation or a peak in frequency we need a complex pair (conjugate). In the case of 3 and * which are made from a dual-tone system there is a need to create 2 oscillations (or equivalently 2 frequency peaks), meaning that for 2 conjugate pairs to exist and AR process of 4 is needed at minimum.

In this section we take the 3 and * tones and look at the various estimated PSD under various noise conditions.

In Figure 20,Figure 21 and Figure 22 we notice that the higher the SNR the more prevalent the peak whereas the lower the SNR gets the more spread out the PSD is, showing the effect of noise corruption. We however notice than an AR4 process is not enough to model the two expected peaks as only one peak appears in all the tested processes.

This is essentially because the AR4 process takes an estimate which leads to outputting a "blurred" PSD of a model of the sinusoids with the noise (after all the AR process is simply estimating what is given to it which also means the noise).

[4] http://www.ece.umd.edu/class/enee630.F2012/slides/part-3_sec2_slidesAll.pdf

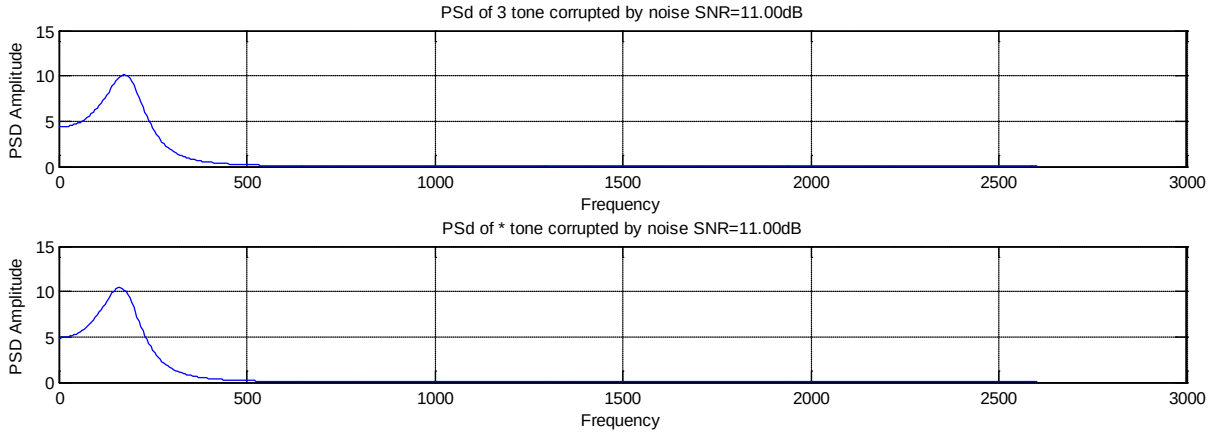


Figure 20 – AR4 PSD estimate model of both signals with a SNR of 11

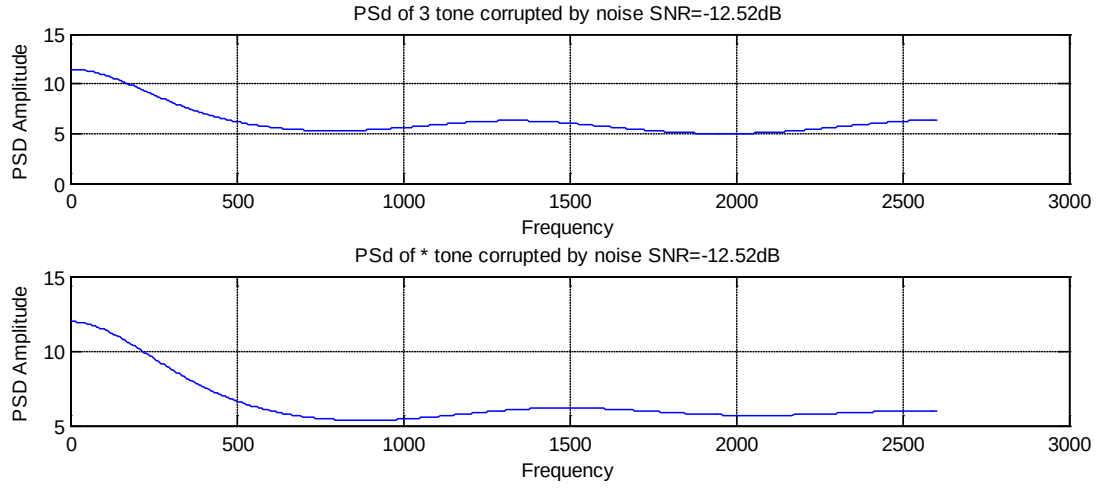


Figure 21 - AR4 PSD estimate model of both signals with a SNR of -11

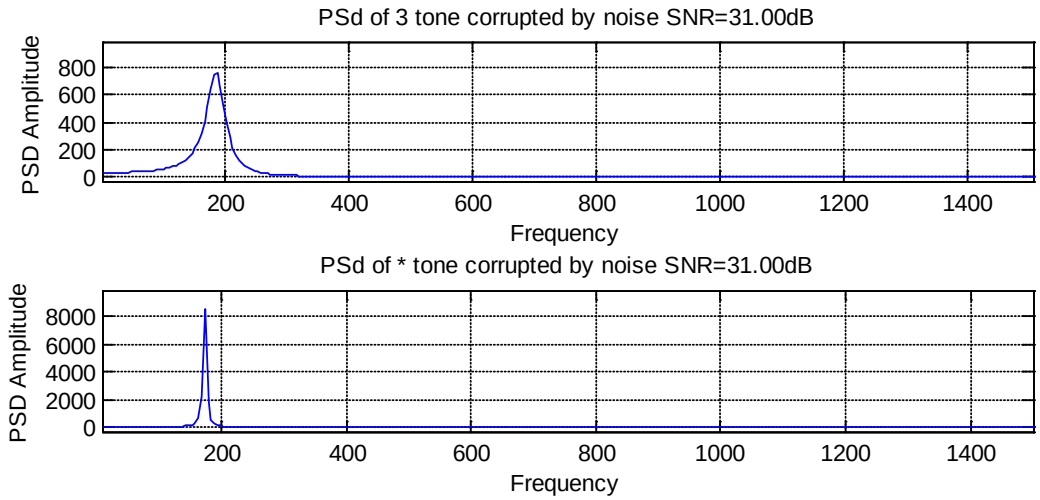


Figure 22 - AR4 PSD estimate model of both signals with a SNR of 31

Seeing how an AR4 seems to not show 2 peaks it was tried on a clean input as seen in Figure 23 and yet again AR4 is not enough. Thus it was chosen to increase the order to see if an order capturing 2 peaks existed. Figure 24 and Figure 25 show that for various SNR values a higher order does indeed give rise to the 2 expected peaks which are so useful in frequency identification. Indeed for a relatively noisy signal with -3dB an order of 50 was able to identify the 2 peaks and an order of 25 was needed to identify peaks with an SNR of 90dB.

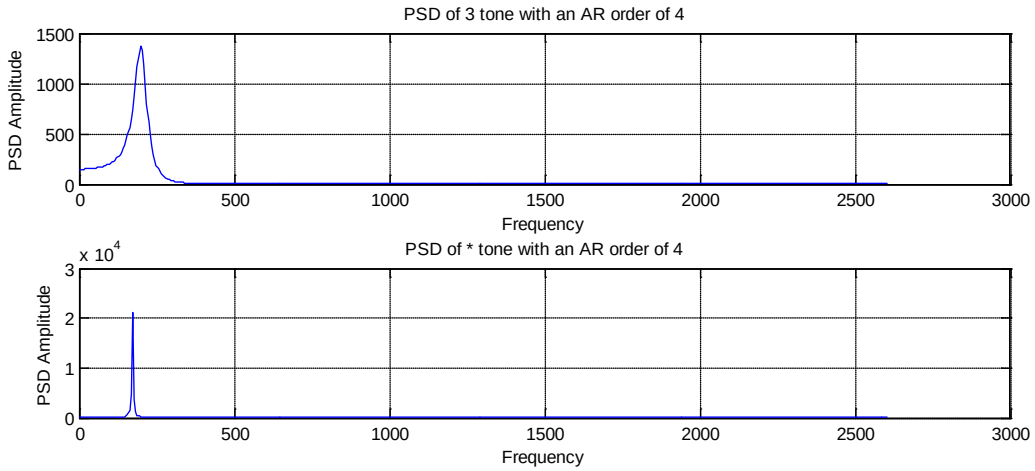


Figure 23 – AR4 PSD estimate of a clean signal

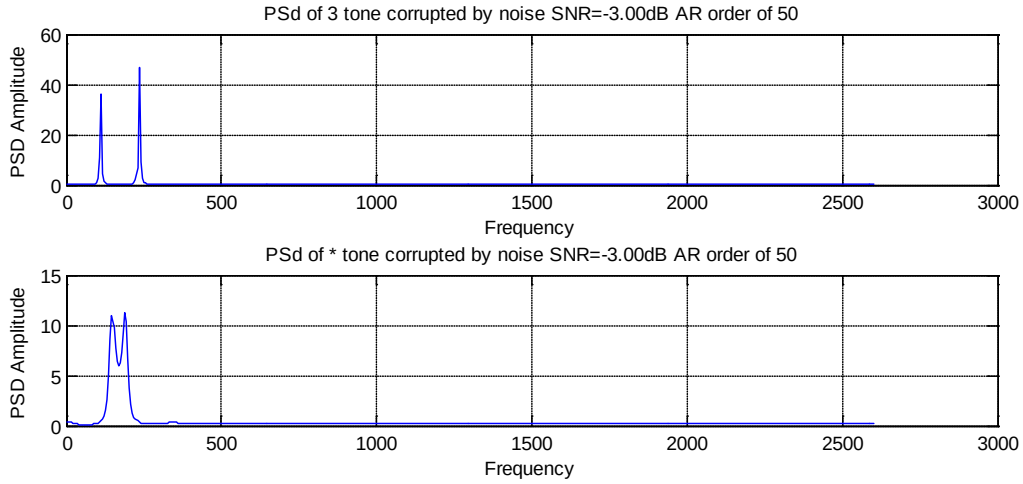


Figure 24 – AR50 PSD estimate of a corrupted signal at -3dB

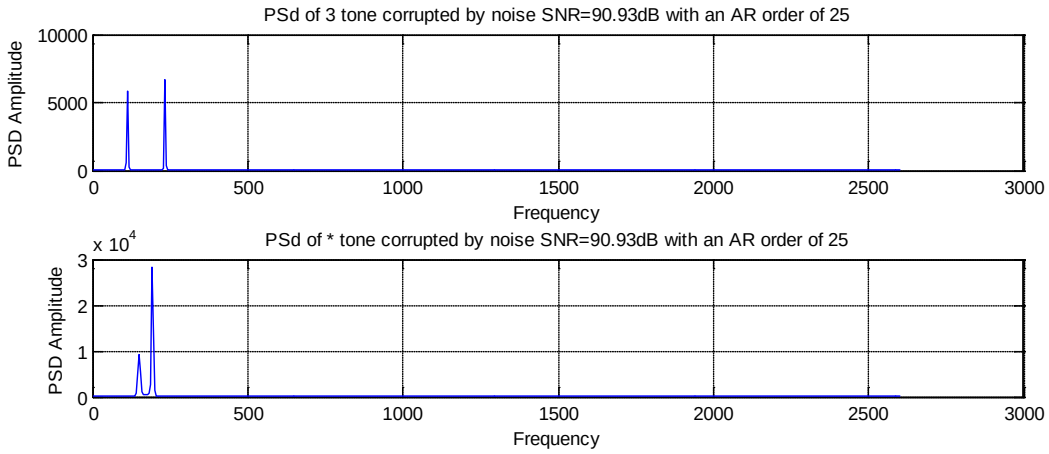


Figure 25– AR25 PSD estimate of a corrupted signal at 90dB

Appendix

Table of Figures

Matlab code

Periodogram function

```
function [P, k] = pgm(data)
%returns PSD estimate in P with the normalized frequencies in k
%input is a set of input valuesn which form the signal in time domain
N = length(data);
k = (0:1/N:(N-1)/N);
P=zeros(N,1);
for n = 1:N
```

```
    e = exp((-1i*2*pi*(n-1)).*(0:1:N-1)'./N)';
    P(n) = (abs(sum((data.*e)))).^2./N;
end
end
```

Part 3.1

```
clc;
clear all;
close all;
x=randn(1024,1);
c=zeros(128,8);
for i=1:8
    [c(:,i),~] = pgm(x((i-1)*128+1:128*i)');
end
c = mean(c');
%%
[a,b]=pgm(c);
figure(1);
stem(c,'.')
axis tight;
str = sprintf('Periodogram of AWGN average of 8 sample sets');
title(str);
figure(2);

%filtfilt(0.2*[1 1 1 1 1],1,a); %used for 3.1.1
for i=1:8
    subplot(8,1,i)
[a,b]=pgm(x((i-1)*128+1:128*i)');
stem(a,'.')
axis tight;
str = sprintf('Periodogram of AWGN for N=%d to %d',(i-1)*128+1,128*i);
title(str);
end
xlabel('Time/Sample');
ylabel('Amplitude')
```

Part 3.2

```
clc;
clear all;

figure(1)
x=randn(1064,1);
y=filter([1],[1 0.9],x);
y=y(41:1064);
x=x(41:1064);
subplot(2,1,1)
plot(x)
xlabel('Sample #')
ylabel('Amplitude')
axis tight;
str = sprintf('Original Unfiltered Signal');
title(str);
subplot(2,1,2)
plot(y)
xlabel('Sample #')
ylabel('Amplitude')
```

```
axis tight;
str = sprintf('Filtered Signal');
title(str);
[h,w]=freqz([1],[1 0.9],512);
figure(2)
py = pgm(y');
plot(w/(2*pi),abs(h).^2); hold on;
plot(w/(2*pi),py(1:512),'r');hold off;
legend('Ideal PSD','Periodogram of Y (i.e. estimated PSD)')
xlabel('Normalized Frequency (rad)')
ylabel('Amplitude')
axis tight;
grid on;
str = sprintf('PSD of filter');
title(str);
%% plot estimate
corry = xcorr(y,'unbiased');
%calculate a and sigma
a1 = -corry(2)/corry(1);
sigma_x = corry(1)+a1*corry(2);
%sigma_x = var(x); %alternative to line above
figure(3);
%get the data
[h,w]=freqz(sigma_x,[1 a1],512);
%plot estimate with model
plot(w/(2*pi),abs(h).^2); hold on;
plot(w/(2*pi),py(1:512),'r');hold off;
legend('Estimated','Periodogram of Y (i.e. estimated PSD)')
xlabel('Normalized Frequency (rad)')
ylabel('Amplitude')
axis tight;
grid on;
str = sprintf('Estimated PSD of AR(1)\n \\sigma_x = %f a_1=%f',abs(sigma_x),a1);
title(str);
% [a,b]=pgm(x((i-1)*128+1:128*i)');
% %filtfilt(0.2*[1 1 1 1 1],1,a);
% plot(a)
% axis tight;
% str = sprintf('Periodogram of AWGN for N=%d to %d',(i-1)*128+1,128*i);
% title(str);
```

Part 3.3

```
clear all;
clc;
close all;
addpath('voicebox');
N = 1;
Fs = 32768;
%generate phone numebrs
l_phone = floor(10*rand(N,11));
l_phone(:,1:3) = ones(N,1)*[0 2 0];
f1 = zeros(11,1);
f2 = zeros(11,1);
% assign frequencies
for i=1:11
    switch mod(l_phone(1,i),3)
        case 0
            f1(i) = 1477;
        case 1
```

```
            f1(i) = 1209;
        case 2
            f1(i) = 1366;
    end
    switch floor((l_phone(1,i)-1)/3)
        case 0
            f2(i) = 697;
        case 1
            f2(i) = 770;
        case 2
            f2(i) = 852;
    end
    %handle exception
    if l_phone(1,i) == 0
        f1(i) = 1336;
        f2(i) = 941;
    end
end
tone = zeros(1,180224);
%generate sound
for i=1:11

    for t=(1+Fs/2*(i-1)):(Fs/4*i + Fs*(i-1)/4)
        tone(t) = (sin(2*pi*f1(i)*(t)/Fs)+sin(2*pi*f2(i)*t/Fs))/2;
    end
    %silent section for 0.25 seconds
    for t=((1+Fs/4*i)+ Fs*(i-1)/4):Fs/2*i
        tone(t) = 0;
    end
end
figure(1);
i=1;
range = (1+Fs/2*(i-1)):(Fs/4*i + Fs*(i-1)/4);
plot(range,tone(range));hold all;
i=4;
range = (1+Fs/2*(i-1)):(Fs/4*i + Fs*(i-1)/4);
plot((1:Fs/4),tone(range),'r');hold off;
title('Tone generation 0 and 2')
xlabel('Time');
ylabel('Amplitude');
figure(2);
spgrambw(tone, Fs, 'Jcw',20,2000);
title('Tone generation for a random London Number')
% spectrogram(tone,hann(8192),0,512,Fs)
% view(-90,90)
% set(gca,'ydir','reverse')
% spectrogram(tone,Fs/4,Fs/4-40,256,Fs);%[S,F,T,P] =
%surf(T,F,10*log10(P),'edgecolor','none'); axis tight;

%xlabel('Time (Seconds)'); ylabel('Hz');
% soundsc(tone,Fs)
```

Part 3.4

```
clear all;
clc;
close all;
addpath('voicebox');
order = 25;
N = 1;
```

```
Fs = 32768;
f1 = [1477 1209];
f2 = [697 941];
tone = zeros(1,Fs);
awgn = 0.00001*randn(Fs,1);
for i=1:2
    fprintf('start:%i   end:%i\n',(1+Fs/2*(i-1)),(Fs/4*i + Fs*(i-1)/4));
    for t=(1+Fs/2*(i-1)):(Fs/4*i + Fs*(i-1)/4)
        tone(t) = (sin(2*pi*f1(i)*(t)/Fs)+sin(2*pi*f2(i)*t/Fs))/2;
    end
    for t=((1+Fs/4*i)+ Fs*(i-1)/4):Fs/2*i
        tone(t) = 0;
    end
%     tone(i) = sin(2*pi*f1*i/Fs);
end
ptone = 0;
pnoise = 0;

for i = 1:length(tone)-1
    ptone = ptone + tone(i).^2;
    pnoise = pnoise + awgn(i).^2;
    tone(i) = tone(i) + awgn(i);
end
SNR = 10*log10(ptone/pnoise);
% plot((1:Fs/2*11)/Fs,tone)
figure(1);
plot((1:Fs)/Fs,tone)
figure(2);
xlabel('Time');
ylabel('Amplitude');
spgrambw(tone, Fs, 'Jcw',40,2000,60, [0 1]);
str = sprintf('3 and * tone generated with noise SNR=%2.2fdB',SNR);
title(str)
% soundsc(tone,Fs)
ar_three = aryule(tone(1:8192),order);
ar_star = aryule(tone(16385:24576),order);
std_three = std(tone(1:8192));
std_star = std(tone(16385:24576));
figure;
[h,w] = freqz(std_three^2,ar_three,512,Fs);
subplot(2,1,1)
plot(w/(2*pi),abs(h).^2)
grid on
xlabel('Frequency');
ylabel('PSD Amplitude');
str = sprintf('PSd of 3 tone corrupted by noise SNR=%2.2fdB with an AR order of %d',SNR,order);
title(str)
subplot(2,1,2)
[h,w] = freqz(std_star^2,ar_star,512,Fs);
plot(w/(2*pi),abs(h).^2)
grid on
xlabel('Frequency');
ylabel('PSD Amplitude');
str = sprintf('PSd of * tone corrupted by noise SNR=%2.2fdB with an AR order of %d',SNR,order);
title(str)
```

Part 4: A Real World Case Study – Vinyl denoising

Sebastian Grubb (sg3510)

Contents

4.1 Noise Identification

Here we take the periodogram of the file with noise to try and identify it. Figure 1 and Figure 2 were made to assist in the decision.

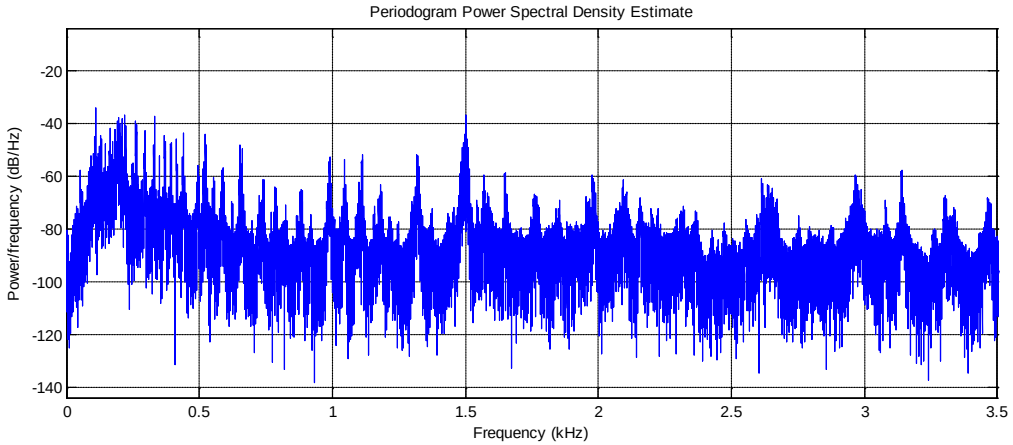


Figure 1 – Periodogram PSD estimate of channel 1 of the noisy sample of Stairway to Heaven

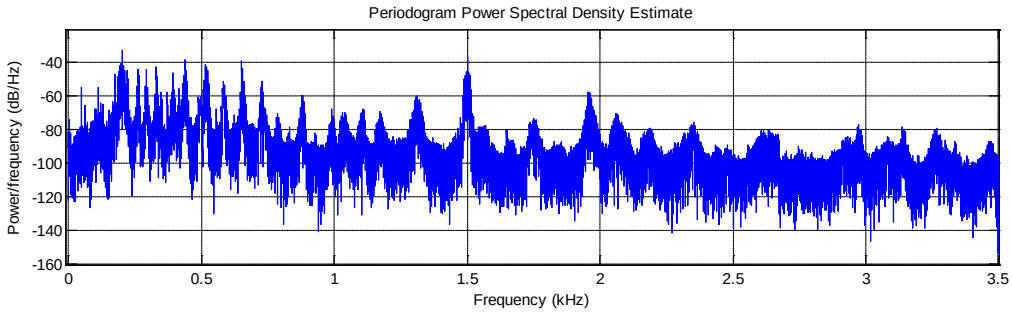


Figure 2 – Periodogram PSD estimate of channel 2 of the noisy sample of Stairway to Heaven

It is possible to notice a very prominent peak at 1.5kHz as well as a smaller one around 200 Hz on channel 2 from the periodograms which would correspond to the intermittent sound heard while playing the track back. However with many other peaks at different frequencies further investigation will have to be done to identify if these frequencies are indeed the noise or not. One method to do so would be to isolate the sections where the noise is heard to sections where no noise is heard and compare the periodograms.

4.2 Identification with clean track

By having the original track the identification of noise if made as a one-to-one comparison can be done.

The first step is to compare, by eye, the differences of the periodograms of the two data sets. From comparing Figure 1 with Figure 3 and Figure 2 with Figure 4 it is possible to compare the two data sets to see where noise is found due to differences in the spectrum.

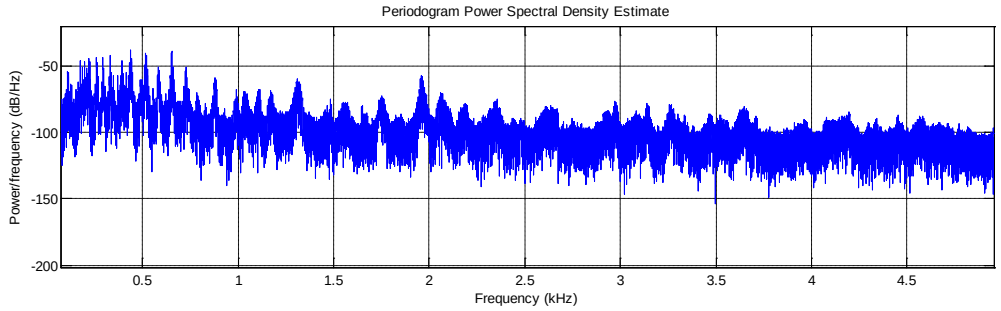


Figure 3 – Periodogram PSD estimate of clean channel 1

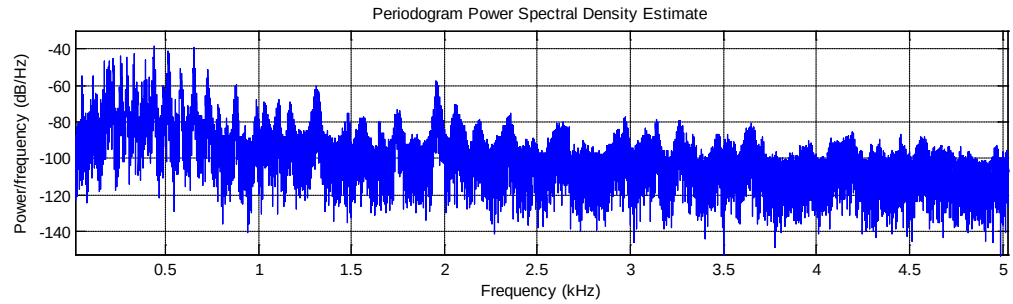


Figure 4 - Periodogram PSD estimate of clean channel 2

Comparing by eye is not the most effective was to extract the noise data, instead two (similar) ways were used to more effectively compare data and find the noise.

The first method consisted of taking the differences between the clean and noisy in frequency domain, giving rise to Figure 5. It allows clear identification of the noise's frequency components as well as the bandwidth the noise occupies.

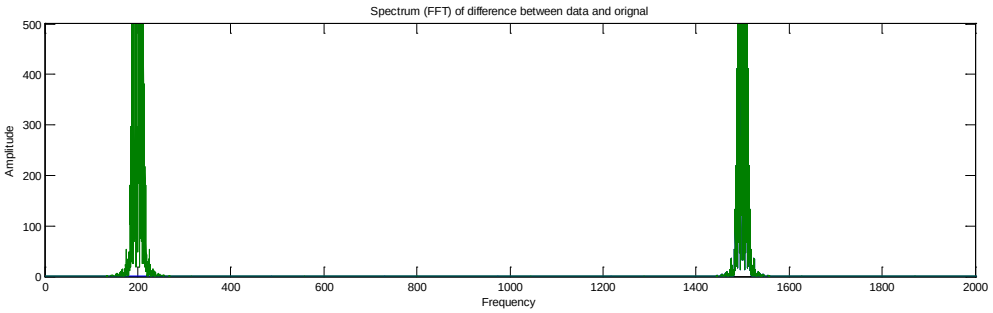


Figure 5 – FFT of the differences between channel 1 (blue) and channel 2 (green) of the clean and noisy signal.

The second method would be to take the differences between clean and noisy in terms of periodogram PSD estimation. In Figure 6 the main noise component in channel 1 can be identified at 1500 Hz and in Figure 7 helps identify the noise spectral components at 200 and 1500 Hz.

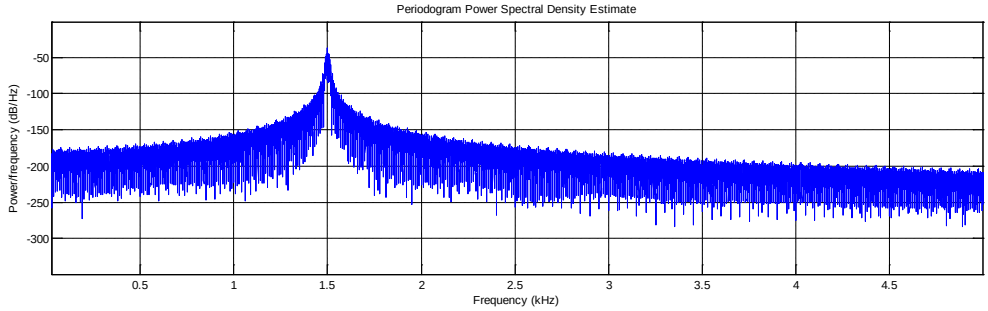


Figure 6 – Periodogram PSD estimate of difference between clean and noisy signal channel 1

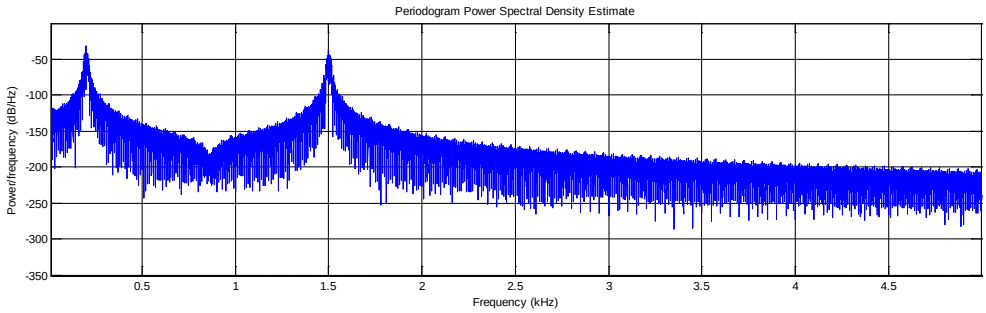


Figure 7 - Periodogram PSD estimate of difference between clean and noisy signal channel 2

4.3 Removing Noise

Once noise has been identified in terms of its spectral components removing it simply consists of constructing the appropriate filters with the intent of removing these frequencies and not harm the rest too much.

Using Butterworth filters designed in matlab the noise from the corrupted track was removed.

This was done by choosing the noise with the appropriate bandwidth to suppress the correct pairs of frequency ranges.

From Figure 5 frequency pairs of 150 to 250Hz for channel 2 and 1420 to 1560 were chosen as they covered the complete spectrum.

```
data=s2h;
[b,a]=butter(3, [1420 1560]/Fs*2, 'stop');
data = filtfilt(b,a,data); %filtfilt used instead of filter to avoid change in phase
[b,a]=butter(3, [150 250]/Fs*2, 'stop');
data(:,2) = filtfilt(b,a,data(:,2));
```

By listening to the output it was determined that such filters were effective and hearing a difference with the clean signal was difficult meaning that the filters had achieved their intent.

4.4 Analysis of Noise removal

Taking the periodograms of the original and the processed data it is possible to appreciate the improvement visually. From Figure 8 it is possible to see that noise is now gone from the

frequencies where it would otherwise be expected but that the rest of the signal is intact, with the same other frequency components - by comparing it to the original clean data.

From Figure 9 it can be seen that the filters are aggressive in terms of spectral removal giving rise to another difference which explains the existence of the peaks at the filtered-out frequencies. However the relative errors of Channel 1 (0.007378) and Channel 2 (0.231638) suggest that noise removal was indeed effective.

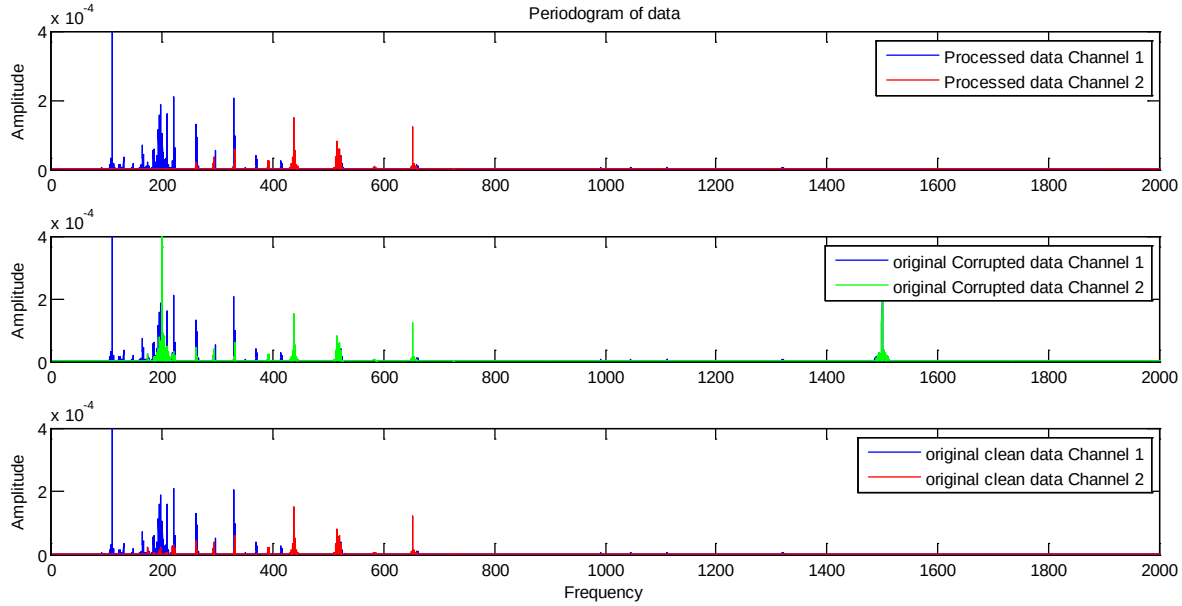


Figure 8 - Periodogram of clean signal, output and input.

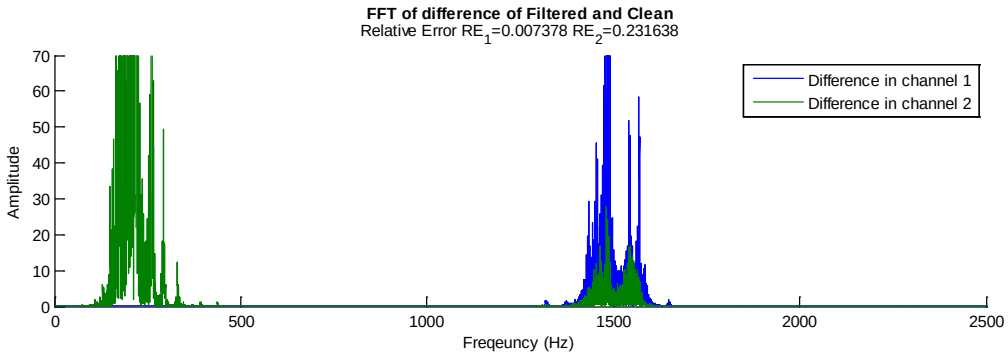


Figure 9 – FFT of difference between filtered signal and clean signal. Relative error is calculated by $\frac{\|P_c - \widehat{P_c}\|}{\|P_c\|}$.

4.5 Supervised adaptive learning algorithm

4.5.1 Optimal Coefficients

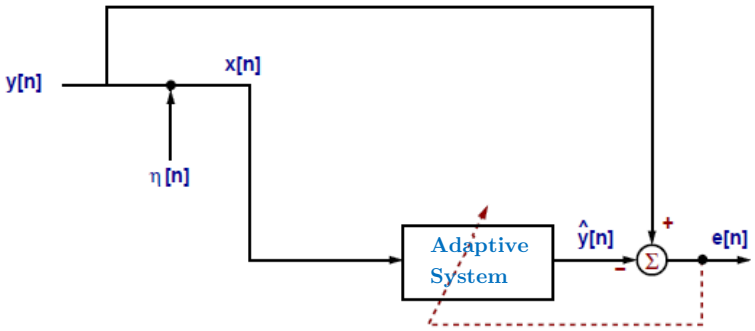


Figure 10 – Diagram of a supervised learning system

Identifying the frequencies of noise and then constructing the relevant filters is not always the best option as this process cannot be done in real. Thus another method – supervised adaptive learning – is used instead.

The idea behind this algorithm is to estimate AR coefficients which would remove noise. For this the Normalized Least Mean Square (NLMS) equations were used:

$$\hat{y}[n] = w[n]^T x[n]$$

$$e[n] = y[n] - \hat{y}[n]$$

$$w[n+1] = w[n] + \mu \times e[n] \times \frac{[x[n], \dots, x[n-p]]}{[x[n], \dots, x[n-p]]^T [x[n], \dots, x[n-p]]}$$

$w[n]$ represents the AR coefficients
$e[n]$ is the error between the estimated signal and the learning signal
$x[n]$ is the input signal
$y[n]$ is the clean signal – also referred to as the learning signal
$\hat{y}[n]$ is the estimated signal

μ is the learning rate – ranging from 0 to 1 for an NLMS filter
p is the order of the filter to filter out the noise.

Figure 10 provides a diagram of the way this adaptive filter works.

The matlab code resembled this:

```
for i = p:N
    ŷ(i) = w(i,:)*x(i:-1:i-p+1); % estimate
    e(i) = y(i) - ŷ(i); % error
    n(i) = x(i:-1:i-p+1)'*x(i:-1:i-p+1); % normalising factor
    if n(i) <= 0 % avoid division by zero!
        w(i+1,:) = w(i,:);
    else
        w(i+1,:) = w(i,:) + μ*e(i)*x(i:-1:i-p+1)'./n(i);
    end
```

```
end
```

Different orders and values of μ were explored over time. First tested was a filter of order 4 with μ=0.5 as seen in Figure 11. The coefficients can be seen to be extremely sensitive and changing very frequently however the result was encouraging due to most noise being removed.

To investigate the effects each of the two parameters was changed, keeping the other one constant, to try and determine optimal parameters. From Figure 11, Figure 12, Figure 13 and Figure 14 and extra collected data, Table 1 was made:

mu\p	p=10		p=8		p=4		p=2		p=1	
mu=1	0.0196	0.0455	0.0192	0.0471	0.0341	0.0481	0.0548	0.0366		
mu=0.5	0.0153	0.0516	0.0163	0.0539	0.0350	0.0501	0.0457	0.0374	0.3877	1.1504
mu=0.1	0.0166	0.086	0.0219	0.0852	0.0534	0.0804	0.0736	0.0806	0.8806	5.554
mu=0.01	0.086	0.1308	0.0488	0.1301	0.0725	0.1302	0.0849	0.1336	0.2398	3.4578

Table 1 – Colour coded table of relative error of channel 1 and channel 2. Green values are lower, red ones are higher. In general it was found that for a $\mu = 0.5$ increasing the p (order) led to less noise and crackling.

From this table it can be seen that having different parameters for difference channels would be preferable. The code was thus changed to reflect this discovery and the following values were used: $p_1 = 10$,$p_2 = 2$, $\mu_1 = 0.5$ and $\mu_2 = 1$. It was found, by listening and looking at the relative error, that increasing the order removed the noise and the crackling noticed at orders around 2 and 4, creating a track with very little crackling. While not computationally desirable a test was done for the values $p_1 = 20$,$p_2 = 40$, $\mu_1 = 0.5$ and $\mu_2 = 1$ and the results can be found in Figure 16. The listening test revealed that most noise and crackling was gone with only a faint intermittent tick.

For computational reasons it would be preferable not to increase the order too much thus it was felt that having $p_1 = 10$,$p_2 = 2$, $\mu_1 = 0.5$ and $\mu_2 = 1$ was the best compromise of quality to computation required.

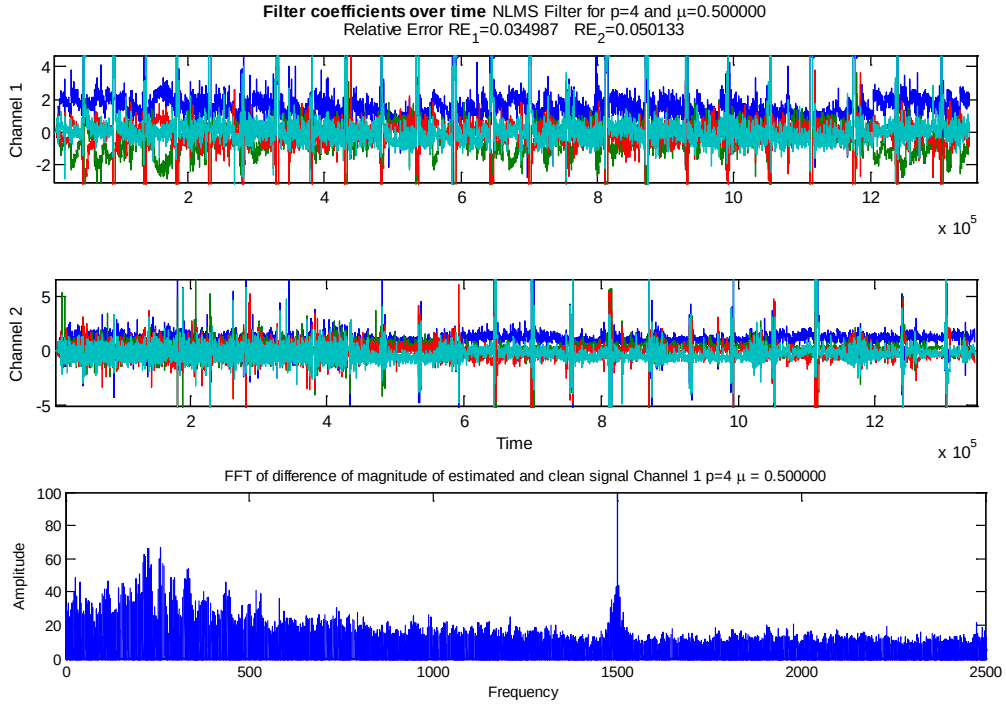


Figure 11 – Evolution of coefficients over time for an order 4 NLMS filter of $\mu = 0.5$ with the associated frequency differences. Listening test: Noise removed by crackling heard instead of noise thus perhaps a bit too aggressive.

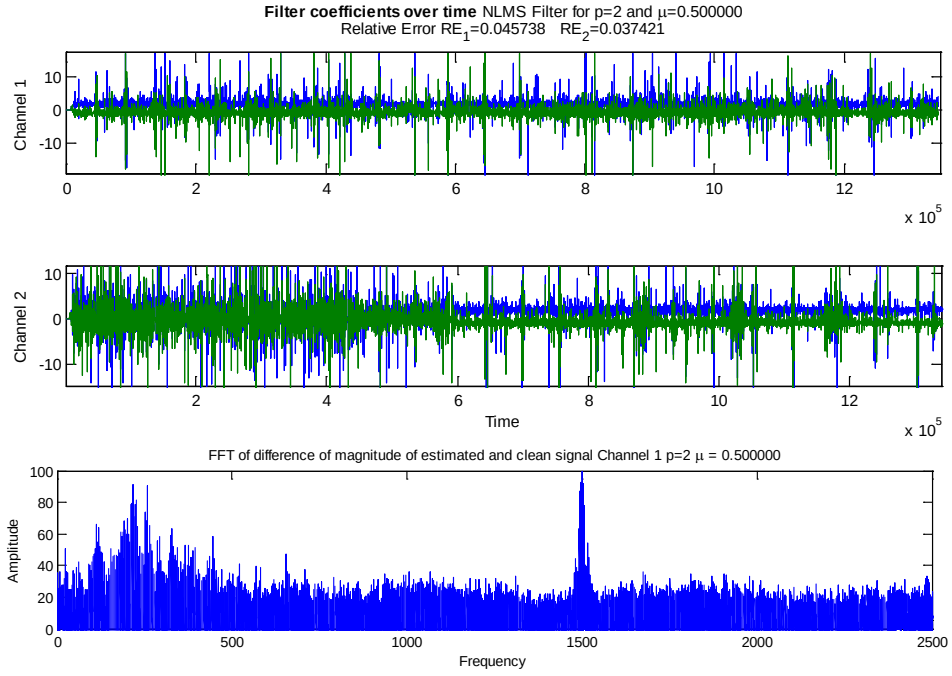


Figure 12 - Evolution of coefficients over time for an order 2 NLMS filter of $\mu = 0.5$ with the associated frequency differences. Listening test: Noise removed by crackling heard instead of noise, not too different from Figure 11 except with possibly more crackling

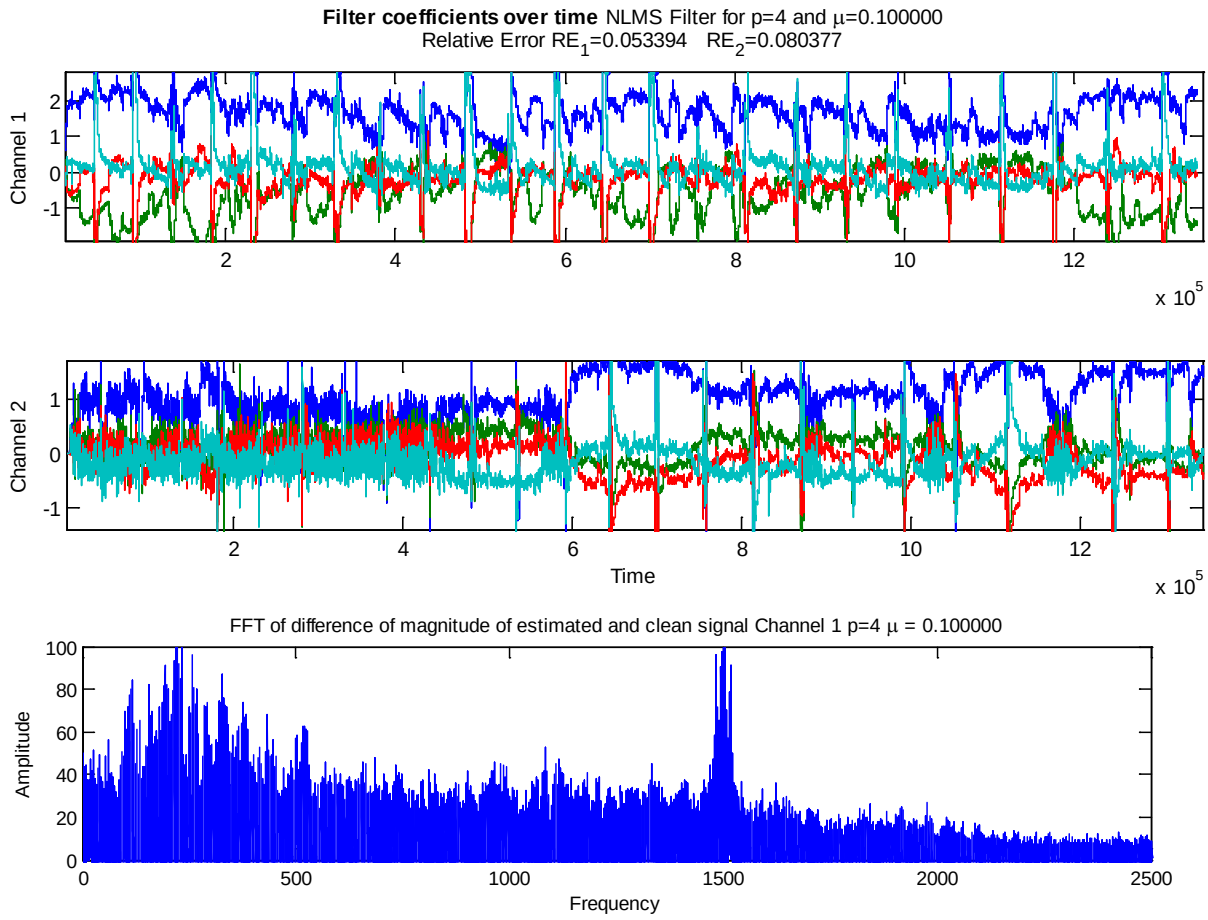


Figure 13 - Evolution of coefficients over time for an order 4 NLMS filter of $\mu = 0.1$ with the associated frequency differences. Listening test: Slight crackling mixed with a soft sound of the original noise. Possibly best listening experience so far due to good compromise between noise removal and crackling,

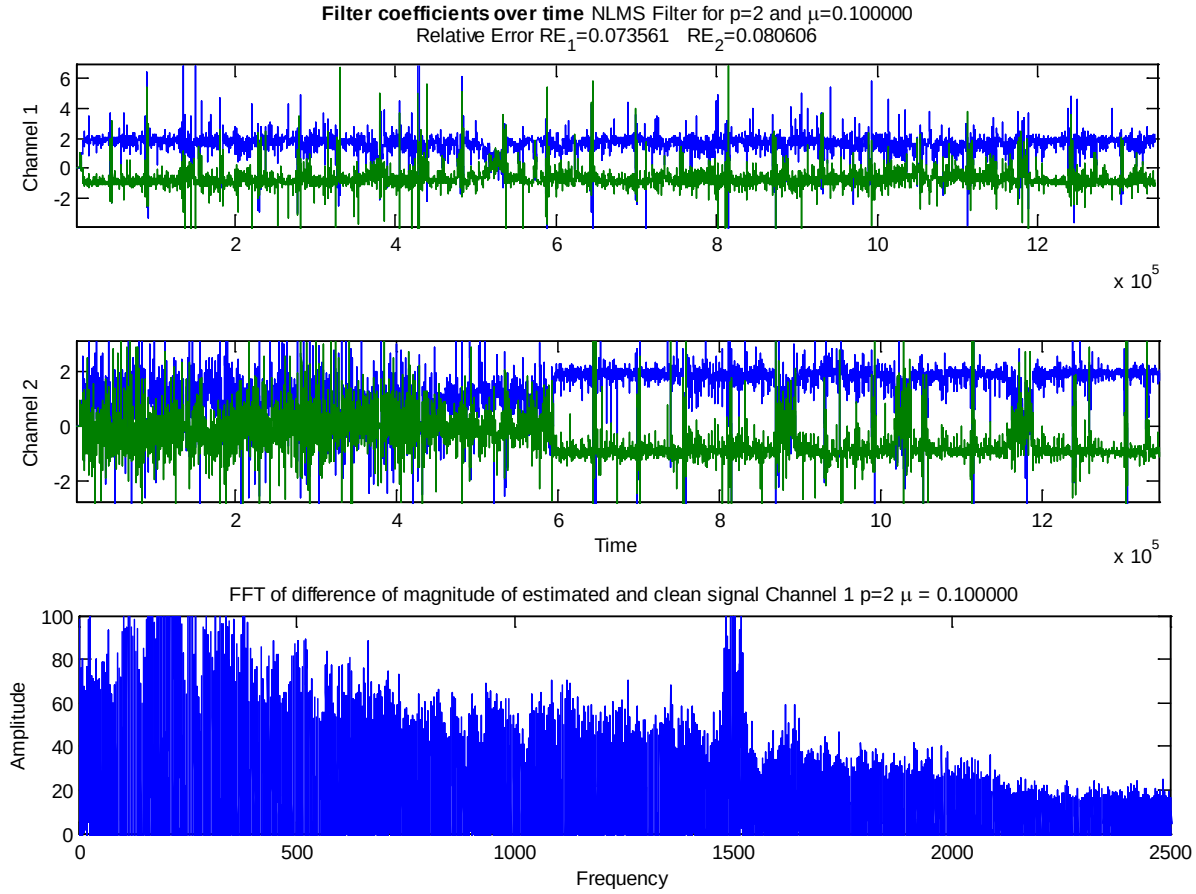
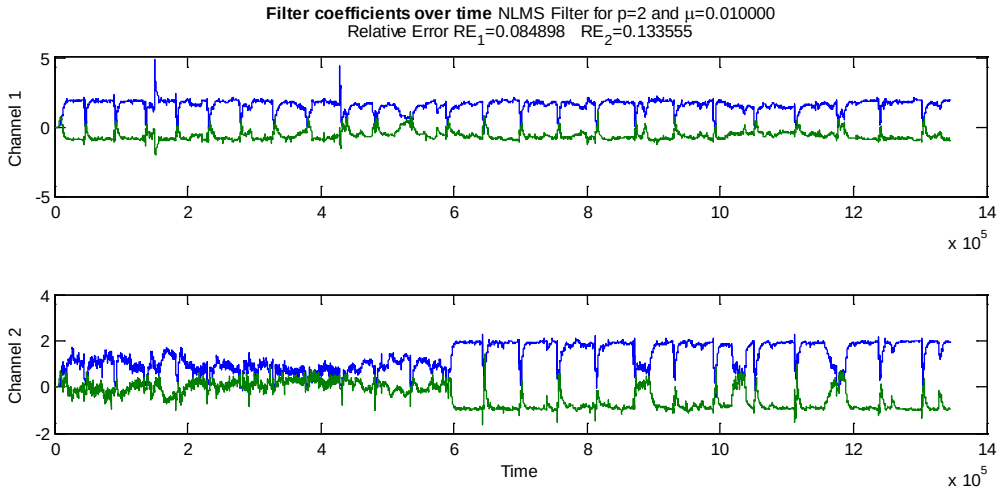


Figure 14 - Evolution of coefficients over time for an order 2 NLMS filter of $\mu = 0.1$ with the associated frequency differences. Listening test: Slight crackling mixed with a soft sound of the original noise. Very similar to the filter with same μ but with order 4.



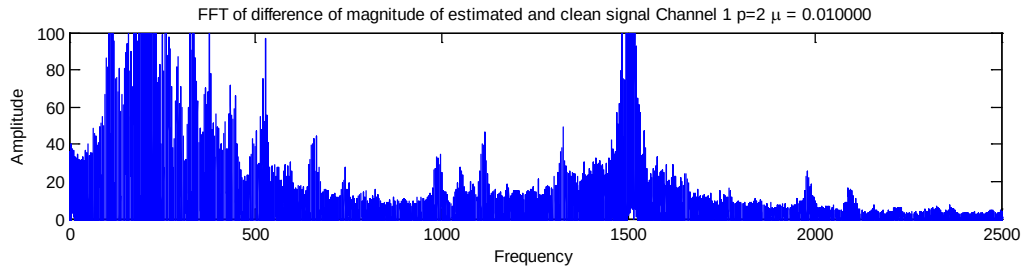


Figure 15 - Evolution of coefficients over time for an order 2 NLMS filter of $\mu = 0.01$ with the associated frequency differences. Listening test: Little to no crackling mixed with a soft sound of the original noise though noise louder than figure 14.

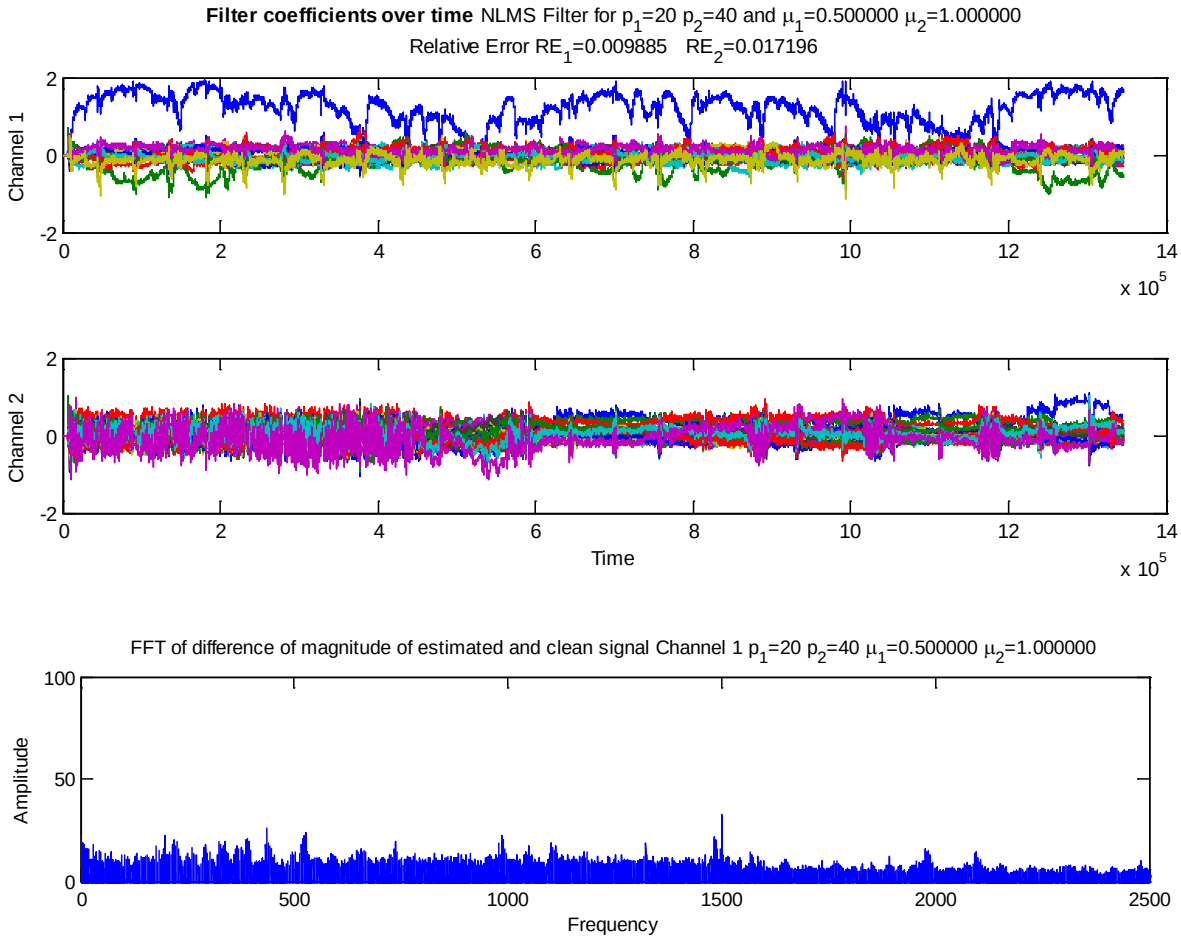


Figure 16 - Evolution of coefficients over time for an order 20 and 40 NLMS filter with the associated frequency differences. It can be seen that increasing the order has led to the FFT of frequency differences to be noticeably lower. Listening test: Very little crackling with no noise heard – only a very faint tick every so often.

4.5.2 Comparing LMS with NLMS

The adaptive filter used what is called a Normalized Least Mean Squared formula – which is designed in order to be stable. This is due to the "basic" LMS filter not being inherently stable and choosing a correct learning rate (μ).

Thus to compare the performance two instances with the same parameters were tested out and the results can be found in Figure 17 and Figure 18. Two measures allow us to see that the NLMS filter is more preferable. Firstly the relative error values of each channel are lower

(thus better) for the NLMS filter and secondly the plot of the differences in spectrum of the clean signal and the estimated signal do not show the noise frequency components for NLMS filter whereas a LMS filter does. It is reasonable to conclude that NLMS filters are better for the investigated scenario.

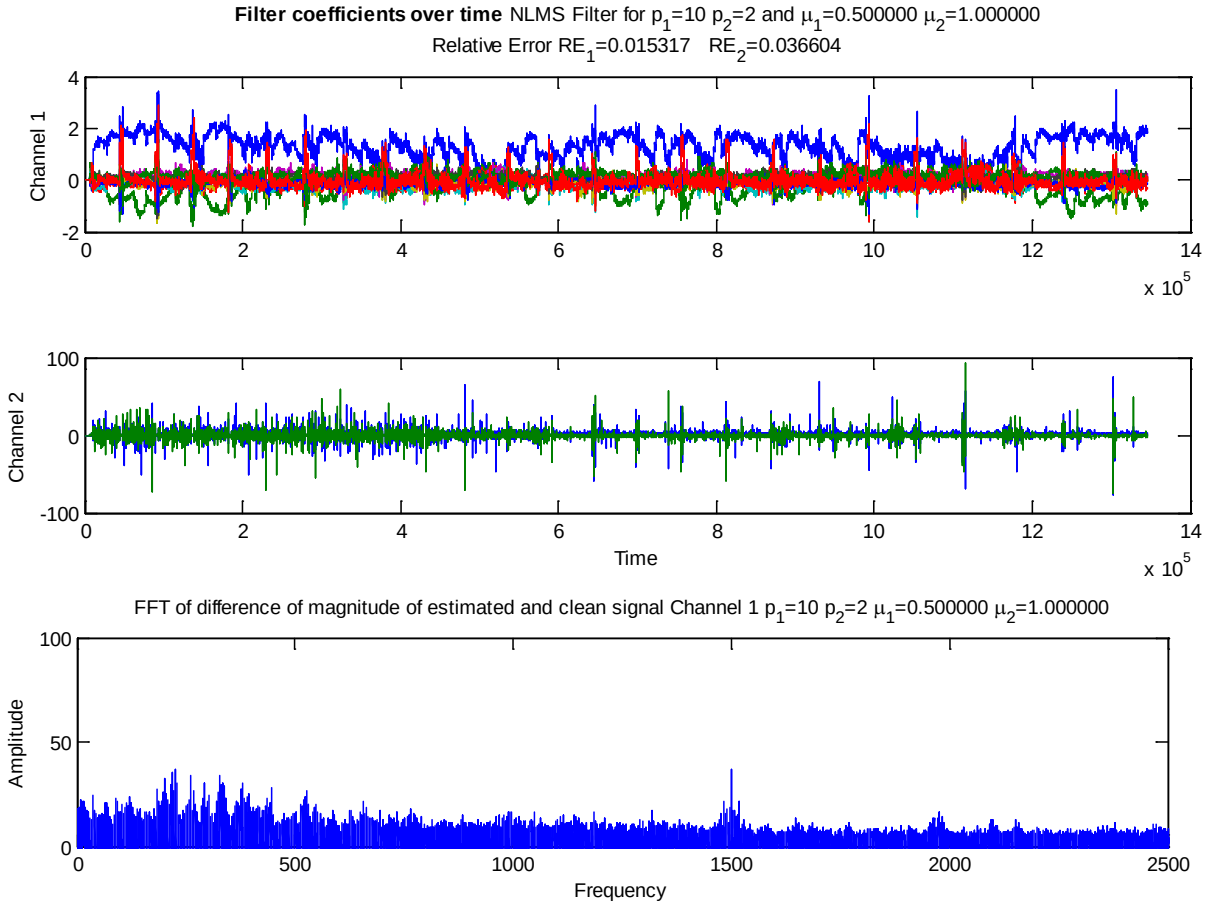


Figure 17 –Evolution of AR coefficients over time for an NLMS filter.

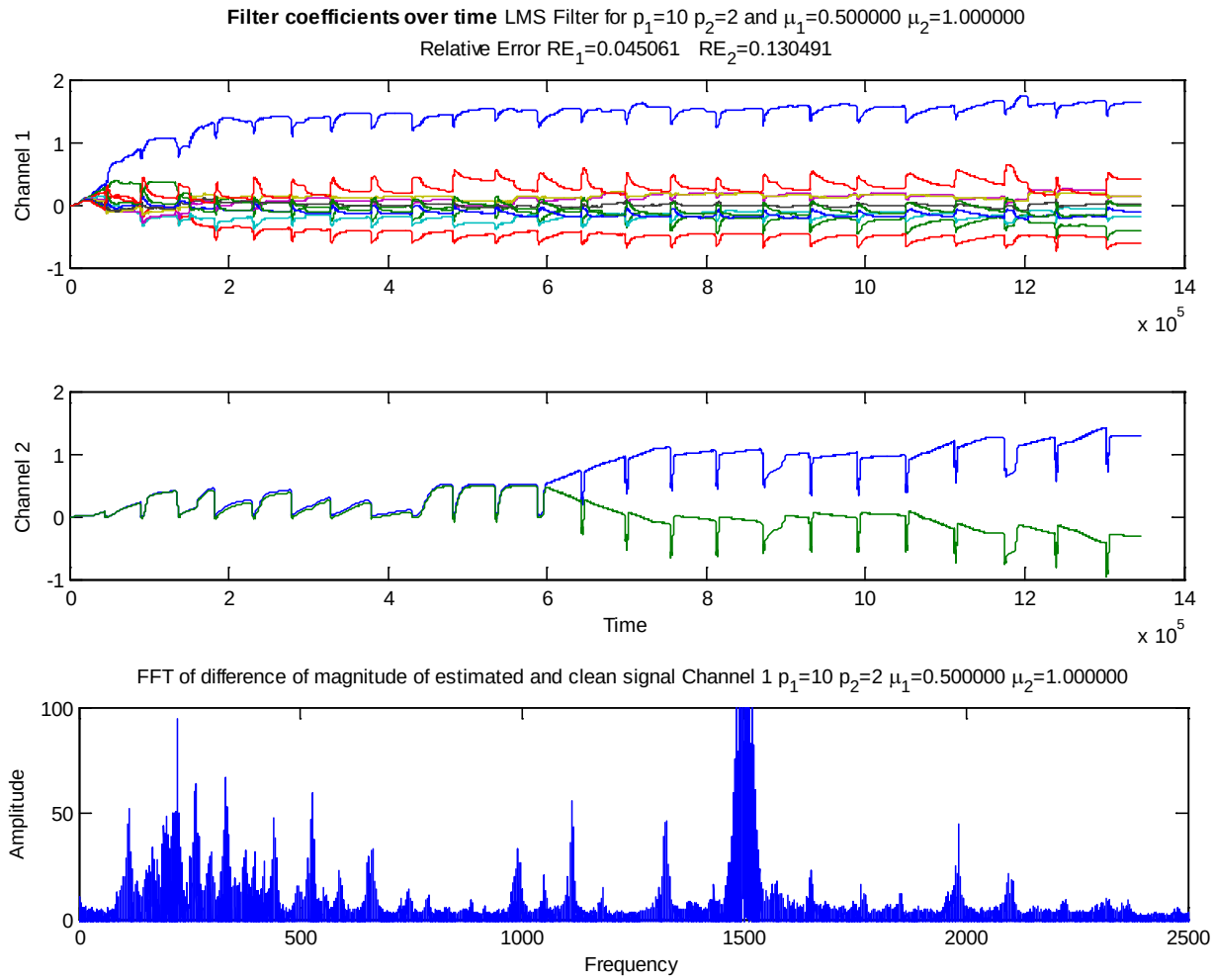


Figure 18 – Evolution of AR coefficients over time for an LMS filter.

4.6 LTE noise removal

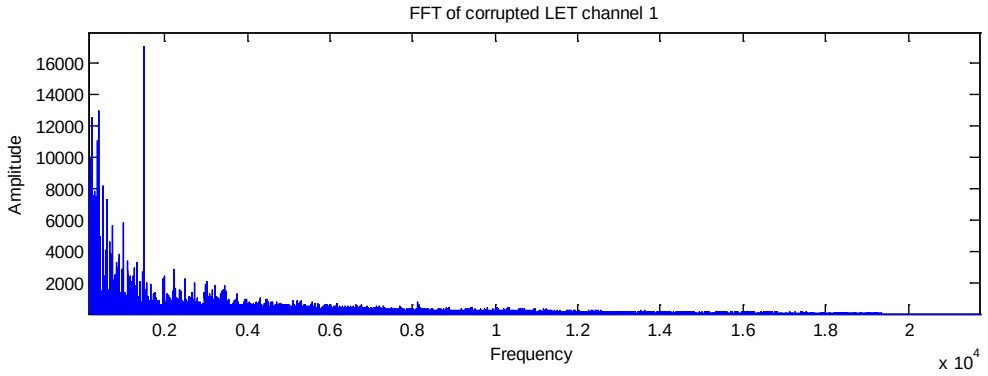


Figure 19 – FFT of the corrupted channel 1 of the LET single

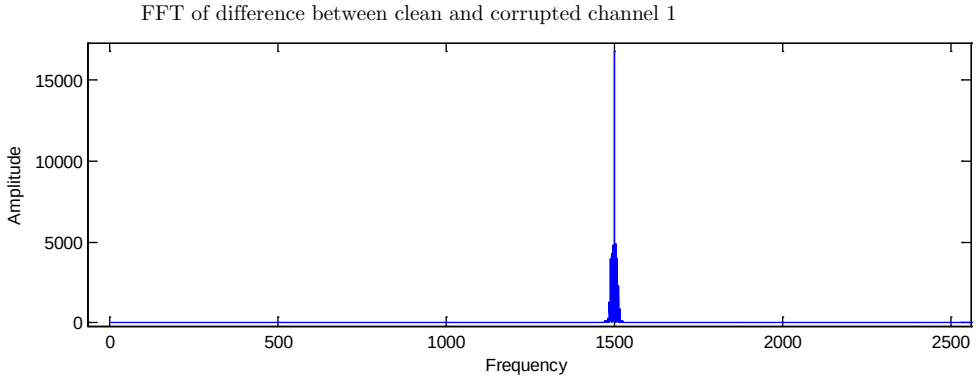


Figure 20 – By taking a difference of the FFT of the clean and corrupted signal noise for channel 1 was identified

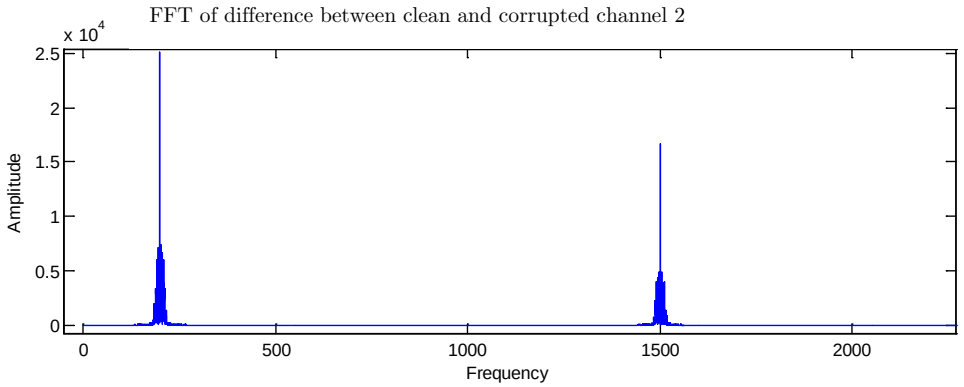


Figure 21 - By taking a difference of the FFT of the clean and corrupted signal noise for channel 2 was identified

The first observation from Figure 19, Figure 20 and Figure 21 was that the noise was exactly the same as the Stairway to Heaven track. However as the track was composed of more frequencies it was judged that a simple Butterworth filter would also remove unwanted frequencies as it operates all throughout the track. This was indeed the case after applying the following code:

```
data=ums;
[b,a]=butter(3, [1420 1560]/Fs*2, 'stop');
data = filtfilt(b,a,data); %filtfilt used instead of filter to avoid change in phase
[b,a]=butter(3, [150 250]/Fs*2, 'stop');
data(:,2) = filtfilt(b,a,data(:,2));
```

Instead a NLMS filter was used for its adaptive properties and various mu and p combinations were done.

Table 2 shows us that this time the optimal mu for both channels is 0.4 and the best tested p is 10. From this these coefficients were chosen and the noise was indeed removed – though some crackling did remain.

Figure 22 shows the evolutions of coefficients over time and justifies the need for a filter of order 10 as the coefficients can be seen to be all distinct. This is due to the LET soundtrack having a wider range of frequencies thus a higher order can lock in to the sound frequencies more accurately and avoid removing essential frequency components.

mu\p	p=10		p=8		p=4		p=2	
mu=1	0.0508	0.035	0.0602	0.0349	0.0936	0.0479	0.1287	0.0953
mu=0.5	0.0397	0.0269	0.0464	0.0266	0.0717	0.0383	0.1057	0.0785
mu=0.4	0.0383	0.0256	0.0449	0.0253	0.0696	0.0368	0.105	0.0756
mu=0.1	0.0389	0.022	0.0462	0.0224	0.0767	0.0302	0.1199	0.0624
mu=0.01	0.0581	0.029	0.0693	0.0289	0.0964	0.0318	0.1162	0.0515

Table 2 - Colour coded table of relative error of channel 1 and channel 2. Green values are lower, red ones are higher. In general it was found that for a $\mu = 0.5$ increasing the p (order) led to less noise and crackling.

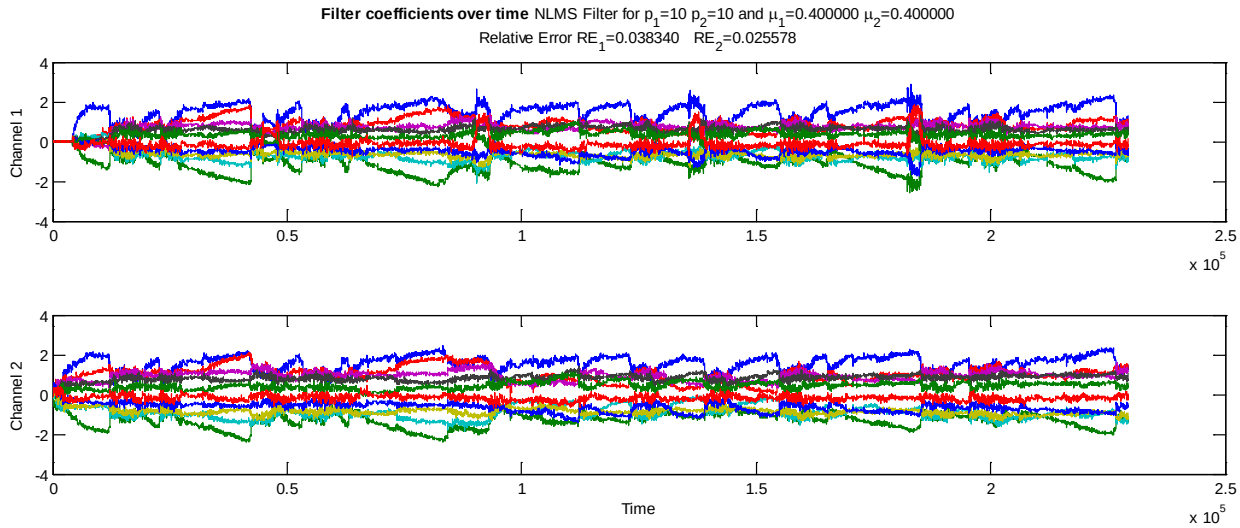


Figure 22 – Evolution of coefficients over time for both channels of the LET album.

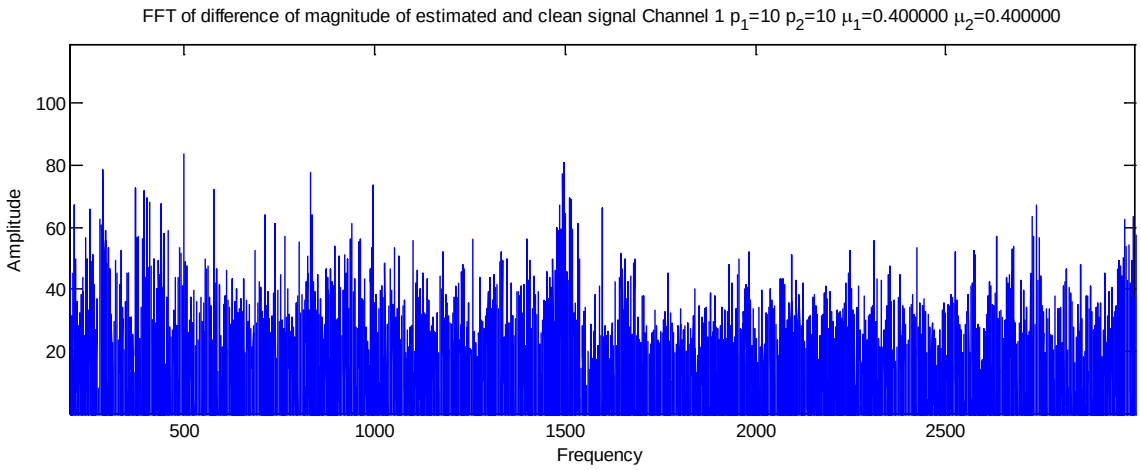


Figure 23 – Difference of the FFT of the estimated signal and the clean signal – as can be seen there still remains some noise at 1500hz but it is greatly attenuated

Appendix

Table of figures

Matlab Code

Part 4.1.1 to 4.1.4

```
clc;
%% Plot periodogram
if ~exist('s2h','var')
    load vinyl.mat
end
Fs=44100;
figure(1);
Hs=spectrum.periodogram;          % Use default values
psd(Hs,s2h_original(:,2),'Fs',Fs)
%% filter
data=s2h;
[b,a]=butter(3, [1420 1560]/Fs*2, 'stop');
data = filtfilt(b,a,data); %filtfilt used instead of filter to avoid change in phase
[b,a]=butter(3, [150 250]/Fs*2, 'stop');
data(:,2) = filtfilt(b,a,data(:,2));
%plot fft of difference
l = length(data);
plot((1:l)*Fs/l,abs(fft(data-s2h_original)))
title('Spectrum (FFT) of difference between data and orignal');
ylabel('Amplitude')
xlabel('Frequency')
axis([0 2000 0 500])
%% play
a = audioplayer(data,Fs);
% play(a);
%% clean periodogram
[periodgm1,w] = periodogram(s2h_original(:,2),[],[],Fs);
figure;
plot(w,periodgm1);
%% 4.1.4 Plot periodograms
figure(4);
% Hs=spectrum.periodogram;        % Use default values
[periodgm1,w] = periodogram(data(:,1),[],[],Fs);
[periodgm2,w] = periodogram(data(:,2),[],[],Fs);
[periodgm3,w] = periodogram(s2h(:,1),[],[],Fs);
[periodgm4,w] = periodogram(s2h(:,2),[],[],Fs);
[periodgm5,w] = periodogram(s2h_original(:,1),[],[],Fs);
[periodgm6,w] = periodogram(s2h_original(:,2),[],[],Fs);

subplot(3,1,1)
plot(w,periodgm1);hold on;
plot(w,periodgm2,'r');hold off;
title('Periodogram of data')
ylabel('Amplitude')
axis([0 2000 0 4*10^-4]);
legend('Processed data Channel 1','Processed data Channel 2');
axis([0 2000 0 4*10^-4]);

subplot(3,1,2)
plot(w,periodgm3);hold on;
```

```
plot(w,periodgm4,'g');hold off;
axis([0 2000 0 4*10^-4]);
legend('original Corrupted data Channel 1','original Corrupted data Channel 2');
axis([0 2000 0 4*10^-4]);
ylabel('Amplitude')

subplot(3,1,3)
plot(w,periodgm5);hold on;
plot(w,periodgm6,'r');hold off;
axis([0 2000 0 4*10^-4]);
legend('original clean data Channel 1','original clean data Channel 2');
axis([0 2000 0 4*10^-4]);
ylabel('Amplitude')
xlabel('Frequency')
%% 4.1.4 Plot data diff
figure(5);
% Hs=spectrum.periodogram;        % Use default values
[periodgm1,~] = periodogram(data(:,1),[],[],Fs);
[periodgm2,~] = periodogram(data(:,2),[],[],Fs);
[periodgm3,~] = periodogram(s2h_original(:,1),[],[],Fs);
[periodgm4,w] = periodogram(s2h_original(:,2),[],[],Fs);
re_1 = abs(norm(periodgm3 - periodgm1)/norm(periodgm3));
re_2 = abs(norm(periodgm4 - periodgm2)/norm(periodgm4));
fftdiff_1 = abs(fft(data(:,1)-s2h_original(:,1)));
fftdiff_2 = abs(fft(data(:,2)-s2h_original(:,2)));
w= (1:length(fftdiff_1))*Fs/length(fftdiff_1);
clf(5);
hold all;
plot(w,fftdiff_1);
plot(w,fftdiff_2);
str = sprintf('\\bf{FFT of difference of Filtered and Clean}\n\\rm{Relative Error RE_1=%f
RE_2=%f}',re_1,re_2);
title(str)
axis([0 2500 0 70]);
ylabel('Amplitude');
xlabel('Freqeuncy (Hz)')
legend('Difference in channel 1','Difference in channel 2');
```

Part 4.1.5

```
%% Adaptation from part 4.1.5 NLMS
clc;
LMS = 1;
p1=10;
p2=2;
mu1=.5;
mu2=1;
x1=s2h(:,1);
x2=s2h(:,2);
N = length(x1);
w1 = zeros(N,p1);
w2 = zeros(N,p2);
e1 = zeros(N,1);
e2 = zeros(N,1);
y_h1 = zeros(N,1);
y_h2 = zeros(N,1);
y1 = s2h_original(:,1);
y2 = s2h_original(:,2);
n1 = zeros(N,1);
n2 = zeros(N,1);
for i = p1+1:N
```

```
    y_h1(i) = w1(i,:)*x1(i-1:-1:i-p1);
    e1(i) = y1(i) - y_h1(i);
    n1(i) = x1(i-1:-1:i-p1)'*x1(i-1:-1:i-p1);
    if LMS == 1
        n1(i) = 1;
    end
    if n1(i) <= 0
        w1(i+1,:) = w1(i,:);
    else
        w1(i+1,:) = w1(i,:) + mu1*e1(i)*x1(i-1:-1:i-p1)'./n1(i);
    end
end
%% bit here
for i = p2+1:N
    y_h2(i) = w2(i,:)*x2(i-1:-1:i-p2);
    e2(i) = y2(i) - y_h2(i);
    n2(i) = x2(i-1:-1:i-p2)'*x2(i-1:-1:i-p2);
    if LMS == 1
        n2(i) = 1;
    end
    if n2(i) <= 0
        w2(i+1,:) = w2(i,:);
    else
        w2(i+1,:) = w2(i,:) + mu2*e2(i)*x2(i-1:-1:i-p2)'./n2(i);
    end
end
disp('Done! :)')
%% fig

%calculate relative error
[periodgm1,~] = periodogram(y_h1,[],[],Fs);
[periodgm2,~] = periodogram(y_h2,[],[],Fs);
[periodgm3,~] = periodogram(s2h_original(:,1),[],[],Fs);
[periodgm4,w] = periodogram(s2h_original(:,2),[],[],Fs);
re_1 = abs(norm(periodgm3 - periodgm1)/norm(periodgm3))
re_2 = abs(norm(periodgm4 - periodgm2)/norm(periodgm4))
%%%%%%%%%%%%%%%%%%%%
figure(1);
subplot(2,1,1);
plot(w1);
str = sprintf('\\bf{Filter coefficients over time} \\rm{NLMS Filter for p_1=%d p_2=%d and
\\mu_1=%f \\mu_2=%f}\nRelative Error RE_1=%f   RE_2=%f',p1,p2,mu1,mu2,re_1,re_2);
title(str);
ylabel('Channel 1')
subplot(2,1,2);
plot(w2);
ylabel('Channel 2')
xlabel('Time')
figure(2)
plot([1:length(y_h1)]*FS/length(y_h1),abs(fft(y_h1))-abs(fft(s2h_original(:,1))))
str = sprintf('FFT of difference of magnitude of estimated and clean signal Channel 1 p_1=%d p_2=%d
\\mu_1=%f \\mu_2=%f',p1,p2,mu1,mu2);
title(str)
ylabel('Amplitude')
xlabel('Frequency')
axis([0 2500 0 100])
%% 4.1.5 Plot data diff
figure(3);
Fs = 44100;
% Hs=spectrum.periodogram;       % Use default values
```

```
[periodgm1,~] = periodogram(y_h1,[],[],Fs);
[periodgm2,~] = periodogram(y_h2,[],[],Fs);
[periodgm3,~] = periodogram(s2h_original(:,1),[],[],Fs);
[periodgm4,w] = periodogram(s2h_original(:,2),[],[],Fs);
periodgm1 = abs((periodgm3-periodgm1))./abs(periodgm3);
periodgm2 = abs((periodgm4-periodgm2))./abs(periodgm4);
abs(periodgm3);
clf(3);
hold all;
plot(w,periodgm1);
str = sprintf('Periodogram of normalized difference of magnitude of estimated and clean signal p_1=%d
p_2=%d \\mu_1=%f \\mu_2=%f',p1,p2,mu1,mu2);
title(str)
ylabel('Amplitude')
plot(w,periodgm2);
legend('(P1-Ph1)/P1','(P2-Ph2)/P2');
axis tight;
ylabel('Amplitude')
xlabel('Frequency')
%% play the sample
a =audioplayer([y_h1 y_h2],FS);
play(a)
```

Part 4.1.6

```
%% Adaptation from part 4.1.5 NLMS
clc;
LMS = 0;
p1=2;
p2=p1;
mu1=.4;
mu2=mu1;
x1=um(1:end/10,1);
x2=um_original(1:end/10,2);
N = length(x1);
w1 = zeros(N,p1);
w2 = zeros(N,p2);
e1 = zeros(N,1);
e2 = zeros(N,1);
y_h1 = zeros(N,1);
y_h2 = zeros(N,1);
y1 = um_original(:,1);
y2 = um_original(:,2);
n1 = zeros(N,1);
n2 = zeros(N,1);
for i = p1+1:N
    y_h1(i) = w1(i,:)*x1(i-1:-1:i-p1);
    e1(i) = y1(i) - y_h1(i);
    n1(i) = x1(i-1:-1:i-p1)'*x1(i-1:-1:i-p1);
    if LMS == 1
        n1(i) = 1;
    end
    if n1(i) <= 0
        w1(i+1,:) = w1(i,:);
    else
        w1(i+1,:) = w1(i,:) + mu1*e1(i)*x1(i-1:-1:i-p1)'./n1(i);
    end
end
%% bit here
for i = p2+1:N
    y_h2(i) = w2(i,:)*x2(i-1:-1:i-p2);
```

```
    e2(i) = y2(i) - y_h2(i);
    n2(i) = x2(i-1:-1:i-p2)'*x2(i-1:-1:i-p2);
    if LMS == 1
        n2(i) = 1;
    end
    if n2(i) <= 0
        w2(i+1,:) = w2(i,:);
    else
        w2(i+1,:) = w2(i,:) + mu2*e2(i)*x2(i-1:-1:i-p2)'./n2(i);
    end
end
disp('Done! :)')
%% fig
Fs = 44100;
%calculate relative error
[periodgm1,~] = periodogram(y_h1,[],[],Fs);
[periodgm2,~] = periodogram(y_h2,[],[],Fs);
[periodgm3,~] = periodogram(um_original(1:N,1),[],[],Fs);
[periodgm4,w] = periodogram(um_original(1:N,2),[],[],Fs);
re_1 = abs(norm(periodgm3 - periodgm1)/norm(periodgm3))
re_2 = abs(norm(periodgm4 - periodgm2)/norm(periodgm4))
%%%%%%%%%%%%%%%%%%%%
figure(1);
subplot(2,1,1);
plot(w1);
str = sprintf('\\bf{Filter coefficients over time} \\rm{NLMS Filter for p_1=%d p_2=%d and
\\mu_1=%f \\mu_2=%f}\nRelative Error RE_1=%f   RE_2=%f',p1,p2,mu1,mu2,re_1,re_2);
title(str);
ylabel('Channel 1')
subplot(2,1,2);
plot(w2);
ylabel('Channel 2')
xlabel('Time')
figure(2)
plot([1:length(y_h1)]*FS/length(y_h1),abs(fft(y_h1))-abs(fft(um_original(1:N,1))))
str = sprintf('FFT of difference of magnitude of estimated and clean signal Channel 1 p_1=%d p_2=%d
\\mu_1=%f \\mu_2=%f',p1,p2,mu1,mu2);
title(str)
ylabel('Amplitude')
xlabel('Frequency')
axis tight;
%% 4.1.5 Plot data diff
figure(3);
Fs = 44100;
% Hs=spectrum.periodogram;       % Use default values
[periodgm1,~] = periodogram(y_h1,[],[],Fs);
[periodgm2,~] = periodogram(y_h2,[],[],Fs);
[periodgm3,~] = periodogram(um_original(1:N,1),[],[],Fs);
[periodgm4,w] = periodogram(um_original(1:N,2),[],[],Fs);
periodgm1 = abs((periodgm3-periodgm1))./abs(periodgm3);
periodgm2 = abs((periodgm4-periodgm2))./abs(periodgm4);
abs(periodgm3);
clf(3);
hold all;
plot(w,periodgm1);
str = sprintf('Periodogram of normalized difference of magnitude of estimated and clean signal p_1=%d
p_2=%d \\mu_1=%f \\mu_2=%f',p1,p2,mu1,mu2);
title(str)
ylabel('Amplitude')
plot(w,periodgm2);
```

```
legend('(P1-Ph1)/P1','(P2-Ph2)/P2');
axis tight;
ylabel('Amplitude')
xlabel('Frequency')
%% play the sample
a =audioplayer([y_h1 y_h2],FS);
play(a)
```

Part 5 –Optimal filtering

Contents

5 Optimal filtering - fixed and adaptive

5.1 Wiener filter

In this section a 1000 randomly distributed Gaussian sample sequence (known as x) was filtered by the coefficients $= [1, 2, 3, 2, 1]\, a = [1]$, giving rise to the sequencey. Another 1000 sample sequence (named n) was generated but with a standard deviation of 0.1 . From this the SNR for $z = y + n$ was calculated and found to be 33.17 dB.

5.1.1 Cross and autocorrelation

Values of the vectors R_{xx} and p_{zx} were calculated using the following matlab script:

```
%% Part 5.1.1
r_xx_c = xcorr(x,'unbiased');
r_xx=zeros(N_w,N_w);
for i=1:N_w
    r_xx(:,i) = r_xx_c(N-i+1:N+N_w-i);
end
plot(r_xx)
p_zx = xcorr(z,x,'unbiased');
p_zx = p_zx(N:N+N_w-1);
```

5.1.2 Optimal Wiener coefficients

The optimal values of the Wiener coefficients was found by the standard formula $\boldsymbol{w_{opt} = R_{xx}^{-1} P_{zx}}$:

```
%% Part 5.1.2
w_opt = r_xx\p_zx;
```

The found values were $w_{opt} = [0.999, 2.001, 3.003, 2.006, 1.002]$ which compared to the ideal value of $b = [1, 2, 3, 2, 1]$ are similar. If it was known that the coefficients to the system were integers the correct coefficients could be determined by:

```
w_opt = round(w_opt)
```

The difference can be attributed to the noise but also due to simple inaccuracies by "only" having 1000 samples to act upon. It was found that increasing the sample size and reducing the noise the w_{opt} values can be better estimated.

5.1.3 Optimal Wiener coefficients

It was noticed that different noise deviation changed the performance of the Wiener filter. Table 1 shows the effect of SNR of coefficient estimation. It can be seen that, as expected, increasing the noise power (related to sigma) led to less good estimates.

SNR (dB)	Sigma	$w_{opt}[1]$	$w_{opt}[2]$	$w_{opt}[3]$	$w_{opt}[4]$	$w_{opt}[5]$
32.53	0.1	1.00064	2.00046	3.00341	2.00377	1.01097
6.34	2.08	1.06795	1.96392	3.00366	2.08057	1.08715
0.23	4.06	0.90325	2.12412	3.26438	2.09174	0.91824
-2.98	6.04	1.12672	2.26215	3.13486	2.02013	1.10089
-5.41	8.02	1.08948	1.82705	3.04220	2.47822	1.23995
-7.54	10	0.79500	1.79999	2.66814	1.72159	0.48651

Table 1 -Table of Wiener filter predicted coefficients at different SNR values for N=1000.

However the signal and the noise are statistically independent – this is meant in the sense that x and n are independent. Thus this would allow us to look at the autocorrelation of z and x:

$R_{zx}[k] = E\{z[n]x[n+k]\}$ which can be re-expressed as $E\{n[n]x[n+k]\} + E\{y[n]x[n+k]\}$

We can expand on this as the noise is Gaussian with a mean of 0 to get the result that $E\{n[n]x[n+k]\} = E\{n[n]\}E\{x[n+k]\} = 0 \times E\{x[n+k]\} = 0$ (this is only true if x and n are uncorrelated).

Thus we are left with $R_{zx}[k] = E\{y[n]x[n+k]\}$ which is useful due to it implying that the result is only dependant on the cross correlation of x and y.

The results above were obtained for a (relatively) small value of N thus increasing the value of N should lead to better coefficients regardless of the noise. Table 2 confirms this.

SNR (dB)	Sigma	$w_{opt}[1]$	$w_{opt}[2]$	$w_{opt}[3]$	$w_{opt}[4]$	$w_{opt}[5]$
32.76	0.1	0.9998	2.0002	2.9998	1.9999	1.0001
6.41	2.08	0.9898	2.0158	2.9930	2.0098	0.9979
0.62	4.06	1.0203	1.9705	2.9925	1.9934	0.9878
-2.84	6.04	0.9967	1.9886	3.0026	2.0322	1.0062
-5.29	8.02	1.0071	2.0141	3.0238	2.0575	1.0256
-7.22	10	1.0282	1.9888	3.0077	1.9643	0.9567

Table 2 - Table of Wiener filter predicted coefficients at different SNR values for N=1000000.

The effect of increasing N_w was investigated and it was found that all later coefficients were effectively 0. Indeed for $N_w = 10$ with a SNR of 32 the following was found $w_{opt} = [0.999, 1.999, 3.005, 2.002, 0.997, 0.000, -0.006, -0.002, -0.003, 0.0049]$.

Again this is easily explained due to all the remaining weightings being effectively 0. Using our previous result of $R_{zx}[k] = E\{y[n]x[n+k]\}$ and $y[n] = x[n] + 2x[n-1] + 3x[n-2] + 2x[n-3] + x[n-4]$ we get $R_{zx}[k] = E\{x[n+k](x[n] + 2x[n-1] + 3x[n-2] + 2x[n-3] + x[n-4])\}$ which shows how all values "earlier" than $x[n-4]$ are completely independent

(due to the way the way that the autocorrelation of $x[n]$ will be at its maximum when it intersects with itself with no time difference, the remaining ACF values being close to zero).

Again increasing the sample size led to better zero estimates.

5.1.4 Computational Complexity of the Wiener solution

Computing the computational complexity of the Wiener solution we need to find the complexity of the following functions:

- Calculation of p_{zx} and R_{xx} i.e. the cost of $N_w + 1$ correlations.
- The computational complexity of $w_{opt} = R_{xx}^{-1} P_{zx}$

We $N_w + 1$ computations of $O(N)$ (complexity for a single discrete correlation) for p_{zx} and we need $N_w + 1$ of $O(N)$ for R_{xx} (while it is a square matrix we use the symmetry and shifting properties to form R_{xx}). Thus in total we have $O(NN_w)$ computations.

Solving $w_{opt} = R_{xx}^{-1} P_{zx}$ takes $O(N_w^3)$ for the matrix inversion and multiplying a vector by a matrix takes $O(N_w^2)$ operations.

Thus in total $O(NN_w) + O(N_w^3) + O(N_w^2)$ operations are needed, approximating this we get an order of $O(NN_w) + O(N_w^3)$ (while we could ignore $O(NN_w)$ it would be bad practice as it is not dependant on N_w).

5.2 The least mean square (LMS) algorithm

5.2.1 LMS Function in matlab

The Wiener filter cannot operate correctly in non-stationary environments due to assuming stationary and can be unsuitable for many real world applications. Instead we have the LMS filer which works in non-stationary environments by adjusting each iteration towards an estimate it calculates.

```
function [w e] = lms_cust(x,z,mu,N_w)
%lms function
%takes in the noise, the filtered system and the learning constant in addition to the order of the filter
%returns the estimated coefficients over time as well as the errror
%initilize all variables
N=length(x);
w=zeros(N,N_w);
y_h = zeros(N-N_w,1);
%perform lms algorithm
for i=N_w:N
y_h(i)=w(i,:)*x((i-N_w+1):i);
e(i) = z(i) - y_h(i);
w(i+1,:) = w(i,:) +mu*e(i)*x((i-N_w+1):i)';
end
w = w(2:N+1,:);
end
```

5.2.2 Testing the LMS Function

Here the impact on changing the learning constant μ is investigated for estimating the same unknown system. Figure 1, Figure 2 and Figure 3 were plotted for 3 different values of μ and show that choosing this value is decision guided by trade-offs. This is due to a larger learning constant leading to a faster correct estimation of the unknown system but with oscillations around the estimated values. Lower values reach a reasonable system estimate slower but are more stable around the system's estimated values. Too small values would take ages to converge and too big ones could lead to instability – for the tested system from 5.1 values of μ larger than 0.4 led to instability.

Thus for all (stable) tested values of μ it was found that they would converge to the Wiener solution.

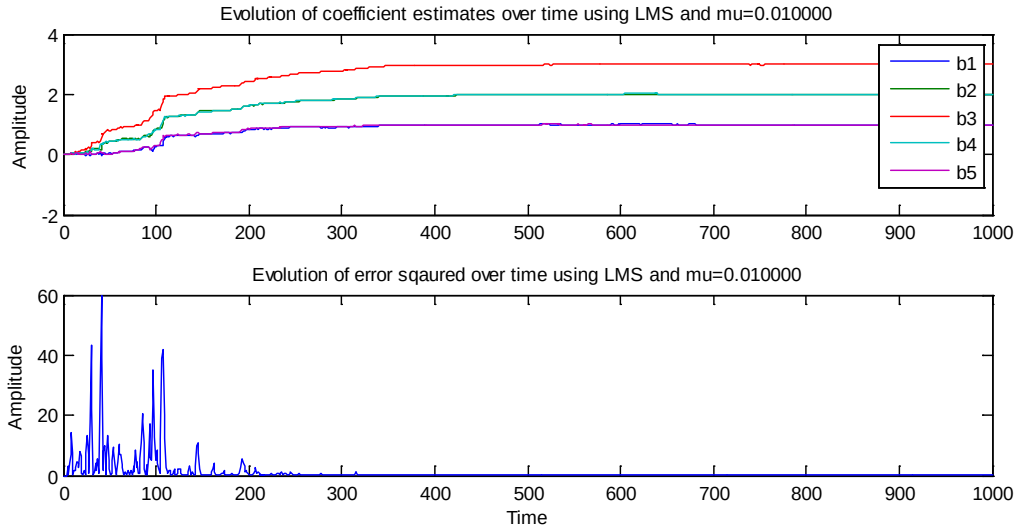


Figure 1 – Plot of adaptation of coefficients over time using the LMS function with its squared error function over time for $\mu = 0.01$. In this example the Wiener solution is reached at around 400 samples in time.

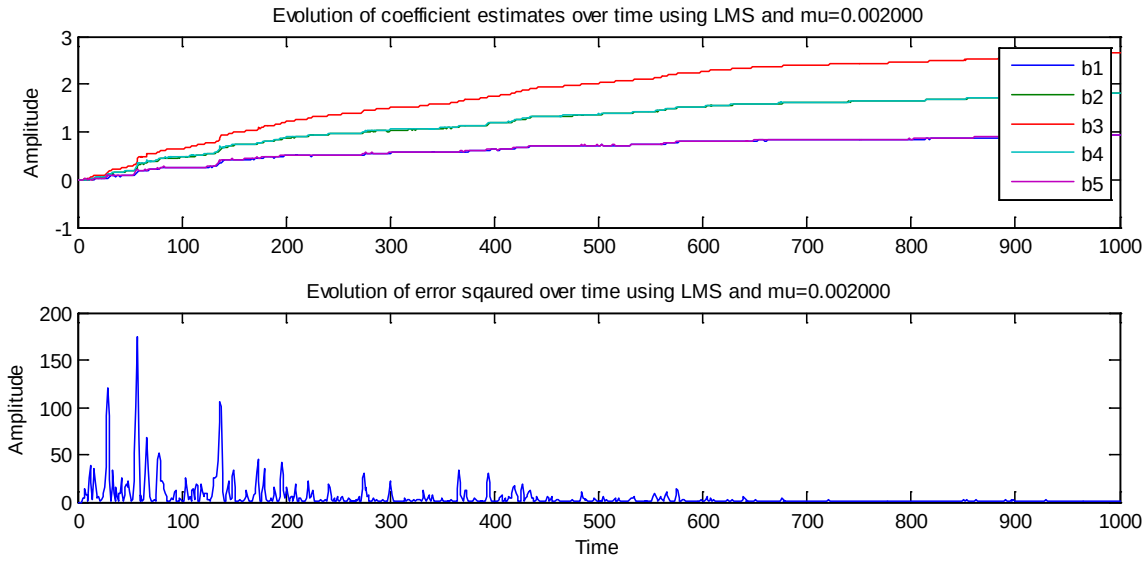


Figure 2 - Plot of adaptation of coefficients over time using the LMS function with its squared error function over time for $\mu = 0.002$. This time the Wiener solution is reached at a much slower pace that it was reached for Figure 1 (the error decays over time slower).

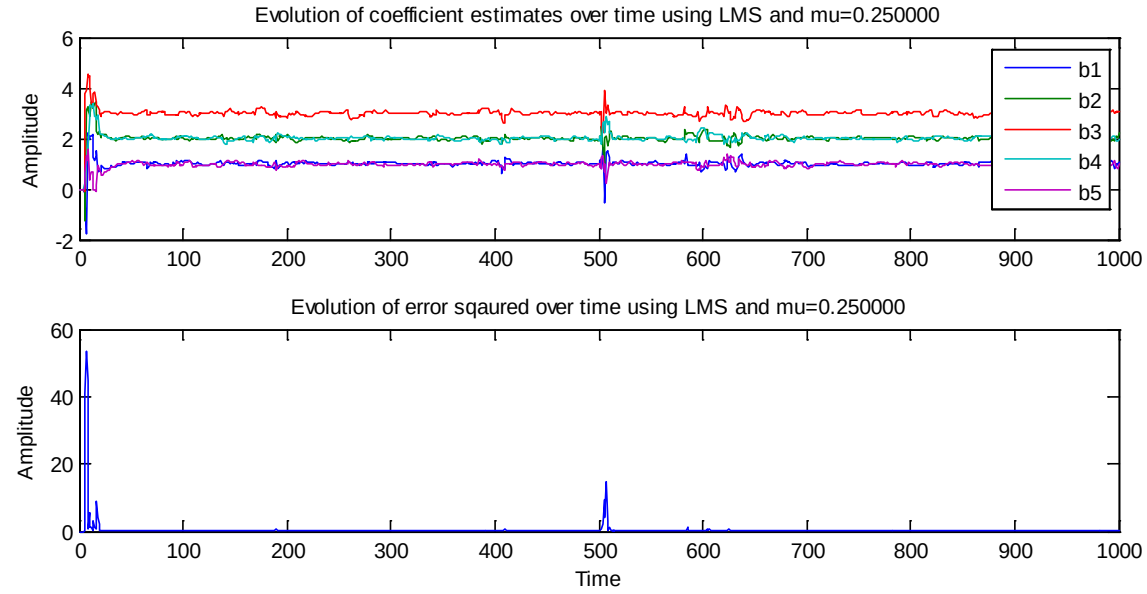


Figure 3 - Plot of adaptation of coefficients over time using the LMS function with its squared error function over time for $\mu = 0.25$. This is an example of overshoot where the Wiener solution values are reached extremely fast but many oscillations occur around the solution.

5.2.3 Computational Complexity of LMS

To calculate the computational complexity of LMS we estimate the following:

- $\hat{y}[n] = \boldsymbol{w}^T[n]\boldsymbol{x}[n]$ takes $O(2N_w)$ operations (from inspection)
- $e[n] = z[n] - \hat{y}[n]$ takes 1 operation
- $\boldsymbol{w}[n+1] = \boldsymbol{w}[n] + \mu \times e[n]\boldsymbol{x}[n]$ takes N_w additions $1+N_w$ multiplication($\mu \times e[n]\boldsymbol{x}[n]$)

Thus in total we get $O(4N_w + 1) \rightarrow O(N_w)$ operations. Considering N_w is often a relatively small number this algorithm is suited for a real-time application.

5.3 Gear shifting

In stationary environments a technique known as gear shifting can be used to reduce adaptation time by dynamically adjusting the learning constant. It was noticed that between 0.005 and 0.2 the LMS algorithm was stable and thus functions which varied between those two equations were made using empirical observations. For consistent comparison the function "rng default" was used in matlab to reset the random number generator.

- $\mu = min(max(0.055 * e(i) - 0.0733, 0.01), 0.2)$ is a simple linear fit between these two values and its performance can be found in Figure 4.
- $\mu = min(0.006e^{e(i)*1.1513} - 0.001, 0.2)$ was made from the observation that a linear fit did not adapt fast enough for high errors and could also have values of μ not as low as hoped to make it completely stable. Its performance was found to be better than the linear fit as once at the Wiener solution would stay very stable. Its performance can be seen in Figure 5.

Comparing Figure 1 with Figure 4 and Figure 5 allows us to see that gear shifting is indeed effective.

Initial μ value was 0.02

.

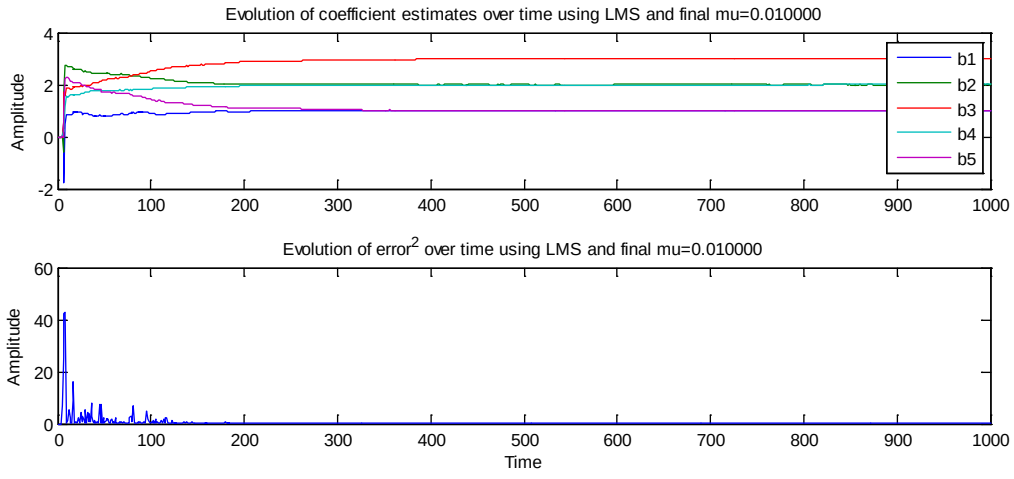


Figure 4 – LMS filter with μ=min(max(0.055*e(i)-0.0733,0.01),0.2) gear shifting. Values converge relatively quickly with the error going down reasonably fast.

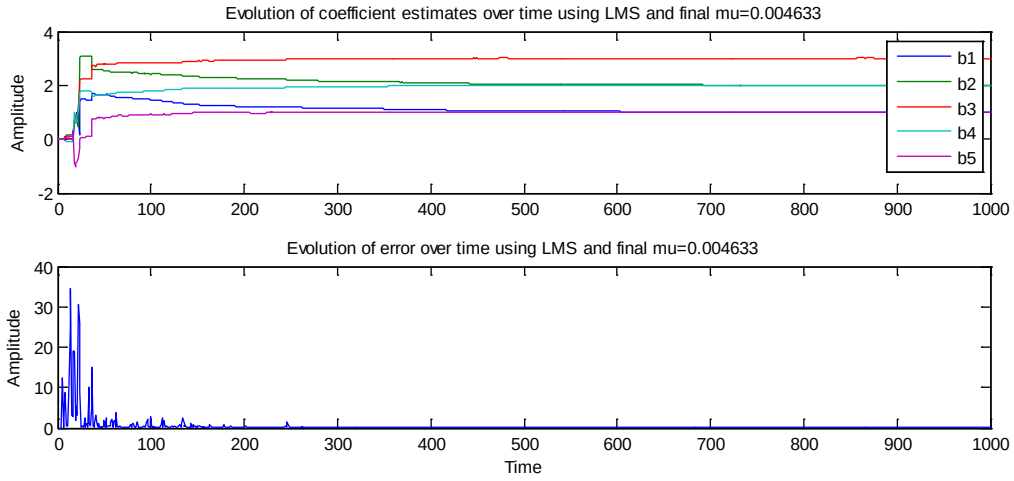


Figure 5 – LMS filter with μ=min(0.006*exp(e(i)*1.1513)-0.001,0.2) gear shifting. Values converge faster than the previous implementation with the error also going down faster – however it is larger at the start. The main advantage of this algorithm is the stability once converged to the Wiener solution.

5.4 Identification of AR processes

In this section an input was passed through an AR process and using a LMS algorithm the coefficients of the AR process were identified. Figure 6 shows the diagram of this process where a predictor compares the output it produces to the measured values to see if the estimate is correct. In this case we are modelling an AR2 process which has equation:

$$x[n] = a_1 x[n-1] + a_2 x[n-2] + \eta[n]$$

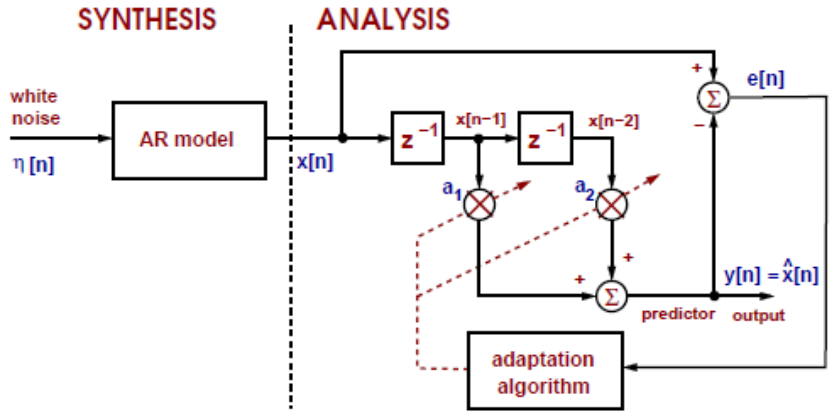


Figure 6 – Diagram of the structure of a LMS AR estimator

5.4.1 Matlab Code

The following code was made to run the AR estimation process:

```
clc; clear;
%initialise all variables
p=2;
N = 1000;
n = randn(N,1);
```

```
a = [1, 0.9, 0.2];
x= filter(1,a,n);
a = zeros(N,p);
% can also be made a function
%function [y_e, e, a] = LMS_AR(x, mu, p)
mu=0.01;
e = zeros(N,1);
%perform AR recognition/estimation
y_h = zeros(N,1);
for i = p+1:N
    y_h(i) = a(i,:)*x(i-1:-1:i-p); %a1(i)*x(i-1)+a2(i)*x(i-2);
    e(i) = x(i) - y_h(i);
    a(i+1,:) = a(i,:) + mu*e(i)*x(i-1:-1:i-p)';
end
%end %use for function, comment the rest out
%% plot everything
figure(1);
plot(e.*e);
figure(2);
clear figure(2);
plot(a);
str = sprintf('final a1:%f and final a2:%f estimation \n\\mu = %f',a(N,1),a(N,2),mu);
title(str);
ylabel('Amplitude')
xlabel('Time')
axis([0 N -1 .3])
```

The AR estimation process worked as shown in Figure 7 and estimated values around -.2 and -.9 . This is due to the difference equation of a filter being:

$$y(n) = \sum_{i=0}^{M} b_i x(n-i) - \sum_{j=0}^{N} a_j x(n-j)$$

Thus using a = [1, 0.9, 0.2] we would get

$$y(n) = -0.9 \cdot x(n-1) - 0.2 \cdot x(n-2)$$

Explaining the negative coefficients.

5.4.2 Test of AR estimator

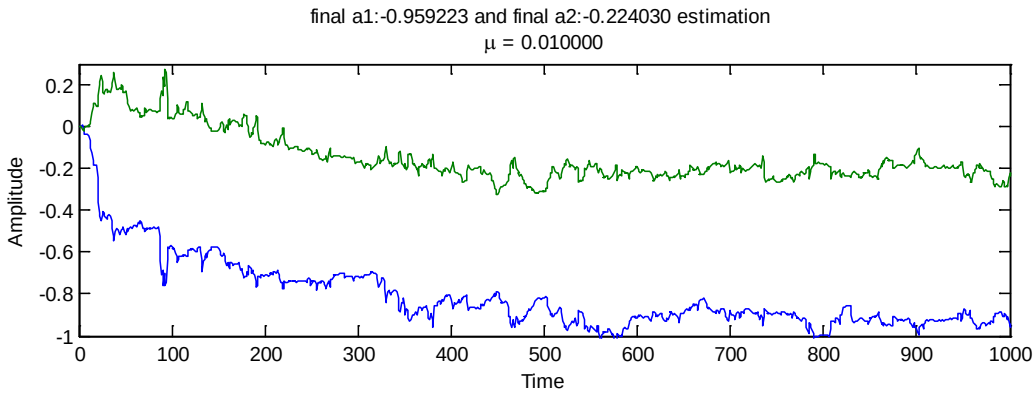


Figure 7 – LMS estimation of AR = [1, 0.9, 0.2] for N=1000 samples.

Figure 7 shows how the convergence towards the AR coefficients is slow and very "jagged" despite using a small learning constant. Using an even smaller learning constant does lead to the values to converge in a smoother way however a longer amount of time is needed to do so, as shown in Figure 8. If a process is being sampled at a high enough sampling rate this low learning constant is still reasonable for practical use.

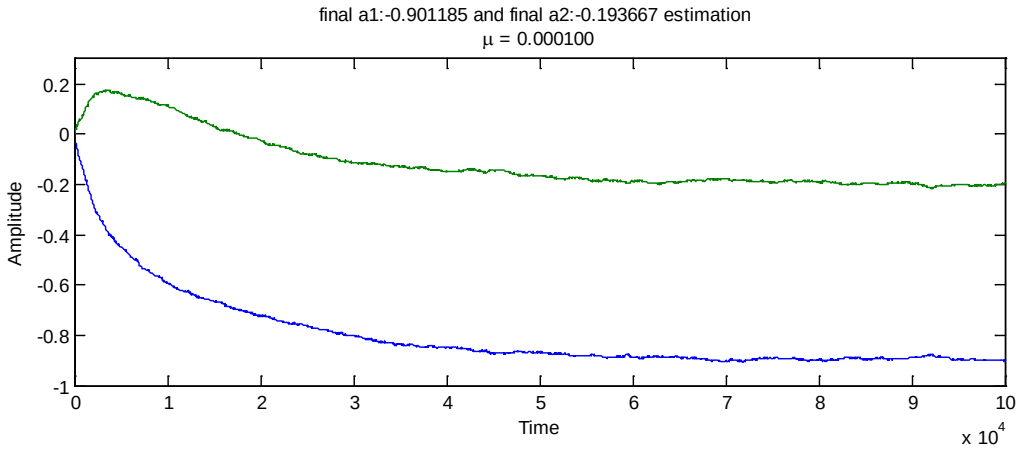


Figure 8– LMS estimation of AR = [1, 0.9, 0.2] for N=100000 samples for a learning constant of 0.0001

5.5 Speech recognition

5.5.1 Test of AR estimator

Instead of noise passed through an AR process the AR estimator built previously was used on recorded speech samples. The idea is that each speech components are distinctive and can be modelled as AR processes.

Letters of speech were sampled at 44100 Hz and 1000 samples were the sound was judged to be the most stationary (in order to allow the coefficients to settle to certain value).

The first consideration was to choose the best order. Thus for each letter the minimum error index was found as can be seen in Table 3.

suggested order (p)	a	e	s	t	x
MDL	13	28	9	27	9
AIC	33	29	33	29	38

Table 3 – Order suggested by the Minimum Description Length (MDL) and the Akaike Information Criterion (AIC) criteria – these should only be used as guidance

Suggested model orders were generally high and did not necessarily need to be so for the purpose of speech recognition. Additionally it can be seen from Figure 9 that past an order of around 4 (for the letter 'a' at least) little differentiating value can be extracted from the rest of the AR coefficients due to being very close to 0. Thus order values were chosen between 4 and 10 instead. The choice of the learning constant was also required and various

combinations were tested. From Table 4 it can be seen that a higher value of p, if not combined with an appropriate value of μ, is not always desirable as equally good values can be found for low order filters with appropriate μ's.

mu\p	**p=4**	**p=6**	**p=8**	**p=10**
$\boldsymbol{\mu = 0.1}$	5.093	5.755	5.674	5.686
$\boldsymbol{\mu = 0.5}$	9.839	9.877	9.770	9.998
$\boldsymbol{\mu = 1}$	11.980	11.636	11.783	11.969
$\boldsymbol{\mu = 2}$	13.740	12.128	4.065	-20.489

Table 4 – Table of various combinations of p and μ values for the letter 's' at 44.1kHz sampling with the prediction gain ($R_p = 10 \cdot log_{10}\left(\frac{\sigma_x^2}{\sigma_e^2}\right)$) values colour coded. Green is for higher –more desirable – values and red is for lower values. The $R_p = -20.5$ at $\mu = 2$ and p=10 is due to the learning constant causing an unstable LMS system.

Gear shifting was considered, however as speech – even in the best chosen regions – is not stationary, implementing it would not lead to sufficiently good results and it would not be practical in real world application. A quick test using the gear shifting implemented in 5.3 was tried however the error was to unstable for gear shifting to create an appreciable difference.

Some letters such as t , e and s would be better suited for an order of 6 or 8 as seen from Figure 10 and Figure 11 as the estimated coefficients are still distinguishable from 0 whereas for 'a' and 'x' the number of coefficients which are easily distinguishable from 0 is around 3 or 4. Thus as we cannot predict the letter which will be spoken a combination of p=8 and μ = 1 to create a 'best compromise' solution.

From these parameters it should be possible to identify different letters without having to listen to them. However some letters such as 't' and 'e' have the same stationary frequency components and thus further signal analysis would be required to identify letters with the same stationary properties. For example the first step would be to look at different locations in time of the 't' and 'e' were the coefficients would converge to different values.

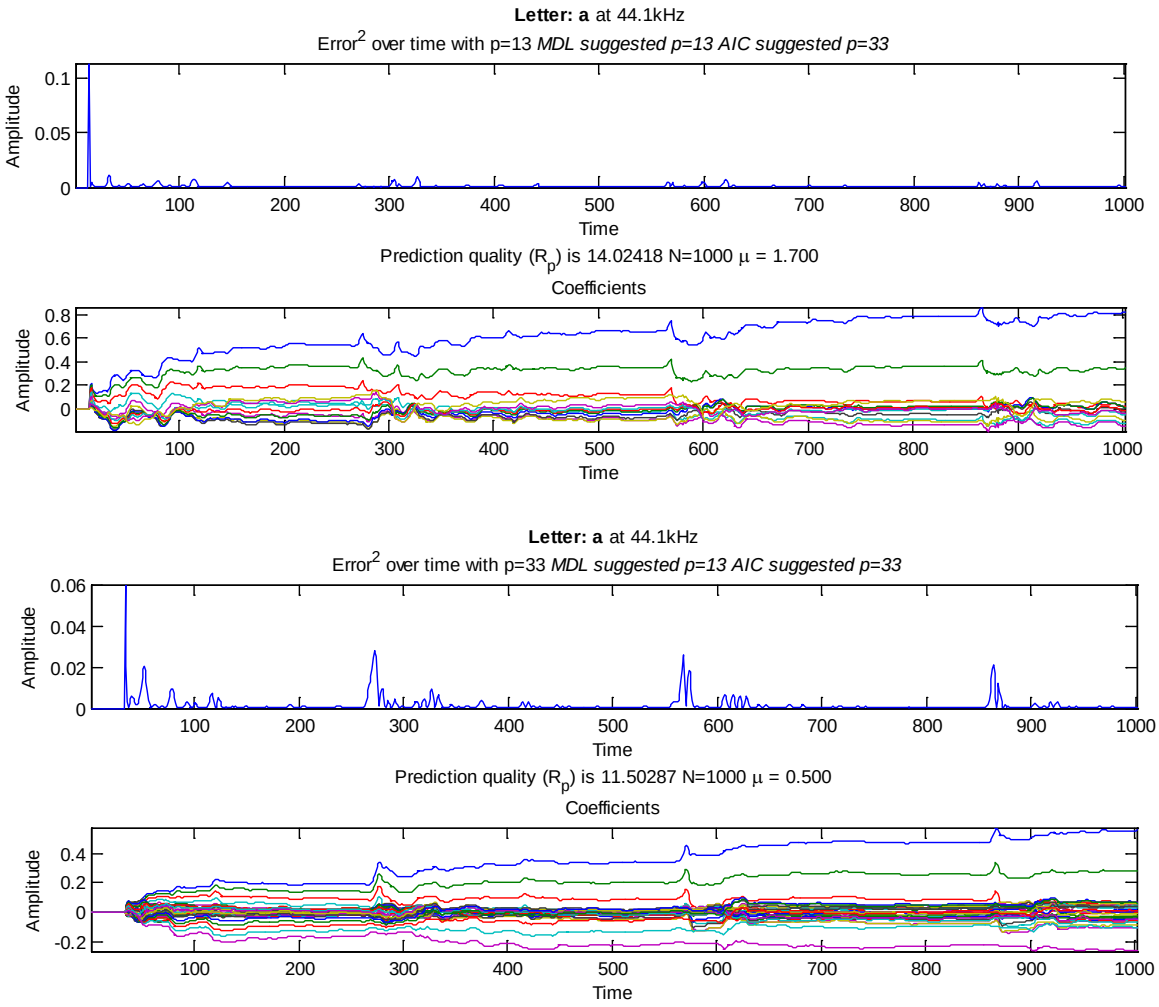


Figure 9 – Plot of the evolution of coefficients over time for an order of 13 and 33 on each graph which are the orders suggested by the MDL and AIC criterion, respectively.

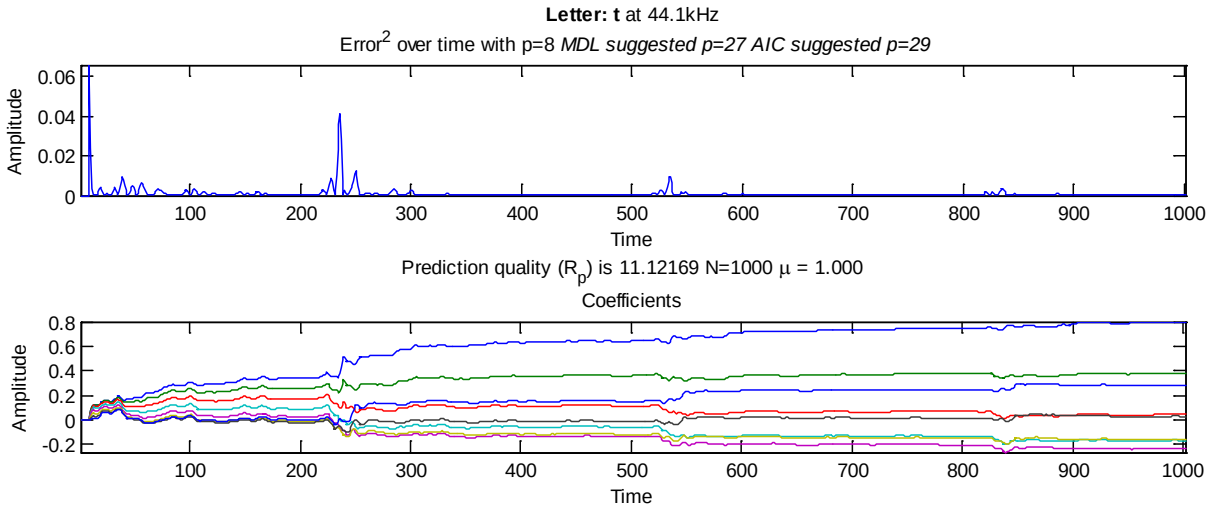


Figure 10 – Plot of the evolution of coefficients over time for the letter 't'. It can be seen that while the order is 8 there are at least 6 easily distinguishable from 0 coefficients.

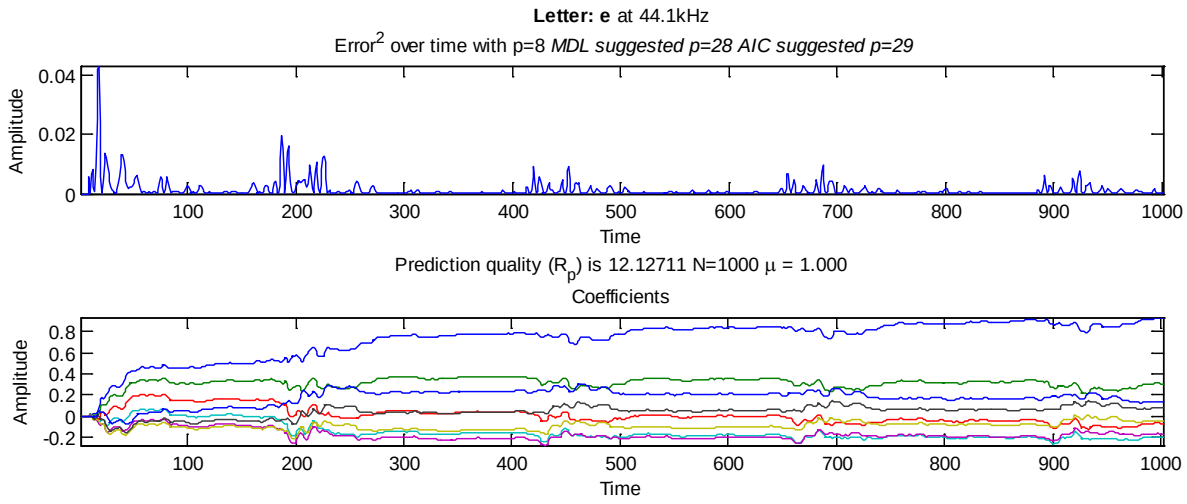


Figure 11 - Plot of the evolution of coefficients over time for the letter 'e'. It can be seen that all 8 coefficients can still be distinguished with only 2 oscillating around a very low (maybe 0) value.

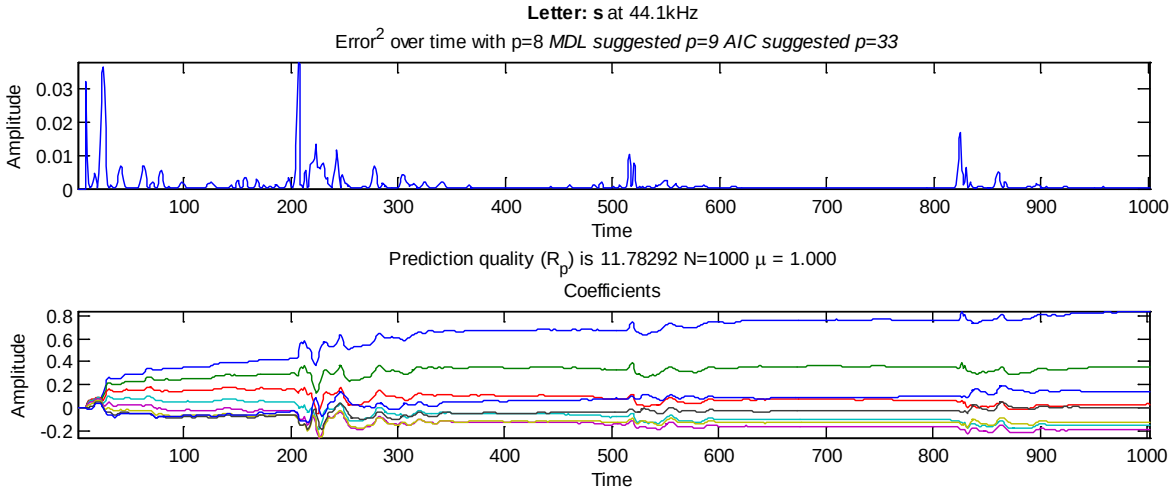


Figure 12 - Plot of the evolution of coefficients over time for the letter 's'. It can be seen that while the order is 8 there are at least 6 easily distinguishable from 0 coefficients.

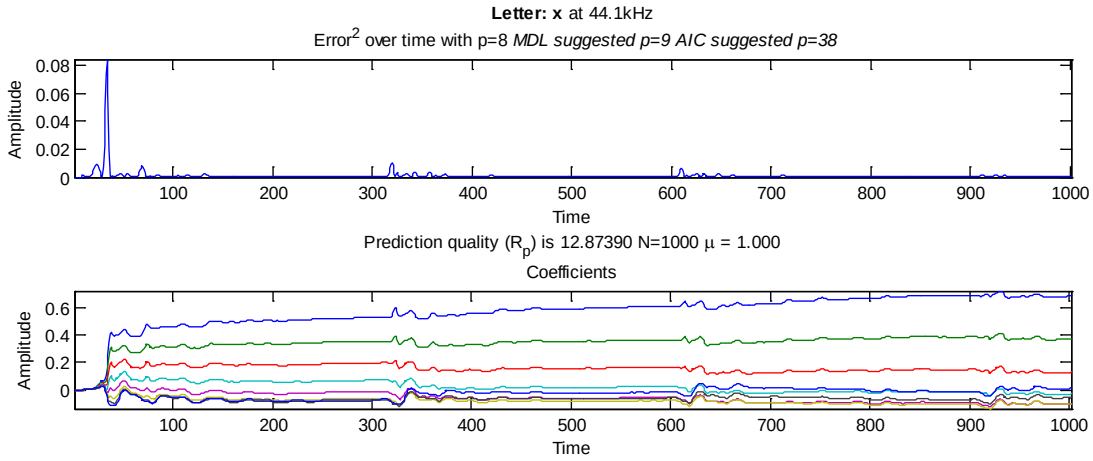


Figure 13 - Plot of the evolution of coefficients over time for the letter 'x'. It can be seen that while the order is 8 there are only 3 easily distinguishable from 0 coefficients.

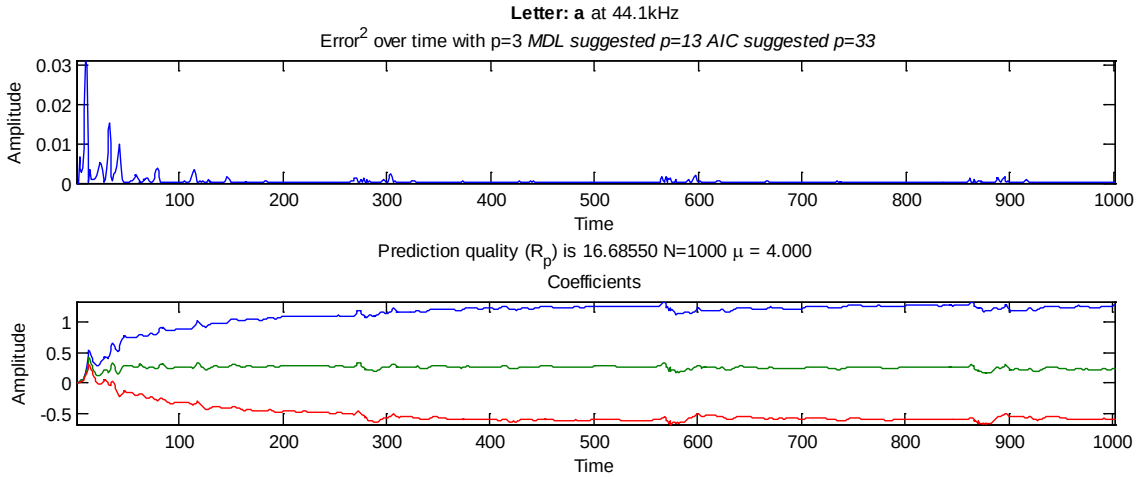


Figure 14 – Plot of the evolution of coefficients of letter a with p=3 resulting in a high $R_p = 16.68$ showing how a high order is not always required.

5.5.2 Effect of changing sampling frequency

Here we investigated the effect of changing sampling frequency with respect to prediction quality. Table 5 gives a quick overview of this effect by showing that sampling at 16 kHz leads to a less good prediction gain.

p=8 μ=1	**a**	**e**	**s**	**t**	**x**
Predicted Gain **44.1kHz**	12.69762	12.12711	11.78292	11.12169	12.8739
Predicted Gain **16kHz**	5.901248	7.221262	4.41588	5.839741	7.100865

Table 5 - Table comparing 44.1kHz and 16kHz sampling frequency AR prediction using prediction gain ($R_p = 10 \cdot log_{10}\left(\frac{\sigma_x^2}{\sigma_e^2}\right)$) to asses which is best. Higher values in green are judged to be better whereas low values are in red- judged to be worse.

It must however be considered that a 1000 samples covers different time frames for different sampling frequencies. For example $44.1kHz$ covers $\frac{1000}{44.1kHz} = 0.02268\,seconds$ whereas $16kHz$

covers $\frac{1000}{16kHz} = 0.0625\,seconds$. This is important as it means that for the same number of samples the AR process has to adapt to a wider range of AR coefficients - due to speech being non-stationary. Taking this into account it can still be expected that a lower sampling frequency is 'worse' due to the time distance between each sample is larger leading to more quantisation error arising leading to a worse performance. Indeed as observed in section 5.4 having more samples would mean the ability to have a lower learning constant which will mean that the AR coefficient equilibria will be less oscillatory.

Another way to appreciate the effect of different sampling frequencies is to look at the magnitude frequency response of the final values of various AR estimations. For example Figure 15 shows how there are still estimated spectral components of speech at frequencies beyond 16kHz, though the main ones are still present within this range. (Human's speech fundamental frequencies are between 80 and 400Hz).

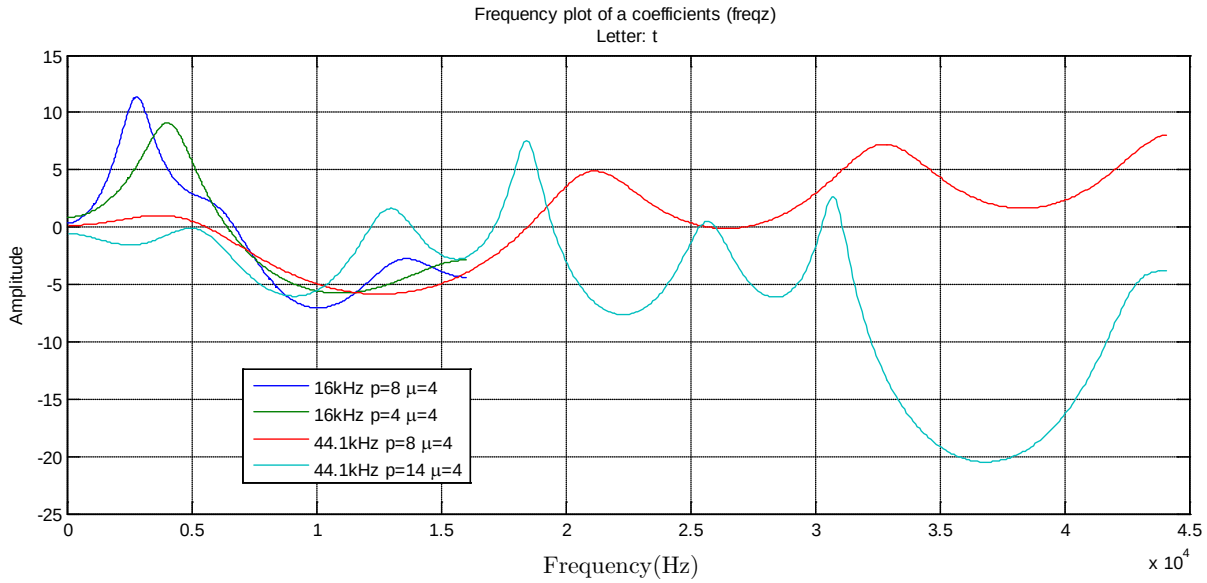


Figure 15 – Magnitude gain of AR process of final values for the letter 't' sampled at different frequencies for various orders.

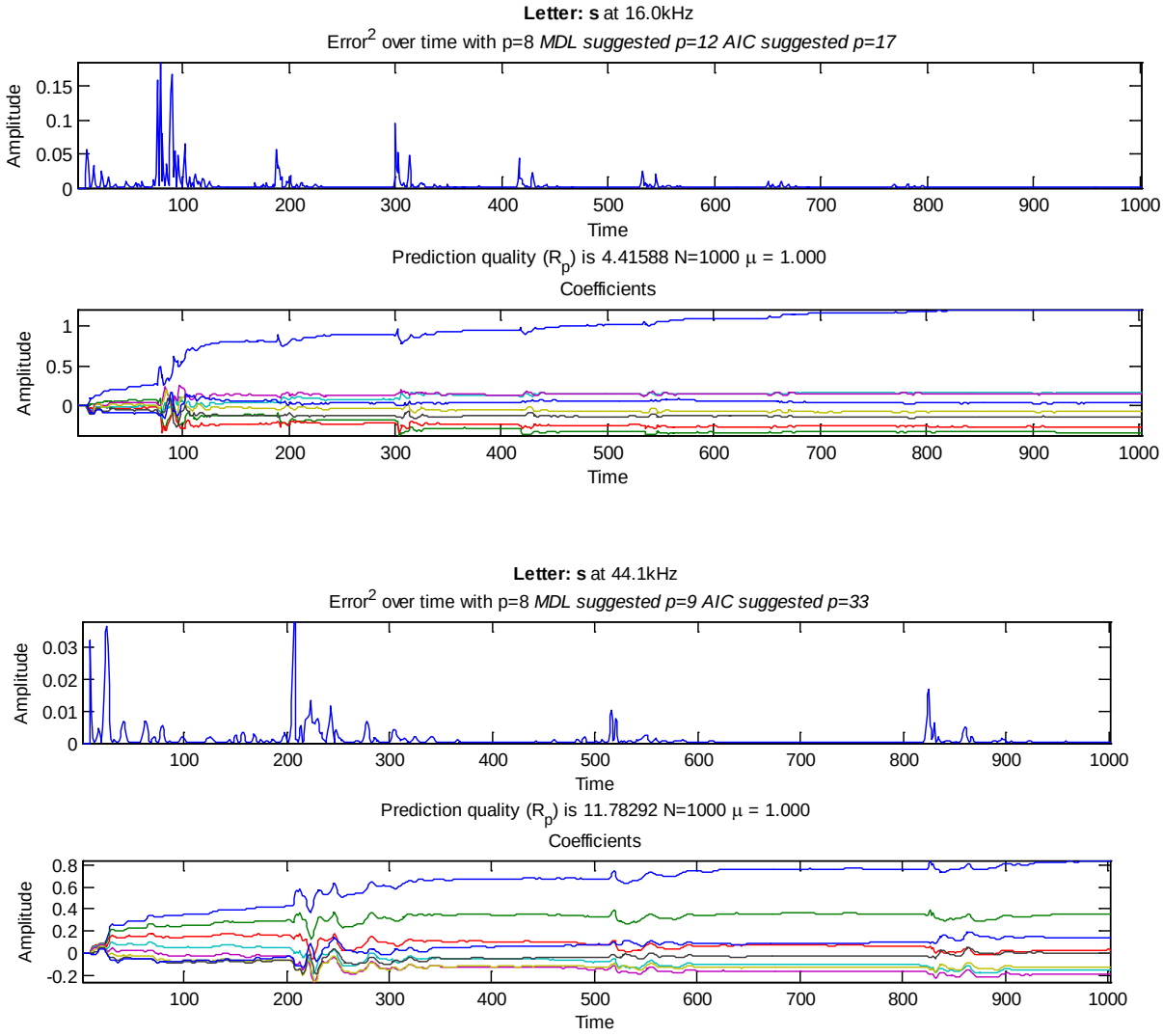


Figure 16 – Plot of the letter 's' sampled at 16kHz and 44.1kHz (above and below, respectively). As can be seen, for the same parameters, the prediction gain is worse for a lower sampling rate.

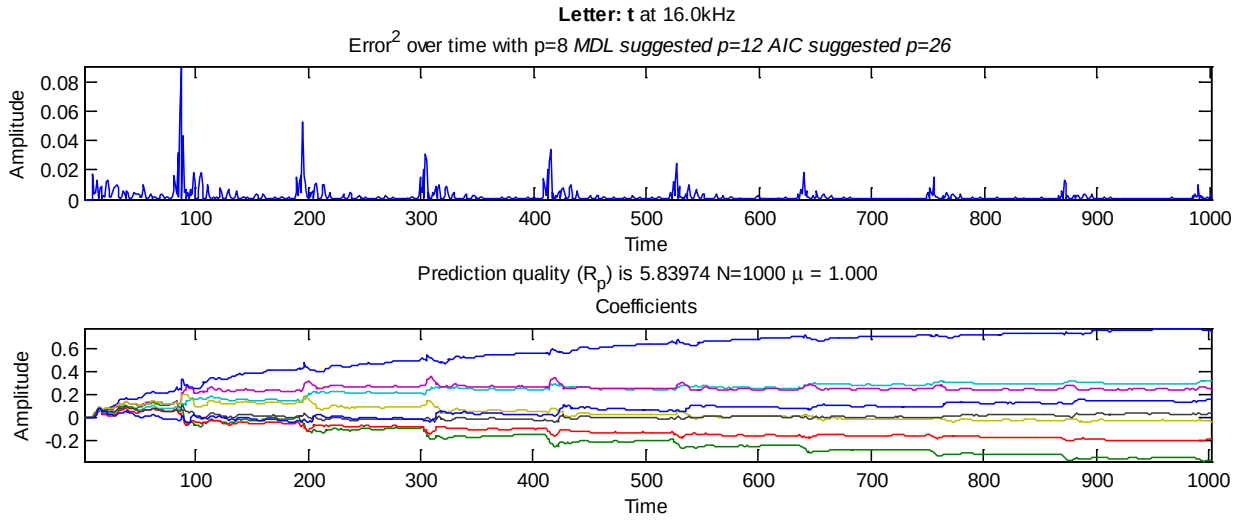


Figure 17 Plot of the letter 't' sampled at 16kHz. As this is essentially over a longer time frame than 44.1 kHz more frequent frequency jumps can be seen.

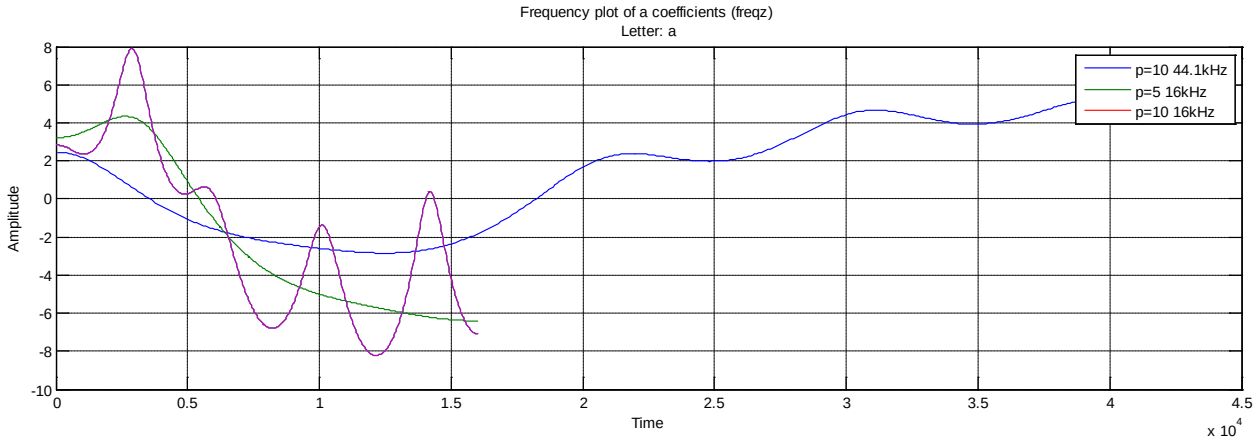


Figure 18 - Magnitude gain of AR process of final values for the letter 'a' sampled at different frequencies for various orders.

5.6 Dealing with computational complexity – sign algorithms

Sometimes LMS can be too intensive on hardware, leaving little place for extra computations to be done. Instead the following alternatives can be used:

$$Sign-error \rightarrow \quad \boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu \times sign(e[n])\boldsymbol{x}(n)$$

$$Sign-regressor \rightarrow \quad \boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu e[n] \times sign(\boldsymbol{x}(n))$$

$$Sign-Sign \rightarrow \quad \boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu \times sign(e[n]) \times sign(\boldsymbol{x}(n))$$

This was easily implemented by:

```
    switch j % allows choosing between each equation
        case 1
            a(i+1,:) = a(i,:) + mu*e(i)*x(i-1:-1:i-p)'; % normal
        case 2
            a(i+1,:) = a(i,:) + mu*sign(e(i))*x(i-1:-1:i-p)'; % signed - error
        case 3
            a(i+1,:) = a(i,:) + mu*e(i)*sign(x(i-1:-1:i-p))'; % signed  -regressor
        case 4
            a(i+1,:) = a(i,:) + mu*sign(e(i))*sign(x(i-1:-1:i-p))'; % signed  -regressora(i+1,:) = a(i,:) +
mu*sign(e(i))*sign(x(i-1:-1:i-p))'; % signed  -regressor
    end
```

Their performance was compared to see if using any of these would make sense.

Figure 19 provides a comparison of all the LMS variations for the AR process of section 5.4. The following observations can be made:

- The sign-sign algorithm, as expected, moves only as fast as the learning meaning that its step size is restricted. This however also causes the estimation by the sign-sign algorithm to be very jagged in time. This is why the sign-sign algorithm was determined to be the worst of all 4 implementations – in addition to the fact that it consistently gave lower prediction gains (as seen from the figures below).
- The signed-error and signed-regressor algorithms converged a bit slower than the normal implementation. A workaround would be to increase the learning constant however this would also cause the convergence to be less stable.

The letters ‘a’ and ‘t’ were also investigated (Figure 21 and Figure 22) were it was found that different coefficients were need to get similar results. This is possibly due to speech being non-stationary and the step size of the 3 LMS algorithm adaptions to be restricted.

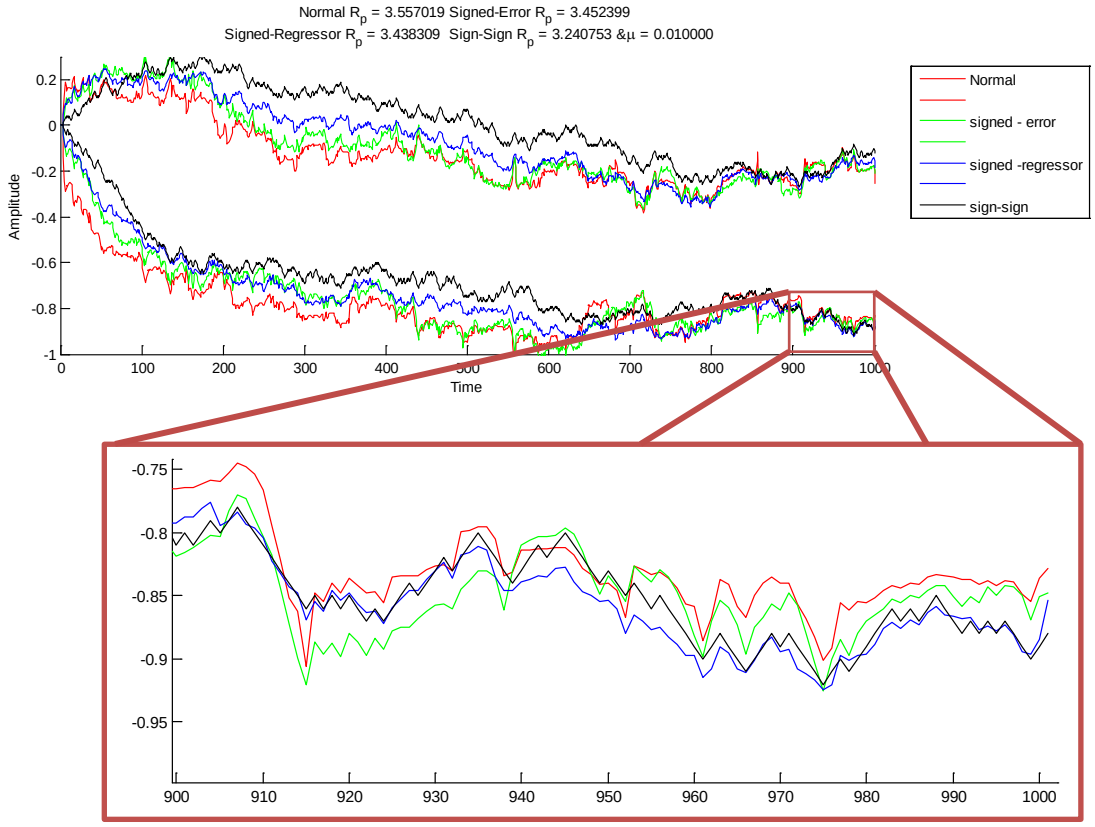


Figure 19 – Graph comparing the different algorithm implementations converging to the AR process a=[1 0.2 0.9]. It can be seen that the normal algorithm is the best algorithm to lock in to the correct values with a prediction gain of 3.55 whereas the sign-sign algorithm is worse having a prediction gain of 3.24. The remaining algorithms all result in prediction gains just under the non-altered LMS equation (3.43 and 3.45).

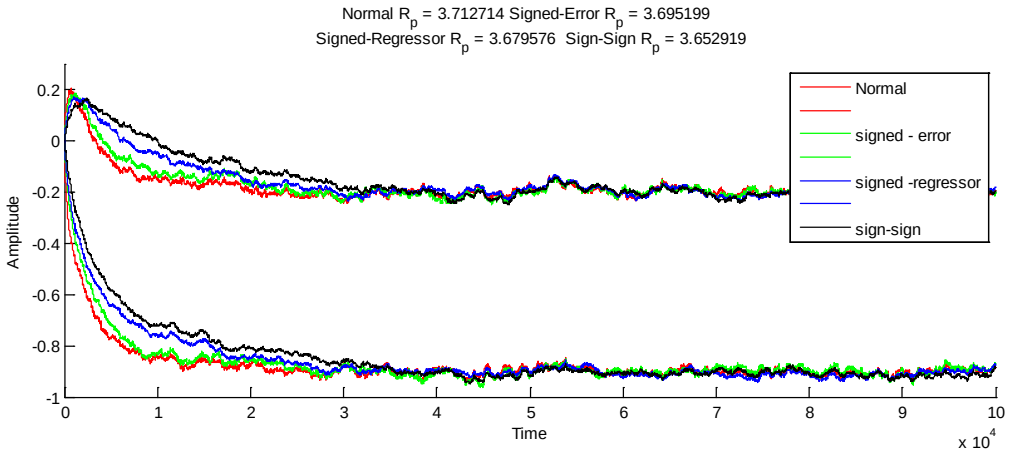


Figure 20 - Graph comparing the different algorithm implementations converging to the AR process a = [1 0.2 0.9]. This time 10000 samples are used instead of 1000 to show how over a longer period of time all implementations converge.

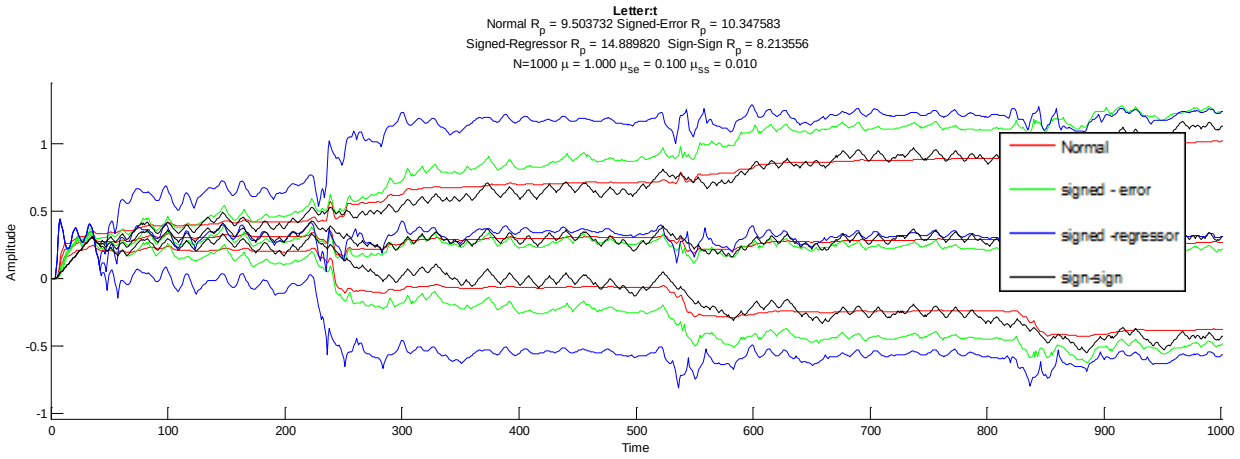


Figure 21 – Graph comparing the different implementations for the letter 't' using different μ values for different algorithms. This was done because each algorithm has μ values (found experimentally) which would converge at a better rate than for other algorithms.

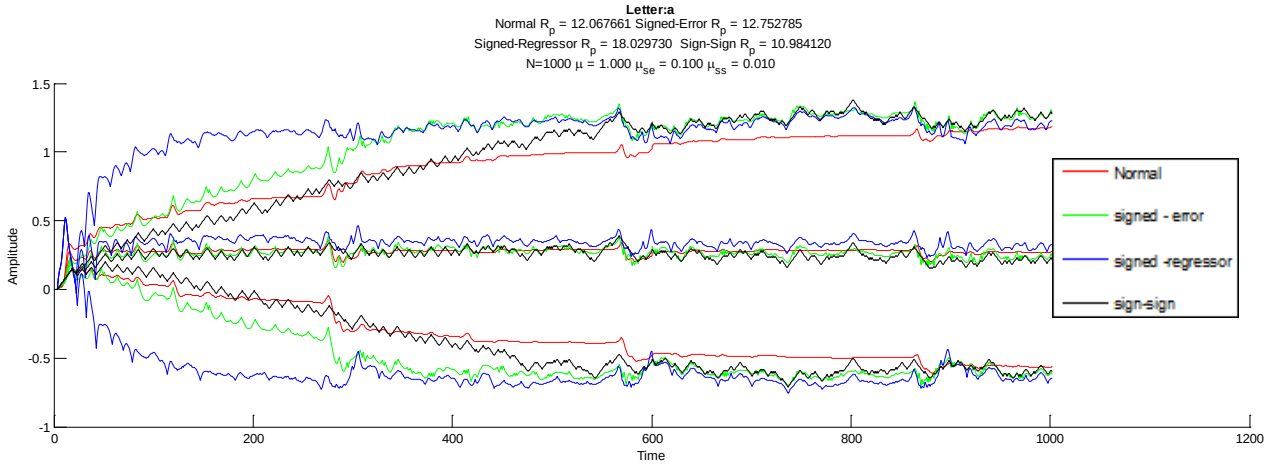


Figure 22 - Graph comparing the different implementations for the letter 't' using different μ values for different algorithms. This was done because each algorithm has μ values (found experimentally) which would converge at a better rate than for other algorithms.

Appendix

Table of figures

Matlab code

Part 5.1

```
%% Setup
clc;
clear all;
close all;
N=1000;
N_w = 20;
x = randn(N,1);
plot(x);
%unknown system
b = [1 2 3 2 1];
a=1;
y = filter(b,a,x);
%normalize system y to have unity variance
%y = y./sqrt((sum(b.*b)));
%generate noise of 0.1 std.dev
n = 0.1.*randn(N,1);
z=n+y;
p_y = sum(y.^2)/1000;
p_n = sum(n.^2)/1000;%get actual noise value
SNR = 10*log10(p_y/p_n);
fprintf('SNR is %f\n',SNR);
%% Part 5.1.1
r_xx_c = xcorr(x,'unbiased');
r_xx=zeros(N_w,N_w);
for i=1:N_w
    r_xx(:,i) = r_xx_c(N-i+1:N+N_w-i);
end
p_zx = xcorr(z,x,'unbiased');
p_zx = p_zx(N:N+N_w-1);
%% Part 5.1.2
w_opt = r_xx\p_zx;
```

LMS Algorithm

```
clear;clc;
N_w = 5;
N=1000;
x = randn(N,1);
b = [1 2 3 2 1];
a=1;
y = filter(b,a,x);
n = 0.1.*randn(N,1);
z=n+y;
mu = 0.01;
w=zeros(N,N_w);
% inputs x,z,mu,N_w

%% function start
%[wp1 e w]
% function [w e] = lms_cust(x,z,mu,N_w)
%lms function
%takes in the noise, the filtered system and the learning constant in addition to the order of the filter
%returns the estimated coefficients over time as well as the errror
N=length(x);
w=zeros(N,N_w);
y_h = zeros(N-N_w,1);
```

```
for i=N_w:N
y_h(i)=w(i,:)*x((i-N_w+1):i);
e(i) = z(i) - y_h(i);
w(i+1,:) = w(i,:) +mu*e(i)*x((i-N_w+1):i)';
end
w = w(2:N+1,:);
% end

%% plots data
figure(1);
subplot(2,1,1)
plot(w);
str = sprintf('Evolution of coefficient estimates over time using LMS and mu=%f',mu);
title(str);
legend('b1','b2','b3','b4','b5')
ylabel('Amplitude')
subplot(2,1,2)
plot(e.*e)
str = sprintf('Evolution of error sqaured over time using LMS and mu=%f',mu);
title(str);
ylabel('Amplitude')
xlabel('Time')
```

Part 5.3

```
clear;clc;
N_w = 5;
N=1000;
x = randn(N,1);
b = [1 2 3 2 1];
a=1;
y = filter(b,a,x);
n = 0.1.*randn(N,1);
z=n+y;
mu = 0.001;
w=zeros(N,N_w);
%inputs x,z,mu,N_w
%% function start
%[wp1 e w]
% function [w e] = lms_cust(x,z,mu,N_w)
N=length(x);
w=zeros(N,N_w);
y_h = zeros(N-N_w,1);
for i=N_w:N
y_h(i)=w(i,:)*x((i-N_w+1):i);
e(i) = z(i) - y_h(i);
w(i+1,:) = w(i,:) +mu*e(i)*x((i-N_w+1):i)';
% mu=min(0.006*exp(e(i)*1.1513)-0.001,0.2);
mu=min(max(0.055*e(i)-0.0733,0.01),0.2);
end
w = w(2:N+1,:);
% end

%% plots data
figure(1);
subplot(2,1,1)
plot(w);
str = sprintf('Evolution of coefficient estimates over time using LMS and final mu=%f',mu);
title(str);
legend('b1','b2','b3','b4','b5')
```

```
ylabel('Amplitude')
subplot(2,1,2)
plot(e.*e)
str = sprintf('Evolution of error^2 over time using LMS and final mu=%f',mu);
title(str);
ylabel('Amplitude')
xlabel('Time')
```

Part 5.4

```
clc; clear;
%initialise all variables
p=2;
N = 100000;
n = randn(N,1);
a = [1, 0.9, 0.2];
x= filter(1,a,n);
a = zeros(N,p);
% can also be made a function
%function [y_e, e, a] = LMS_AR(x, mu, p)
mu=0.0001;
e = zeros(N,1);
%perform AR recognition/estimation
y_h = zeros(N,1);
for i = p+1:N
    y_h(i) = a(i,:)*x(i-1:-1:i-p); %a1(i)*x(i-1)+a2(i)*x(i-2);
    e(i) = x(i) - y_h(i);
    a(i+1,:) = a(i,:) + mu*e(i)*x(i-1:-1:i-p)';
end
%end %use for fucntion, comment the rest out
%% plot everything
figure(1);
plot(e.*e);
figure(2);
clear figure(2);
plot(a);
str = sprintf('final a1:%f and final a2:%f estimation \n\\mu = %f',a(N,1),a(N,2),mu);
title(str);
ylabel('Amplitude')
xlabel('Time')
axis([0 N -1 .3])
```

Part 5.5

```
%% Adaptation from part 5.4
clc; clear;
file = 't.wav';
[x,fs] = wavread(file);
Fs=160;% in 100Hz set less than 441 to downsample
add = ceil(1000*441/Fs);
switch file(1)
  case 'a'
    x=x(39000:39000+add); %for a
  case 'e'
    x=x(23000:23000+add); %for e
  case 's'
    x=x(20000:20000+add); %for s
  case 't'
    x=x(37000:37000+add); %for t
  case 'x'
    x=x(16000:16000+add); %for x
```

```
end
x=resample(x,Fs,441);
[o1,o2, MDL, AIC] = aic_mdl(x,100);
p=mean(o1,o2);
p=8;
N = length(x);
a = zeros(N,p);
mu=1;
mu_ss = 0.01;
mu_se = 0.1;
e = zeros(N,1);
y_h = zeros(N,1);

for j=1:1
for i = p+1:N
    y_h(i) = a(i,:)*x(i-1:-1:i-p);
    e(i) = x(i) - y_h(i);
    switch j
        case 1
            a(i+1,:) = a(i,:) + mu*e(i)*x(i-1:-1:i-p)'; % normal
        case 2
            a(i+1,:) = a(i,:) + mu_se*sign(e(i))*x(i-1:-1:i-p)'; % signed - error
        case 3
            a(i+1,:) = a(i,:) + mu*e(i)*sign(x(i-1:-1:i-p))'; % signed  -regressor
        case 4
            a(i+1,:) = a(i,:) + mu_ss*sign(e(i))*sign(x(i-1:-1:i-p))'; % signed  -regressora(i+1,:) = a(i,:) +
mu*sign(e(i))*sign(x(i-1:-1:i-p))'; % signed  -regressor
    end
end
    switch j
        case 1
            normal = a;
            normal_e = e;
            normal_rp = 10*log10(var(x)/var(e));
            ss_a = 0 ;
            ss_e = 0;
            ss_rp = 0;
        case 2
            se_a = a;
            se_e = e;
            se_rp = 10*log10(var(x)/var(e));
        case 3
            sr_a = a;
            sr_e = e;
            sr_rp = 10*log10(var(x)/var(e));
        case 4
            ss_a = a;
            ss_e = e;
            ss_rp = 10*log10(var(x)/var(e));
    end
end
%% thing
figure(1);
subplot(2,1,1);
plot(e.^2);
axis tight;
str = sprintf('{\\bfLetter: %s} at %2.1fkHz\n Error^{2} over time with p=%d {\\itMDL suggested
p=%d AIC suggested p=%d}',file(1),Fs/10,p,o1,o2);
title(str);
```

```
ylabel('Amplitude')
xlabel('Time')
subplot(2,1,2);
plot(a);
R_p = 10*log10(var(x)/var(e));
fprintf('%f',R_p);
str = sprintf('Prediction quality (R_p) is %5.5f N=%d \\mu = %.3f\nCoefficients',normal_rp,N-1,mu);
title(str);
ylabel('Amplitude')
xlabel('Time')
axis tight;
[x,fs] = wavread(file);
x=resample(x,Fs,441);
x=filter(1,a(N,:),x);

sound = audioplayer(x,Fs*100);
play(sound);

[h,w] = freqz(1,a(N,:));
figure(2)
plot((1:length(h))*Fs*100/length(h),20*log10(abs(h)))
str = sprintf('Frequency plot of a coefficients (freqz) \nLetter: %s',file(1));
title(str);
ylabel('Amplitude')
xlabel('Time')
% legend('16kHz p=8 \mu=4','16kHz p=4 \mu=4','44.1kHz p=8 \mu=4','44.1kHz p=14 \mu=4')
grid on;
```

Part 5.6 -1

```
%% Adaptation from part 5.4
clc; clear;
p=2;
N = 1000;
n = randn(N,1);
a = [1, 0.9, 0.2];
x= filter(1,a,n);
a = zeros(N,p);
mu=0.01;
e = zeros(N,1);



y_h = zeros(N,1);
for j=1:4
for i = p+1:N
    y_h(i) = a(i,:)*x(i-1:-1:i-p);
    e(i) = x(i) - y_h(i);
    switch j
        case 1
            a(i+1,:) = a(i,:) + mu*e(i)*x(i-1:-1:i-p)'; % normal
        case 2
            a(i+1,:) = a(i,:) + mu*sign(e(i))*x(i-1:-1:i-p)'; % signed - error
        case 3
            a(i+1,:) = a(i,:) + mu*e(i)*sign(x(i-1:-1:i-p))'; % signed  -regressor
        case 4
            a(i+1,:) = a(i,:) + mu*sign(e(i))*sign(x(i-1:-1:i-p))'; % signed  -regressora(i+1,:) = a(i,:) +
mu*sign(e(i))*sign(x(i-1:-1:i-p))'; % signed  -regressor
    end
end
```

```
    switch j
        case 1
            normal = a;
            normal_e = e;
            normal_rp = 10*log10(var(x)/var(e));
        case 2
            se_a = a;
            se_e = e;
            se_rp = 10*log10(var(x)/var(e));
        case 3
            sr_a = a;
            sr_e = e;
            sr_rp = 10*log10(var(x)/var(e));
        case 4
            ss_a = a;
            ss_e = e;
            ss_rp = 10*log10(var(x)/var(e));
    end
end
figure(1);
plot(e);
figure(2);
clf(2);
hold;
plot(normal,'r');
plot(se_a,'g');
plot(sr_a,'b');
plot(ss_a,'k');
legend('Normal','','signed - error','','signed -regressor','','sign-sign')
R_p = 10*log10(var(x)/var(e));
str = sprintf('Normal R_p = %f Signed-Error R_p = %f \n Signed-Regressor R_p = %f  Sign-Sign
R_p = %f &\\mu = %f',normal_rp,se_rp,sr_rp,ss_rp,mu);
title(str);
ylabel('Amplitude')
xlabel('Time')
axis([0 N -1 .3])
```

Part 5.6 – 2

```
%% Adaptation from part 5.4
clc; clear;
file = 't.wav';
[x,fs] = wavread(file);
Fs=441;% in 100Hz set less than 441 to downsample
add = ceil(1000*441/Fs);
switch file(1)
  case 'a'
    x=x(39000:39000+add); %for a
  case 'e'
    x=x(23000:23000+add); %for e
  case 's'
    x=x(20000:20000+add); %for s
  case 't'
    x=x(37000:37000+add); %for t
  case 'x'
    x=x(16000:16000+add); %for t
end
x=resample(x,Fs,441);
[o1,o2, MDL, AIC] = aic_mdl(x,100);
p=mean(o1,o2);
```

```
p=3;
N = length(x);
a = zeros(N,p);
mu=2;
mu_ss = 0.01;
mu_se = 0.1;
e = zeros(N,1);
y_h = zeros(N,1);

for j=1:4
for i = p+1:N
    y_h(i) = a(i,:)*x(i-1:-1:i-p);
    e(i) = x(i) - y_h(i);
    switch j
        case 1
            a(i+1,:) = a(i,:) + mu*e(i)*x(i-1:-1:i-p)'; % normal
        case 2
            a(i+1,:) = a(i,:) + mu_se*sign(e(i))*x(i-1:-1:i-p)'; % signed - error
        case 3
            a(i+1,:) = a(i,:) + mu*e(i)*sign(x(i-1:-1:i-p))'; % signed  -regressor
        case 4
            a(i+1,:) = a(i,:) + mu_ss*sign(e(i))*sign(x(i-1:-1:i-p))'; % signed  -regressora(i+1,:) = a(i,:) +
mu*sign(e(i))*sign(x(i-1:-1:i-p))'; % signed  -regressor
    end
end
    switch j
        case 1
            normal = a;
            normal_e = e;
            normal_rp = 10*log10(var(x)/var(e));
            ss_a = 0 ;
            ss_e = 0;
            ss_rp = 0;
        case 2
            se_a = a;
            se_e = e;
            se_rp = 10*log10(var(x)/var(e));
        case 3
            sr_a = a;
            sr_e = e;
            sr_rp = 10*log10(var(x)/var(e));
        case 4
            ss_a = a;
            ss_e = e;
            ss_rp = 10*log10(var(x)/var(e));
    end
end
figure(1);
e = normal_e;
plot(e.*e);
figure(2);
clf(2);
hold;
plot(normal,'r');
plot(se_a,'g');
plot(sr_a,'b');
plot(ss_a,'k');
% legend('Normal','','signed - error','','signed -regressor','','sign-sign')
R_p = 10*log10(var(x)/var(e));
```

```
str = sprintf('\\bf{Letter:%s} \\rm{p=%d}\n',file(1),p);
str = [str sprintf('\\rm{Normal R_p = %f Signed-Error R_p = %f \n Signed-Regressor R_p = %f
Sign-Sign R_p = %f}',normal_rp,se_rp,sr_rp,ss_rp)];
str = [str sprintf('\nN=%d \\mu = %.3f \\mu_{se} = %.3f \\mu_{ss} = %.3f',N-
1,mu,mu_se,mu_ss)]
title(str);
ylabel('Amplitude')
xlabel('Time')
[x,fs] = wavread(file);
x=resample(x,Fs,441);
x=filter(1,a(N,:),x);

sound = audioplayer(x,Fs*100);
play(sound);
```