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 $\boldsymbol{w_{opt}} = \boldsymbol{R_{xx}^{-1} P_{zx}}$:

```
%% 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.

```matlab
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.

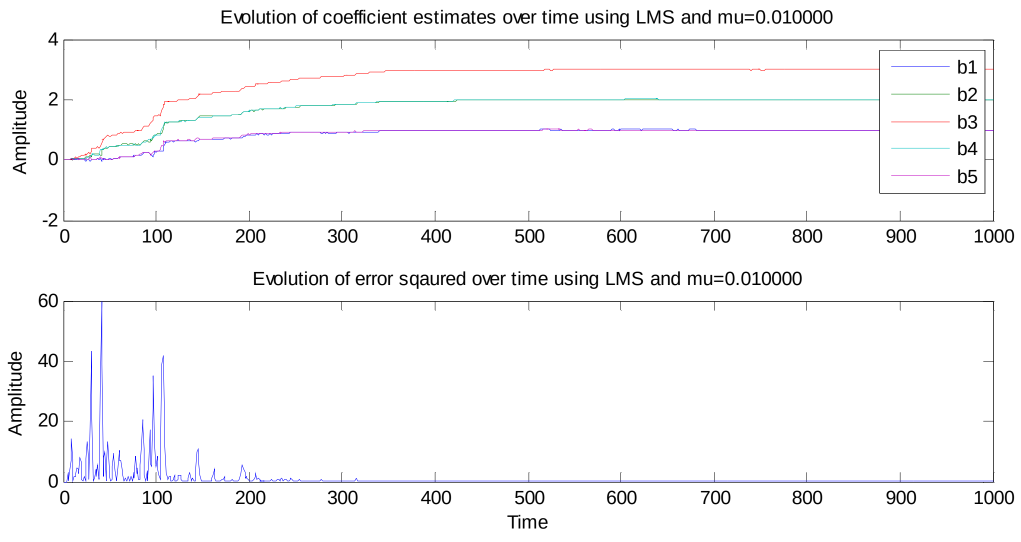

Figure 1 – Plot of adaptation of coefficients over time using the LMS function with its squared error function over time for $\mu = 0.01$. In this example the Wiener solution is reached at around 400 samples in time.

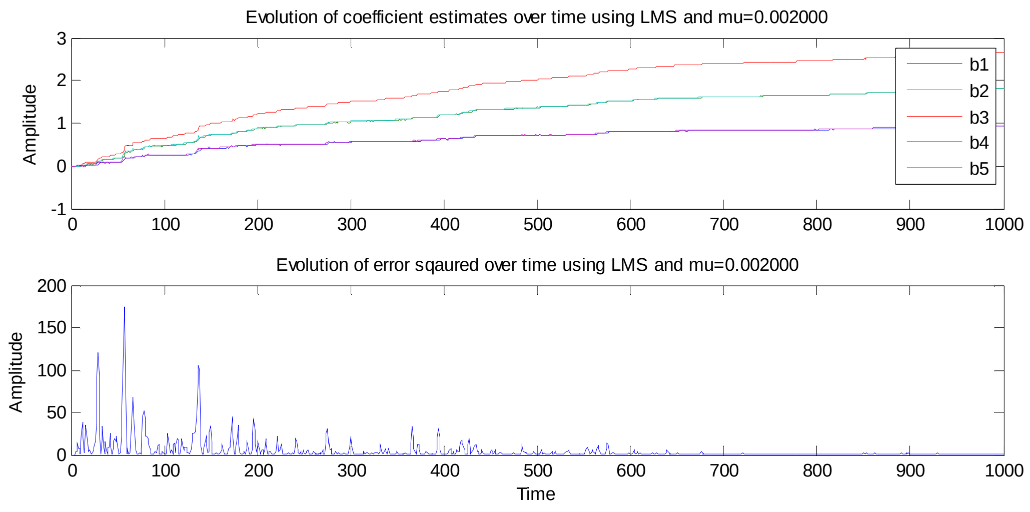

Figure 2 - Plot of adaptation of coefficients over time using the LMS function with its squared error function over time for $\mu = 0.002$. This time the Wiener solution is reached at a much slower pace that it was reached for Figure 1 (the error decays over time slower).

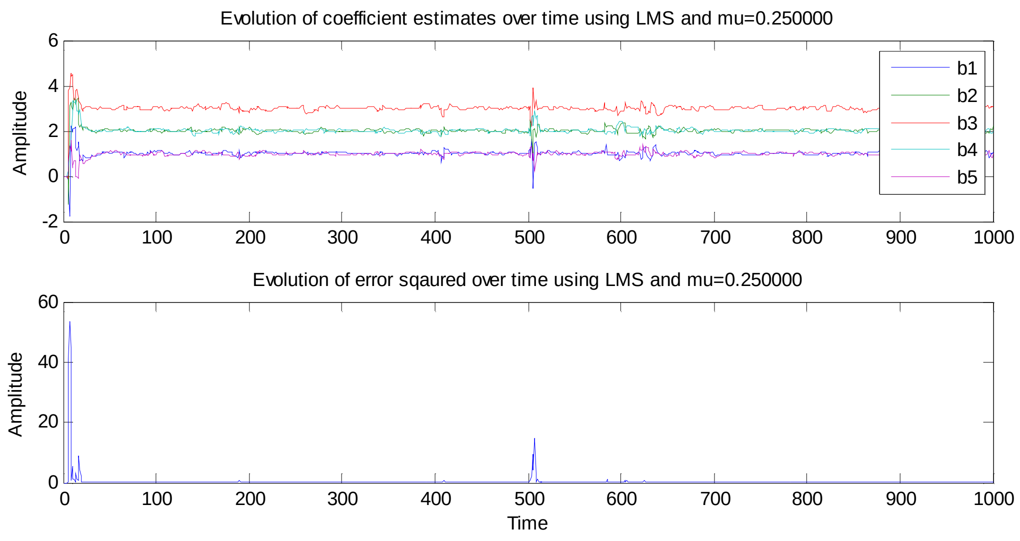

Figure 3 - Plot of adaptation of coefficients over time using the LMS function with its squared error function over time for $\mu = 0.25$. This is an example of overshoot where the Wiener solution values are reached extremely fast but many oscillations occur around the solution.

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 $O(4N_w + 1) \to O(N_w)$ 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

.

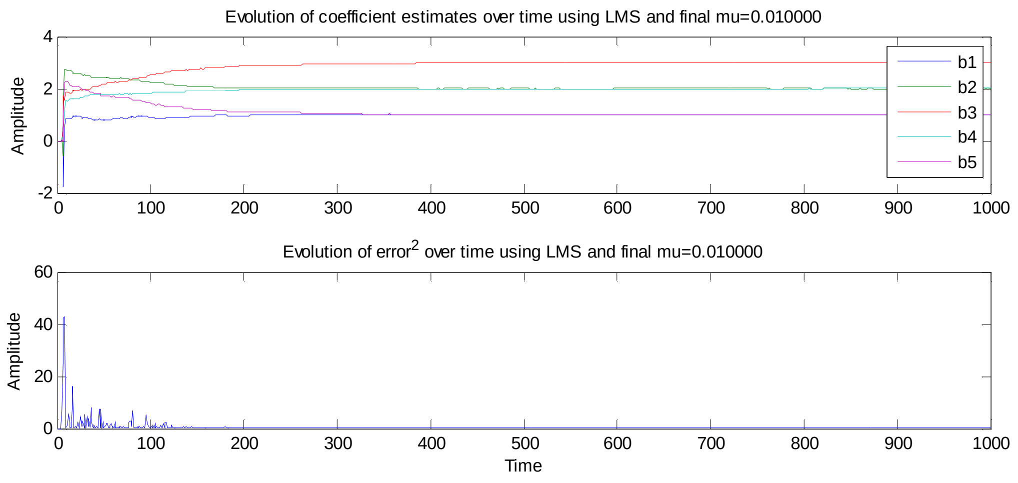

Figure 4 – LMS filter with μ=min(max(0.055*e(i)-0.0733,0.01),0.2) gear shifting. Values converge relatively quickly with the error going down reasonably fast.

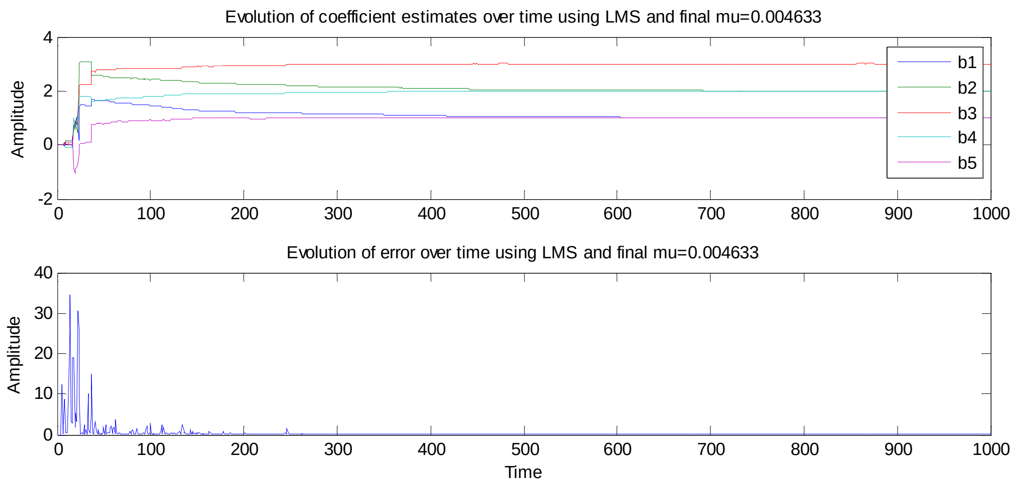

Figure 5 – LMS filter with μ=min(0.006*exp(e(i)*1.1513)-0.001,0.2) gear shifting. Values converge faster than the previous implementation with the error also going down faster – however it is larger at the start. The main advantage of this algorithm is the stability once converged to the Wiener solution.

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:

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

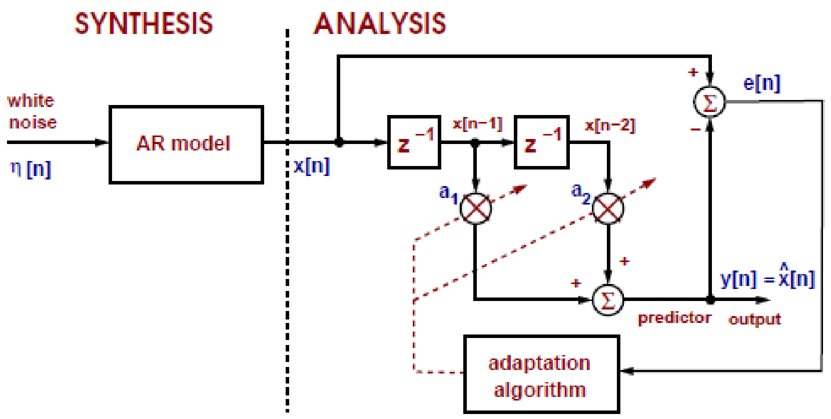

Figure 6 – Diagram of the structure of a LMS AR estimator

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

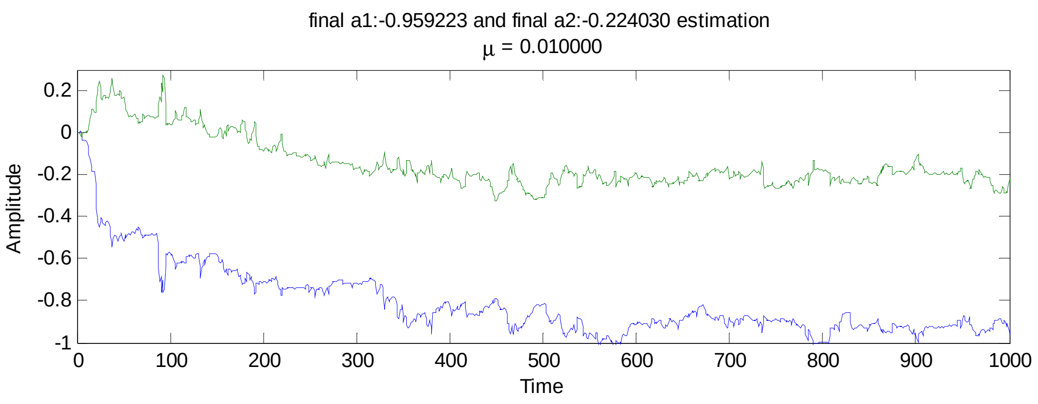

Figure 7 – LMS estimation of AR = [1, 0.9, 0.2] for N=1000 samples.

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.

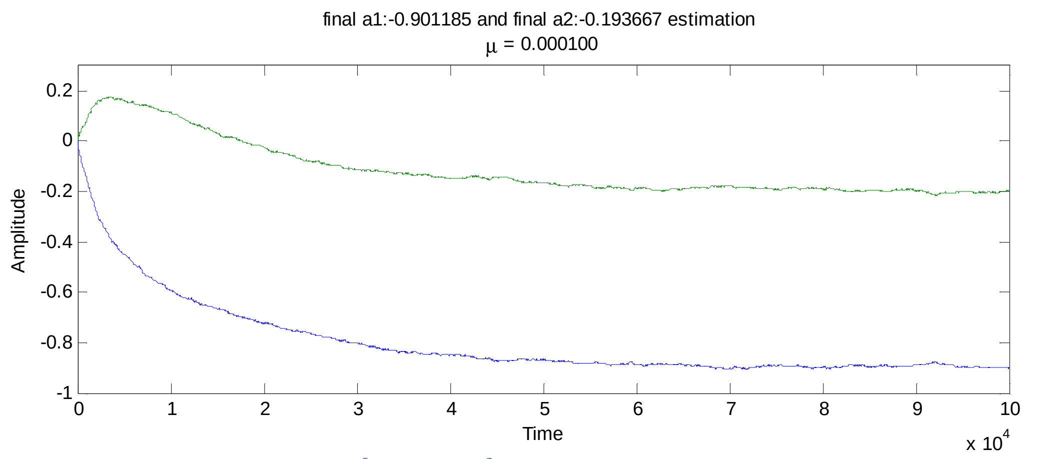

Figure 8– LMS estimation of AR = [1, 0.9, 0.2] for N=100000 samples for a learning constant of 0.0001

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**
$\mu = 0.1$	5.093	5.755	5.674	5.686
$\mu = 0.5$	9.839	9.877	9.770	9.998
$\mu = 1$	11.980	11.636	11.783	11.969
$\mu = 2$	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.

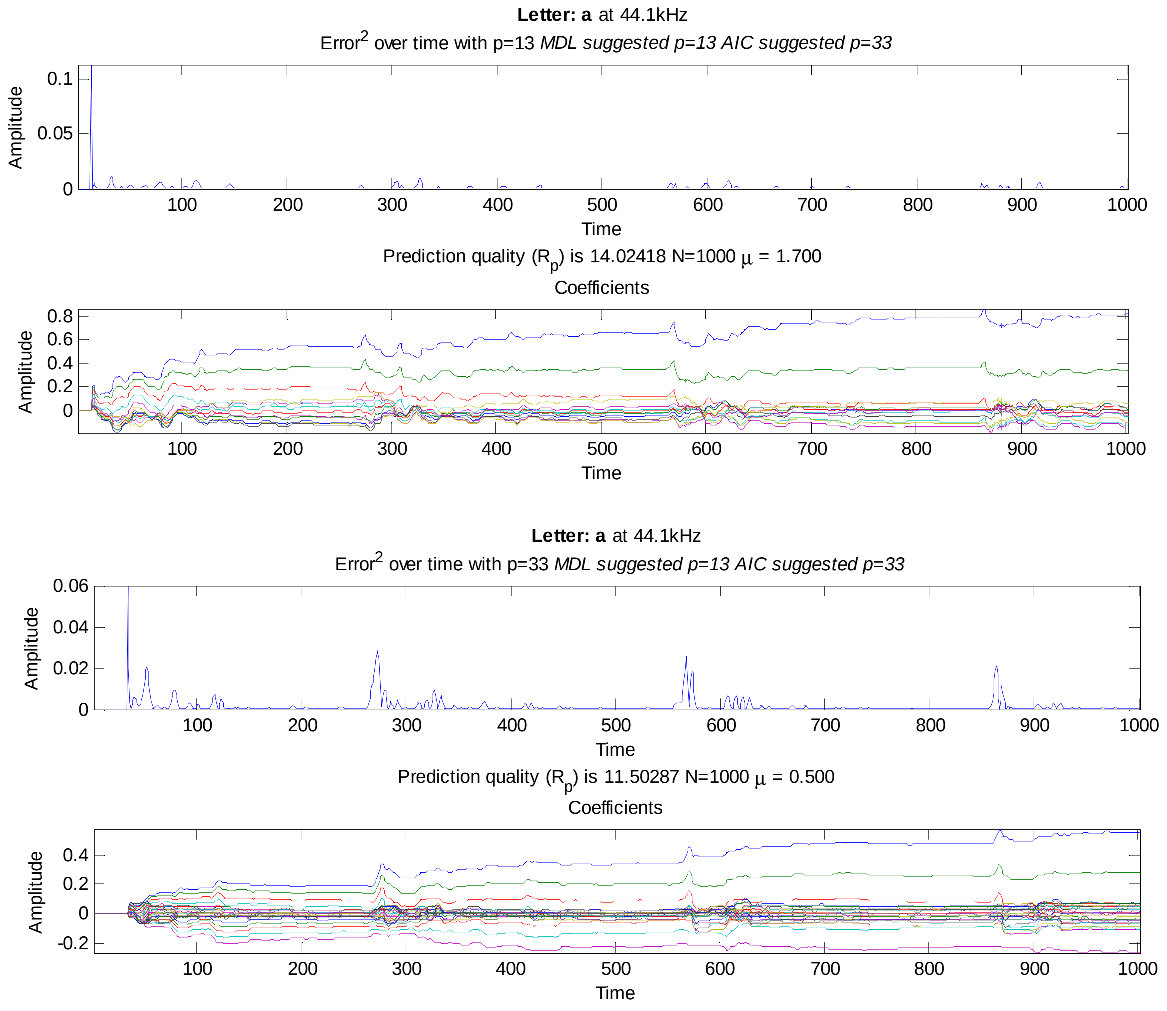

Figure 9 – Plot of the evolution of coefficients over time for an order of 13 and 33 on each graph which are the orders suggested by the MDL and AIC criterion, respectively.

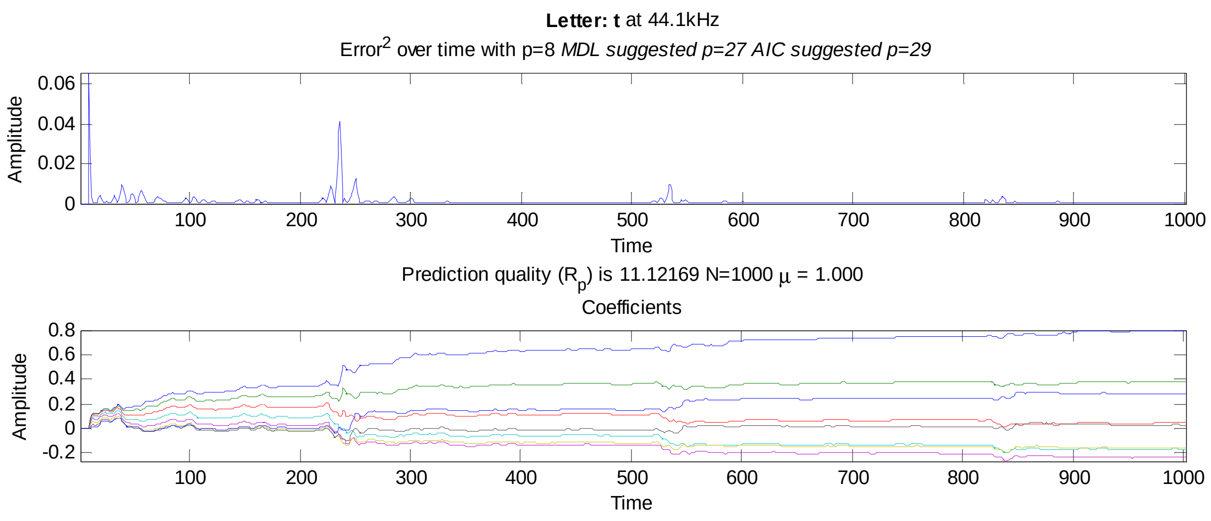

Figure 10 – Plot of the evolution of coefficients over time for the letter 't'. It can be seen that while the order is 8 there are at least 6 easily distinguishable from 0 coefficients.

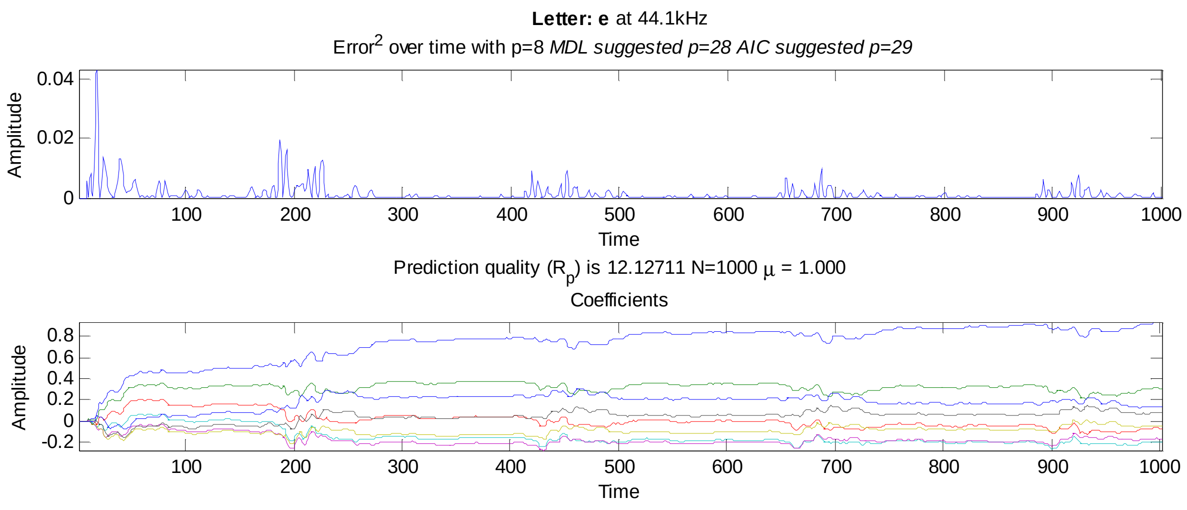

Figure 11 - Plot of the evolution of coefficients over time for the letter 'e'. It can be seen that all 8 coefficients can still be distinguished with only 2 oscillating around a very low (maybe 0) value.

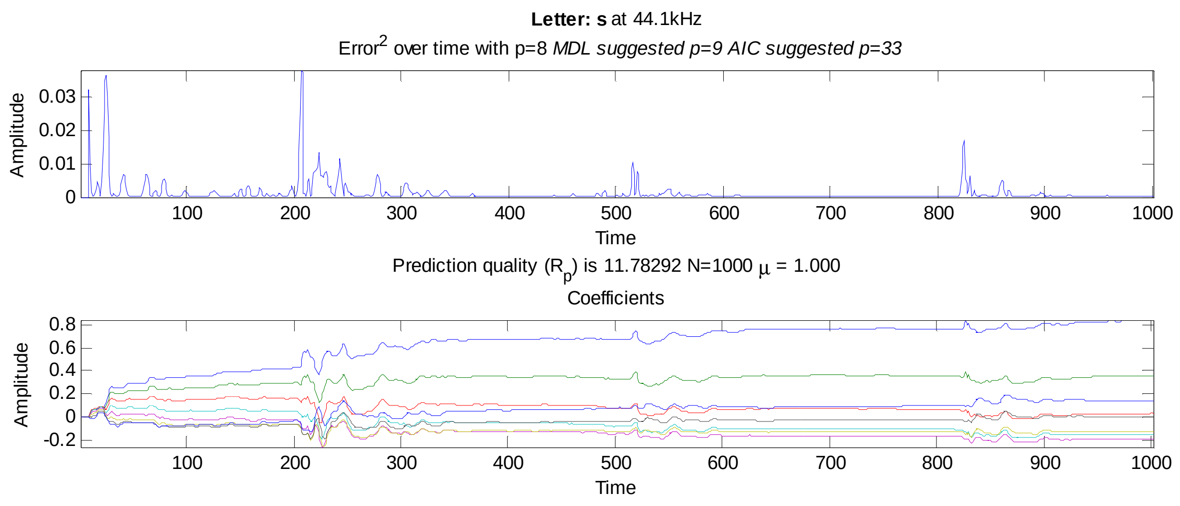

Figure 12 - Plot of the evolution of coefficients over time for the letter 's'. It can be seen that while the order is 8 there are at least 6 easily distinguishable from 0 coefficients.

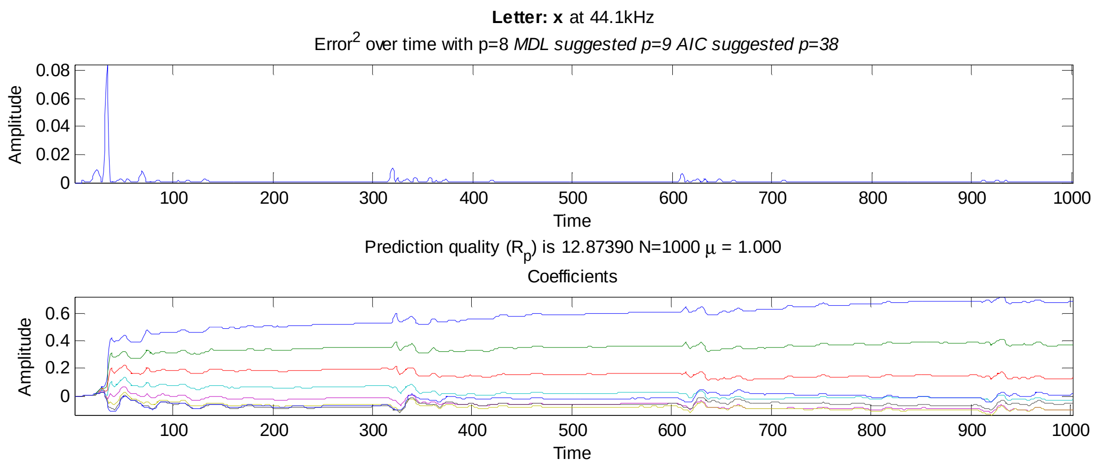

Figure 13 - Plot of the evolution of coefficients over time for the letter 'x'. It can be seen that while the order is 8 there are only 3 easily distinguishable from 0 coefficients.

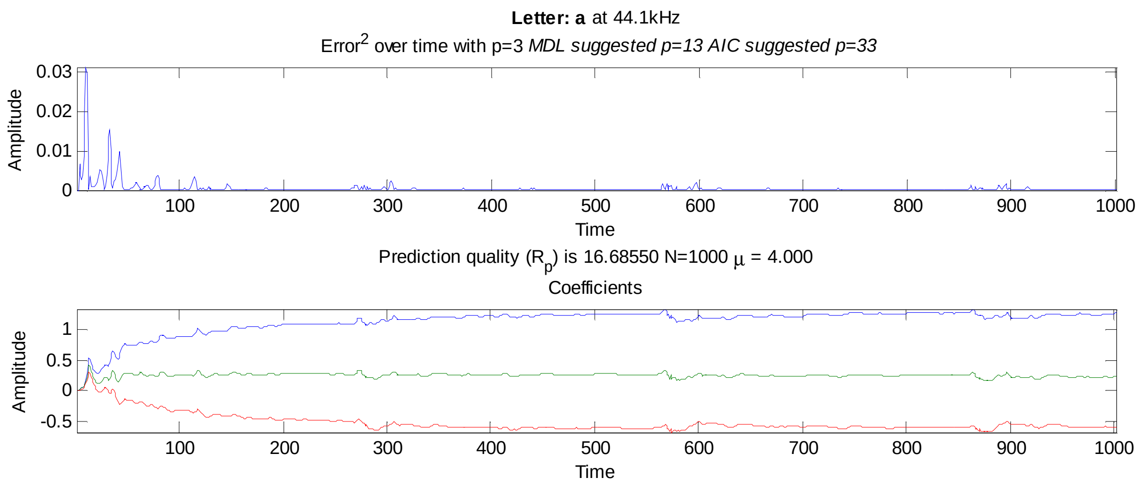

Figure 14 – Plot of the evolution of coefficients of letter a with p=3 resulting in a high $R_p = 16.68$ showing how a high order is not always required.

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).

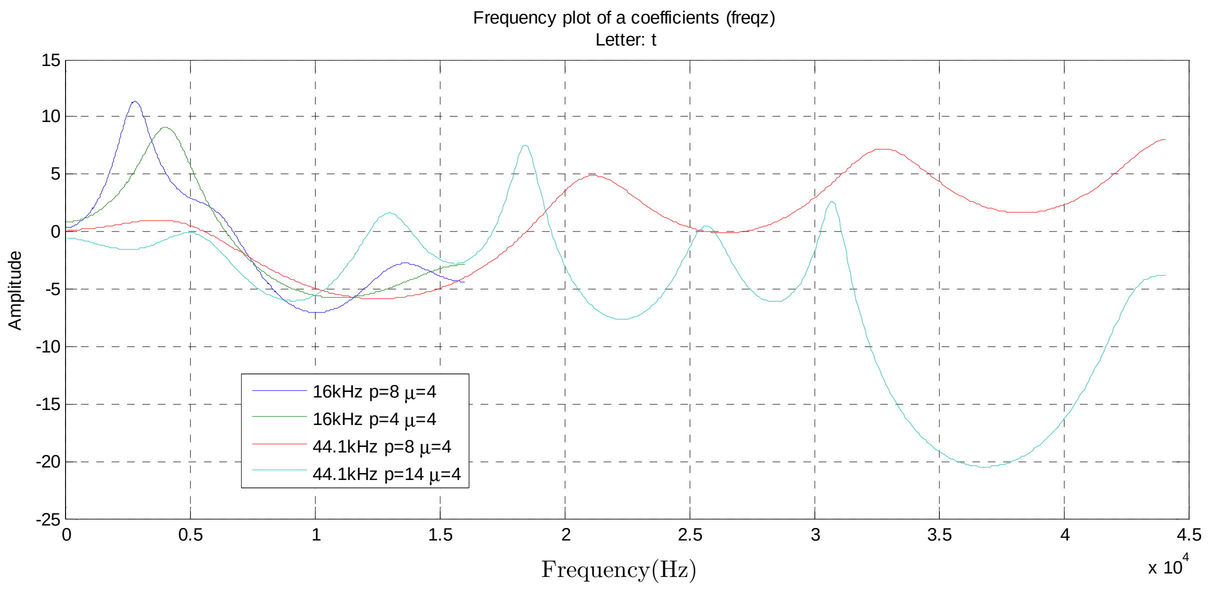

Figure 15 – Magnitude gain of AR process of final values for the letter 't' sampled at different frequencies for various orders.

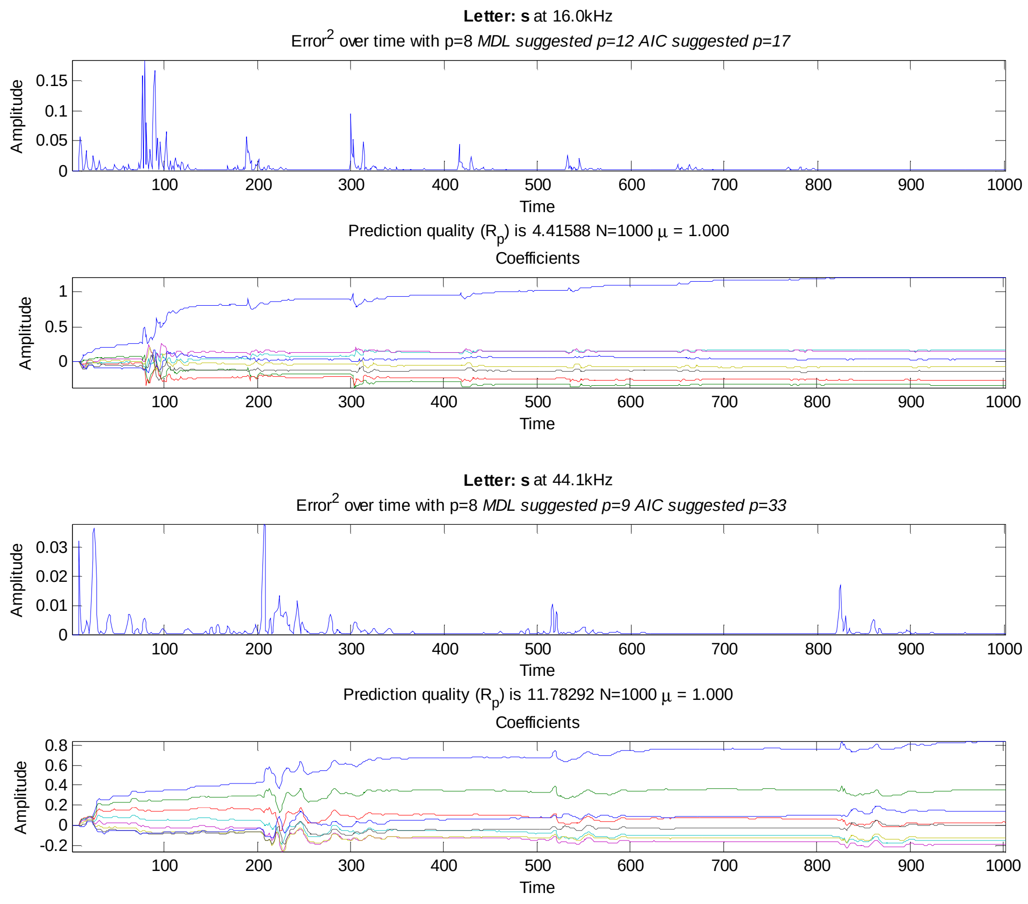

Figure 16 – Plot of the letter ‘s’ sampled at 16kHz and 44.1kHz (above and below, respectively). As can be seen, for the same parameters, the prediction gain is worse for a lower sampling rate.

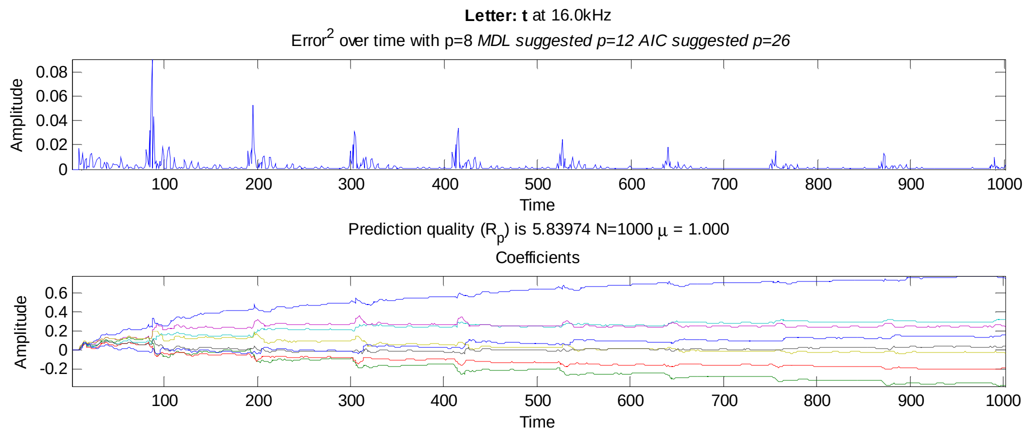

Figure 17 Plot of the letter ‘t’ sampled at 16kHz. As this is essentially over a longer time frame than 44.1 kHz more frequent frequency jumps can be seen.

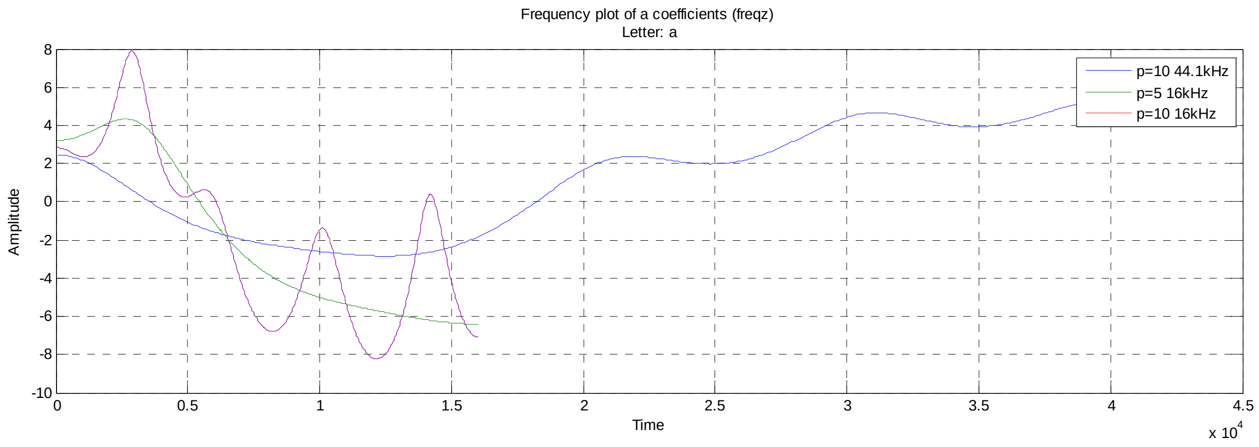

Figure 18 - Magnitude gain of AR process of final values for the letter 'a' sampled at different frequencies for various orders.

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.

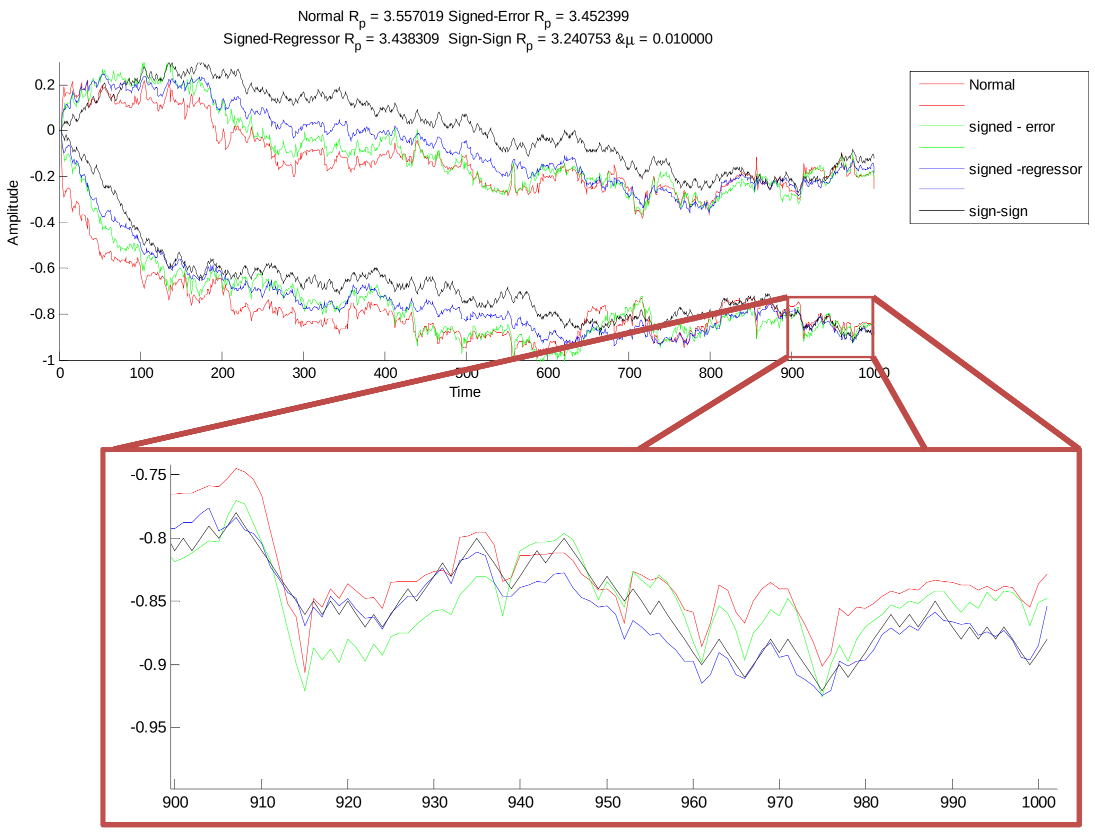

Figure 19 – Graph comparing the different algorithm implementations converging to the AR process a=[1 0.2 0.9]. It can be seen that the normal algorithm is the best algorithm to lock in to the correct values with a prediction gain of 3.55 whereas the sign-sign algorithm is worse having a prediction gain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

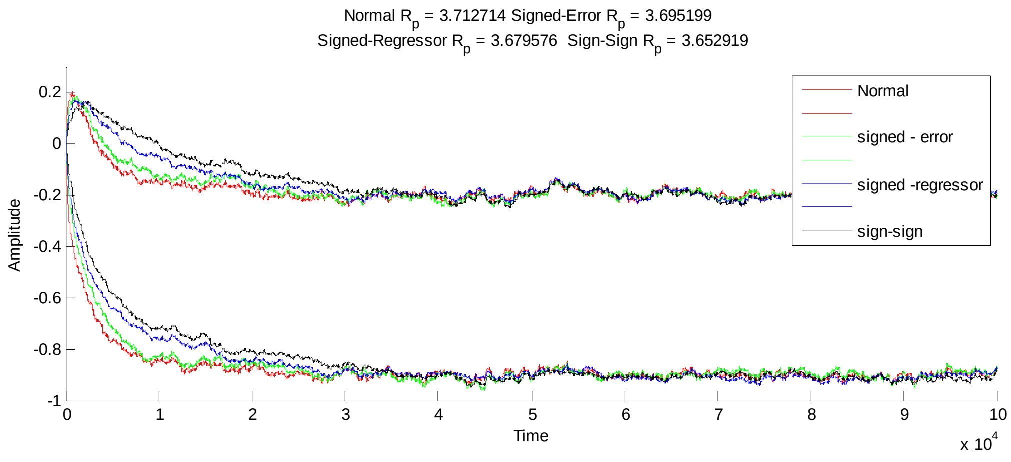

Figure 20 - Graph comparing the different algorithm implementations converging to the AR process a = [1 0.2 0.9]. This time 10000 samples are used instead of 1000 to show how over a longer period of time all implementations converge.

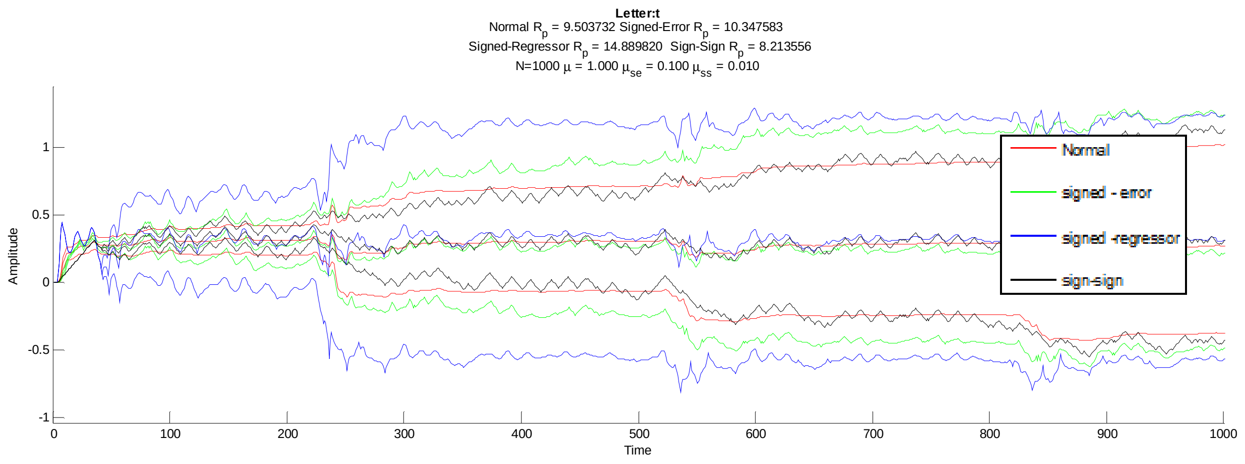

Figure 21 – Graph comparing the different implementations for the letter 't' using different μ values for different algorithms. This was done because each algorithm has μ values (found experimentally) which would converge at a better rate than for other algorithms.

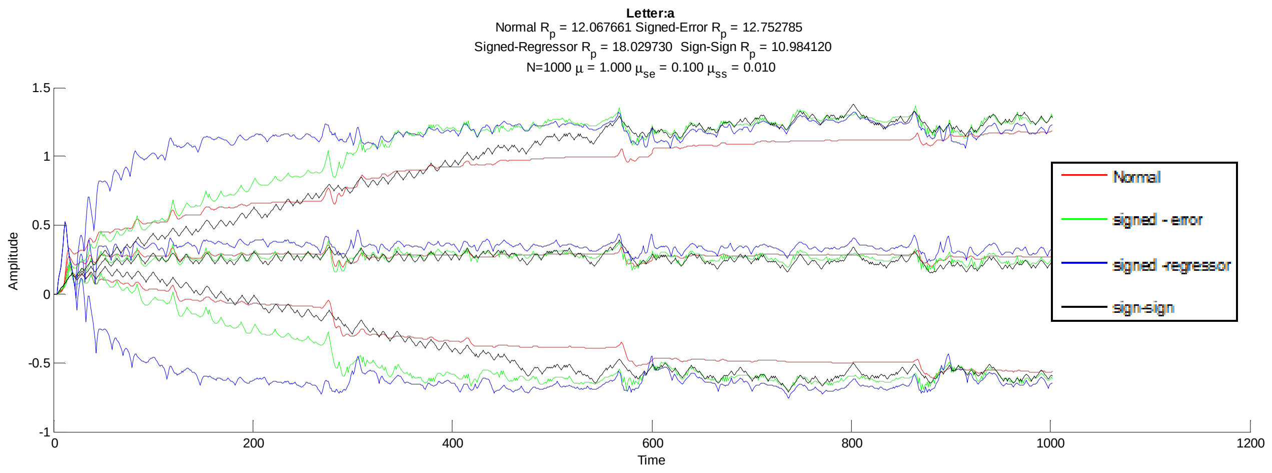

Figure 22 - Graph comparing the different implementations for the letter 't' using different μ values for different algorithms. This was done because each algorithm has μ values (found experimentally) which would converge at a better rate than for other algorithms.

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);
```