Imperial College London

Adaptive Signal Processing and Machine Intelligence

Yee Hong Low
YHL116
01202613

Contents

1 Classical and Modern Spectrum Estimation **2**
1.1 Properties of Power Spectral Density (PSD) . . . 2
1.2 Periodogram-based Methods Applied to Real–World Data . . . 3
1.3 1.3 Correlation Estimation . . . 5
1.4 1.4 Spectrum of Autoregressive Process . . . 7
1.5 1.5 Real World Signals: Respiratory Sinus Arrhythmia from RR-Intervals . . . 8
1.6 1.6 Robust Regression . . . 10

2 Adaptive Signal Processing **11**
2.1 The Least Mean Square (LMS) Algorithm . . . 11
2.2 Adaptive Step Sizes . . . 15
2.3 Adaptive Noise Cancellation . . . 17

3 Widely Linear Filtering and Adaptive Spectrum Estimation **23**
3.1 Complex LMS and Widely Linear Modelling . . . 23
3.2 Adaptive AR Model Based Time-Frequency Estimation . . . 29
3.3 A Real Time Spectrum Analyser Using Least Mean Square . . . 30

4 From LMS to Deep Learning **33**

1 Classical and Modern Spectrum Estimation

1.1 Properties of Power Spectral Density (PSD)

To show that the two definitions of PSD are equivalent, we start of from *Definition 2*:

$$P(\omega) = \lim_{x\to\infty} E\left\{\frac{1}{N}\left|\sum_{n=0}^{N-1} x(n)e^{-jn\omega}\right|^2\right\} \tag{1}$$

$$= \lim_{x\to\infty} E\left\{\frac{1}{N}\sum_{n=0}^{N-1} x(n)e^{-jn\omega}\sum_{m=0}^{N-1} x^*(m)e^{jm\omega}\right\} \tag{2}$$

$$= \lim_{x\to\infty} \frac{1}{N}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} E\left\{x(n)x^*(m)\right\} e^{-jn\omega}e^{jm\omega} \tag{3}$$

$$= \lim_{x\to\infty} \frac{1}{N}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} r_{xx}(n-m)e^{-j(n-m)\omega} \tag{4}$$

Substituting $k = n - m$ and reducing the double summation to a single summation, we get:

$$P(\omega) = \lim_{x\to\infty} \frac{1}{N}\sum_{k=-(N-1)}^{N-1} (N-|k|)r_{xx}(k)e^{-jk\omega} \tag{5}$$

$$= \sum_{k=-\infty}^{\infty} r(k)e^{-jk\omega} - \lim_{x\to\infty}\frac{1}{N}\sum_{k=-(N-1)}^{N-1} |k|r(k)e^{-jk\omega} \tag{6}$$

Finally, from the assumption that the covariance sequence r(k) decays rapidly, the second term equals vanishes to give:

$$P(\omega) = \sum_{k=-\infty}^{\infty} r(k)e^{-jk\omega} \tag{7}$$

1.2 Periodogram-based Methods Applied to Real–World Data

a. Figure 1 shows the periodogram of the sunspot data, with and without pre-processing.

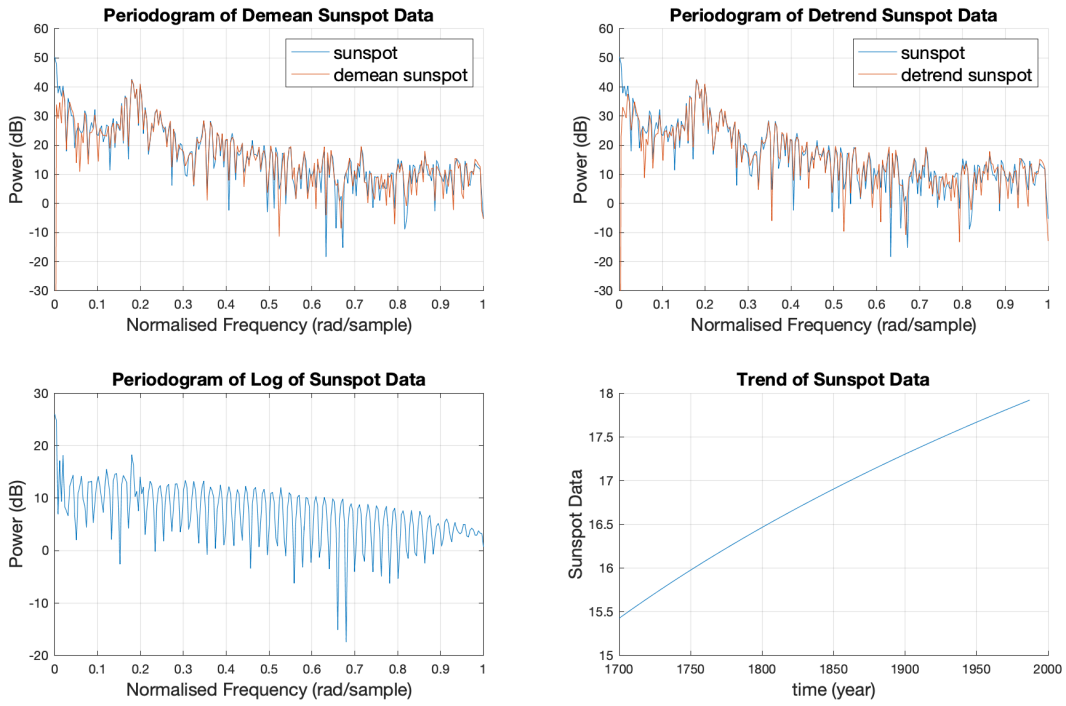

Figure 1: Periodogram of sunspot, detrend sunspot, demean sunspot and the trend of sunspot data.

The mean of the sunspot data is a constant function whereas the trend of the sunspot data is approximately linear. The original sunspot data can then be written as:

- original sunspot = demean sunspot + truncated constant function
- original sunspot = detrend sunspot + truncated linear function

A truncated constant function is a rectangular function, in which its Fourier Transform is a sinc function. From the linearity of Fourier Transform, the periodogram of the demean data is hence given by the periodogram of the original sunspot data minus a sinc function. This explains the huge decrease at $f = 0$ and also the fact that the periodogram of the demean data gets increasingly close to the peridogram of the original data as f increases.

The Fourier Transform of a linear function is a spike at $f = 0$ followed by a function proportional to $1/f^2$. Hence the removal of the trend removed the spike at $f = 0$. For the rest of the spectrum, the periodogram of the detrend data gets increasingly close to the periodogram of the original sunspot data as f increases.

As the original sunspot data contains 0, a small offset of 10^{20} is added to each of the values in the original sunspot data before the logarithm is taken. The periodogram of the sunspot data after the logarithm is taken

b.

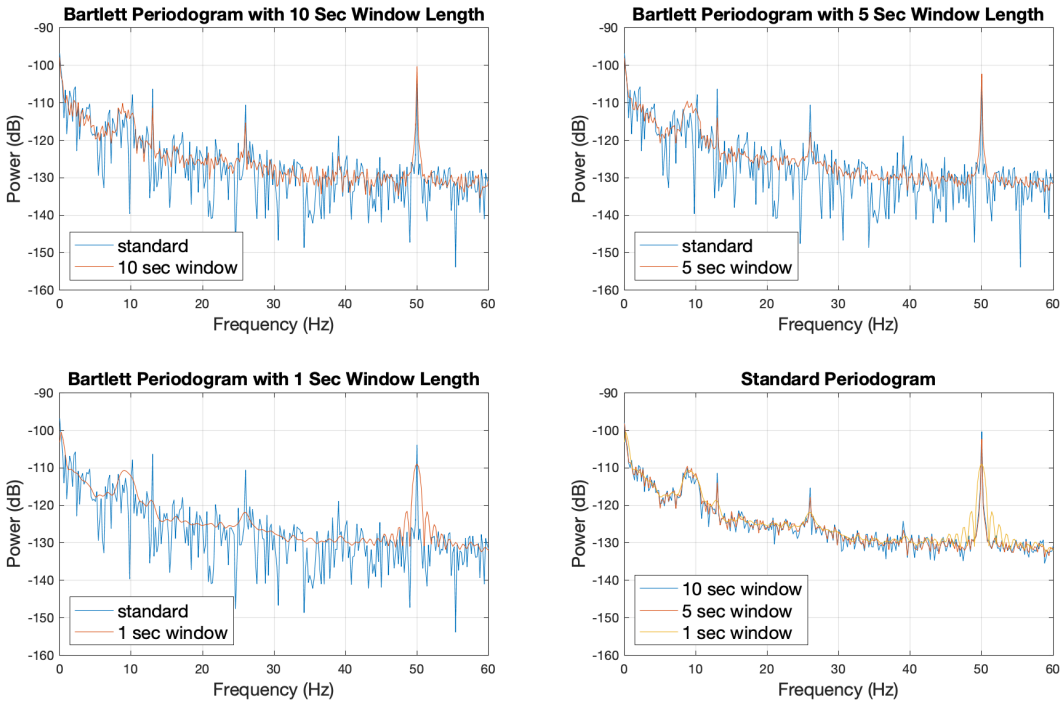

Figure 2: Sunspot data - Standard periodogram and averaged periodogram with different window size.

It is given that the peak at $f = 50Hz$ is due to the power-line interference while the strong response within $8 - 10Hz$ is due to the alpha rhythm. The peaks due to the SSVEP is hence given at $f = 13Hz, 26Hz$ and $39Hz$.

The averaged periodogram of window length 10s has a much lower variance as compared to the standard periodogram, making it easier to determine the peaks of the harmonics.

As the window size decreases, the variance of the periodogram decreases. This is due to the fact that as the window size decreases, the number of periodograms obtained increases. There are hence a greater number of periodograms to average over which leads to values with a lower variance.

Decreasing the window size also decreases the frequency resolution of the periodogram, making it more difficult to identify the peaks. The decrease in frequency resolution is caused by spectral leakage, which becomes more severe as the number of samples in each window decreases.

The effect of having a small window size can be seen in the Bartlett periodogram with 1 second window. It is clear that it has the lowest variance as compared to the other two Bartlett peridogram (with window size of 5 seconds and 10 seconds). It is also observed that it has a lower frequency resolution as depicted by the wider peaks. This made it more difficult to identify the peaks, especially the peak at 39Hz which is severely masked by a much larger peak at 50Hz.

1.3 1.3 Correlation Estimation

a. Figure 3 show the biased and unbiased autocorrelation function (ACF) estimates as well as the corresponding correlogram of three signals: white gaussian noise (WGN), noisy sine-wave, filtered WGN.

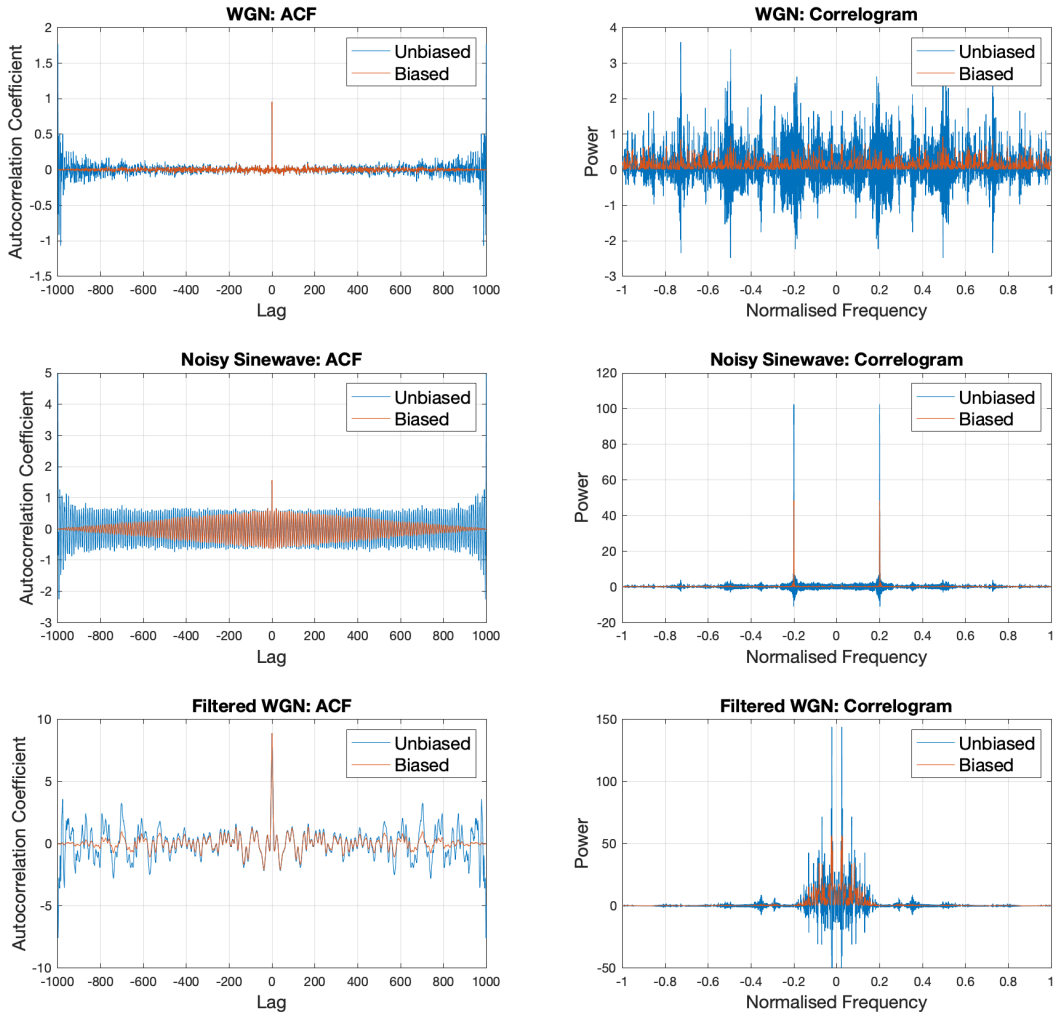

Figure 3: Biased and unbiased autocorrelation function (ACF) estimates, and their corresponding correlogram of three signals: white gaussian noise (WGN), noisy sine-wave, filtered WGN.

It is observed that both the ACF estimates are similar are small lags but the similarity decreases as the lag increases. The biased estimate seems to be squeezed at high lags, resulting in smaller values as compared to the unbiased estimate. It is also clear that the biased ACF estimate results in both positive and negative values for the estimated PSD while the unbiased ACF estimate only result in non-negative values for the estimated PSD.

b.

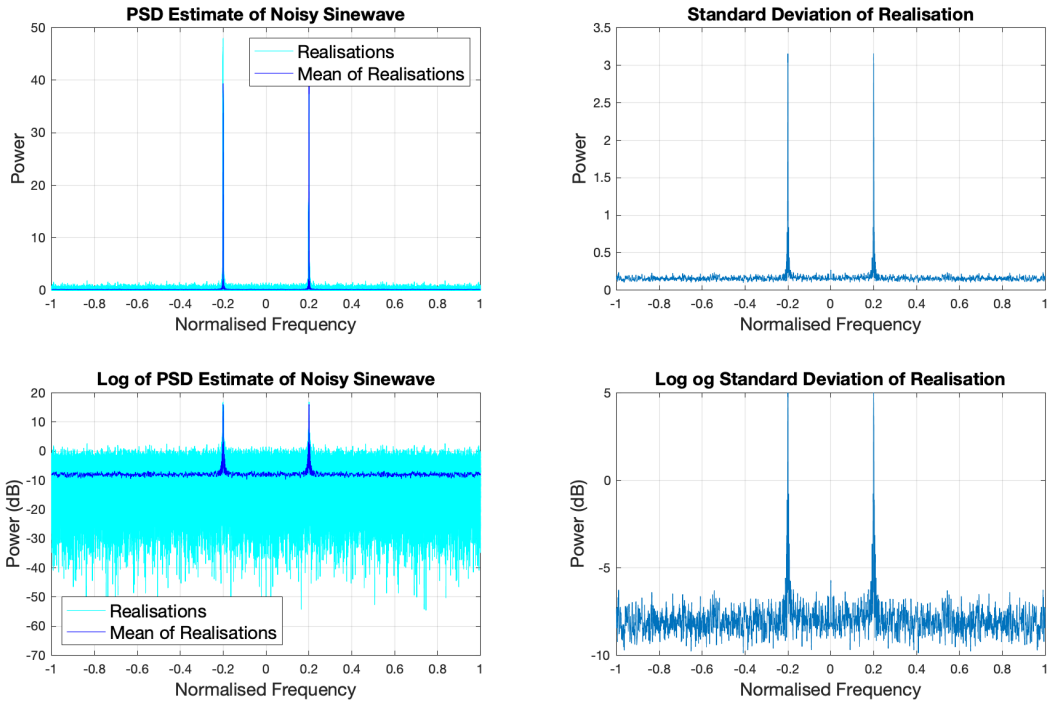

Figure 4: Mean and standard deviation of correlogram spectral estimates of 100 realisations.

It is observed that the standard deviation of the correlogram peaks at the frequency at which the mean of the correlogram also peaks. This infers that the correlogram spectral estimate if not a consistent estimator.

c. When the correlogram and its standard deviation are plotted in dB (in Figure 2), the point made in (b) is emphasized; further proving that the correlogram spectral estimate is not a consistent estimator.

d.

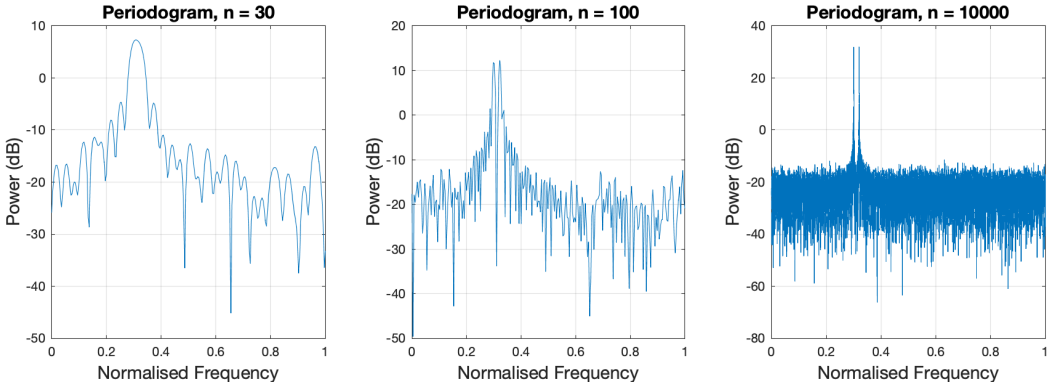

Figure 5: Periodogram with number of samples, n = 30, 100 and 10000.

From Figure 6, it is seen that when n increases, the peaks of the periodogram becomes more and more distinct. This is because the frequency resolution of the periodogram is proportional to $1/n$. To distinguish the peaks at 0.32 and 0.3, we need a frequency resolution of 0.02 which is approximately 50 samples. This is validated from the plots where at $n = 30$, the two peaks are indistinguishable. Hence, it is concluded that the frequency resolution of the periodogram increases with number of samples.

e. MUSIC (MUltiple SIgnal Classification) is an algorithm which uses an eigenspace method to estimate the frequency content of a signal or autocorrelation matrix. It assumes that a signal, $x(n)$, is the sum of p complex exponential and Gaussian white noise:

$$x(n) = \sum_{k=1}^{p} a_k e^{jn\omega_k} + WGN(n)$$

We define $\mathbf{R}_x$ as a $M \times M$ autocorrelation matrix of the signal where its eigenvalues are sorted in descesding order. The MUSIC algorithm assumes that the eigenvectors corresponding to the p largest eigenvalues spans the signal subspace. The remaining $M - p$ eigenvectors span the noise subspace. Note that the noise subspace and the signal subspace is orthogonal as the autocorrelation matrix, $\mathbf{R}_x$ is Hermitian. The frequency estimation function for MUSIC is given by:

$$\hat{P}_{MU}(e^{j\omega}) = \frac{1}{\sum_{M}^{u=p_1} |\mathbf{e}^H \mathbf{v}_i|^2}$$

where

$$\mathbf{e} = \begin{bmatrix} 1 & e^{j\omega} & e^{j2\omega} & \dots & e^{j(M-1)\omega} \end{bmatrix}^T$$

and v are the eigenvector of the noise subspace, $\mathbf{R}_n$.

The first line of the code gives the autocorrelation matrix, $\mathbf{R}_x$ denoted as `R`. `X` is n (n + m)-by-(m + 1) rectangular Toeplitz matrix. `x` is the input signal. The value 14 is the dimension of the autocorrelation matrix. This is selected based on the number of lags we can trust as bigger lags result in worse estimations. As for the second line performs the spectral estimation using the MUSIC algorithm. The argument `R` is again the autocorrelation matrix deduced from line 1. The second arguement, 2, is the dimension of th esignal subspace, p. The third and forth arguments are related to the length of the FFT and the sampling period. The last argument informs the function that the first arguement is a correlation matrix rather than a data matrix.

The MUSIC algorithm can estimate frequencies with a higher accuracy and with a higher frequency resolution as compared to the periodogram estimate when given the same number of samples. This can clearly seen by comparing the spectrum given by the MUSIC algorithm (Figure 6) and the periodogram (Figure 6) when $n = 30$. The MUSIC algorithm managed to produce two distinct peaks where as the periodogram is unable to do this. However, this is only the case when the number of components is known in advance as this information is needed to ignore the noise. Besides that, because the MUSIC algorithm essentially reduces the original data to a lower rank to reduce the effect of noise, it is disregarding information which might be useful.

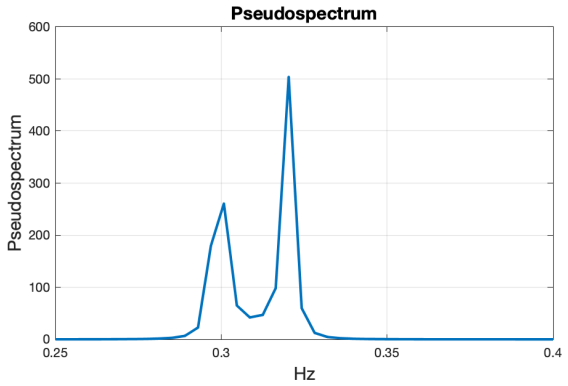

Figure 6: Pseudospectrum using the MUSIC method.

1.4 1.4 Spectrum of Autoregressive Process

a. To find the AR parameters, we can use the Yule-Walker equations which is given by the general AR(p) autocorrelation function:

$$\begin{aligned}
r_{xx}(1) &= a_1 r_{xx}(0) + a_2 r_{xx}(1) + \cdots + a_p r_{xx}(p-1) \\
r_{xx}(2) &= a_1 r_{xx}(1) + a_2 r_{xx}(2) + \cdots + a_p r_{xx}(p-2) \\
&\vdots = \vdots \\
r_{xx}(p) &= a_1 r_{xx}(p-1) + a_2 r_{xx}(p-2) + \cdots + a_p r_{xx}(0)
\end{aligned}$$

which can be expressed in a compact vector-matrix form as:

$$\mathbf{r}_{xx} = \mathbf{R}_{xx}\mathbf{a} \Rightarrow \mathbf{a} = \mathbf{R}_{xx}^{-1}\mathbf{r}_{xx}$$

where $\mathbf{R}_{xx}$ is the ACF estimate. The equation above can only have a solution if $\mathbf{R}_{xx}$ invertible. It is crucial that $\mathbf{R}_{xx}$ is positive definite as it guarantees matrix inversion. Given that the unbiased estimate is not always positive estimate, it is not guaranteed that it will yield a solution for $\mathbf{a}$.

b. The correlogram estimates using AR models are plotted in Figure 7.

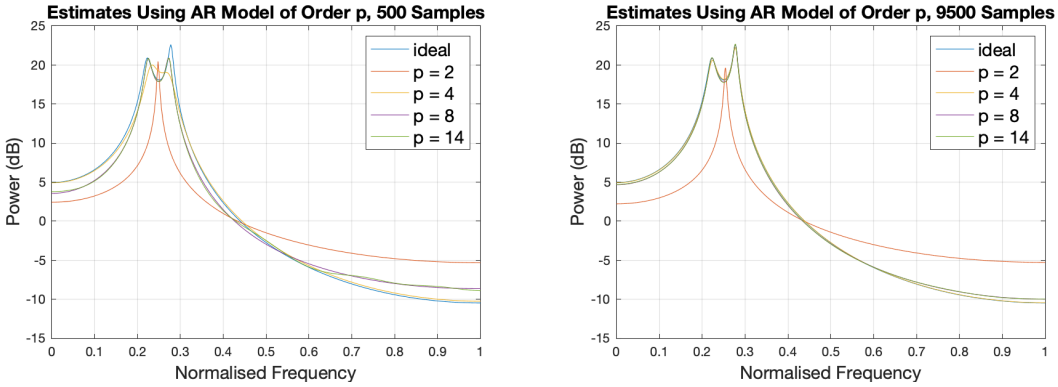

Figure 7: Correlogram estimates using AR models with different order and of different number of samples.

It is seen that at low order (p = 2), the predicted signal has a big error and is the worst out of all the degrees tested. It only has one peaks as oppose to having two peaks which is seen in the actual signal. This is because the AR model has a lower degree than the actual signal (which has a degree of four) and it is not powerful enough to accurately model the actual signal.

When the degree of the AR model increases, it becomes increasingly similar to the actual signal, giving a much lower error. This is because the model has a higher complexity and a higher degree of freedom to accurately model the actual data.

c. From Figure 7, it is seen that by using a higher number of samples, the performance of the AR model improved drastically. The AR model with p=4, N=9500 outperforms the AR model with p=14, N=500.

When increasing the order of the AR model, it is again seen that the AR model of order two only results in 1 peak and has a high error. This suggest that the AR model of two is not complex enough to model the actual signal and that its performance cannot be improved merely by increasing the number of samples. The AR model with order 14 on the other hand over overshot the actual signal. This suggests that it overfit the signal as the model is more complex than the actual signal. Hence it is concluded that to accurately model a signal using the AR model, we should try to match the complexity of the model to the complexity of the signal which we would like to model.

1.5 1.5 Real World Signals: Respiratory Sinus Arrhythmia from RR-Intervals

a. The standard periodogram and averaged periodogram of ECG data for three trials are plotted in Figure 8 and Figure 9:

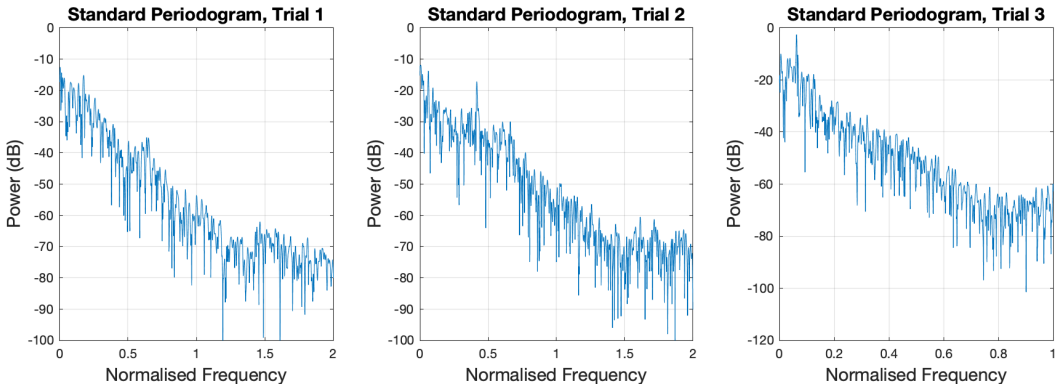

Figure 8: Standard periodogram of RRI data.

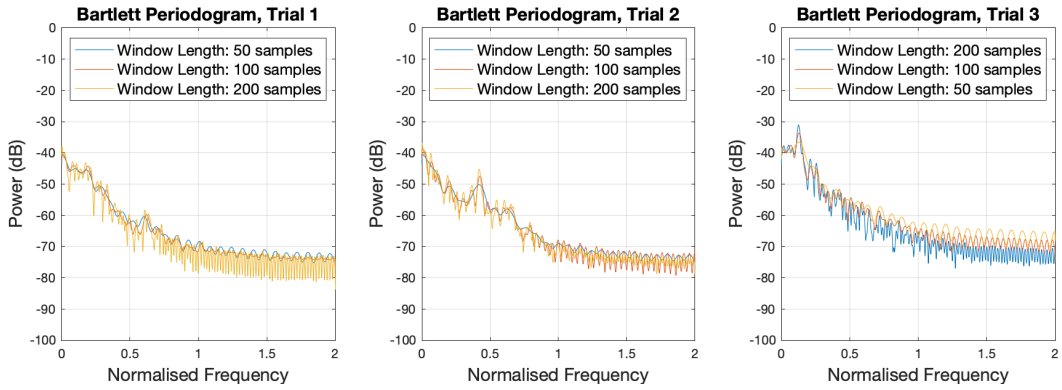

Figure 9: Periodogram of RRI data with different window length.

b. The three trials were conducted such that:

Trial	Breathing Rate
Trial 1	Unconstrained breathing, 20-45 breaths per minute (BPM)
Trial 2	Fast breathing, 50 BPM
Trial 3	Slow breathing, 15 BPM

Table 1: Breathing rate conditions

In Figure 9, it is seen that in Trial 2 and Trial 3 where the peak of the first harmonic can easily be distinguished at 0.4244Hz and 0.1216Hz respectively. In Trial 1 however, the peaks are not as distinct. This might be due to the inconsistency in the unconstrained breathing rate. However, a small peak can be seen at 0.1744Hz which will be used to deduce the breathing rate in Trial 1. The breathing rate in each trial can be deduced as:

Trial	Frequency of First Harmonic (Hz)	Breathing Rate (BPM)
Trial 1	0.1744	$2 \times 60 \times 0.1744 = 20.928 \approx 21$
Trial 2	0.4244	$2 \times 60 \times 0.4244 = 50.928 \approx 51$
Trial 3	0.1216	$2 \times 60 \times 0.1216 = 14.592 \approx 15$

Table 2: Properties of breathe signal.

c. The AR spectrum estimate for the RRI signals for the three trials are plotted in Figure 10.

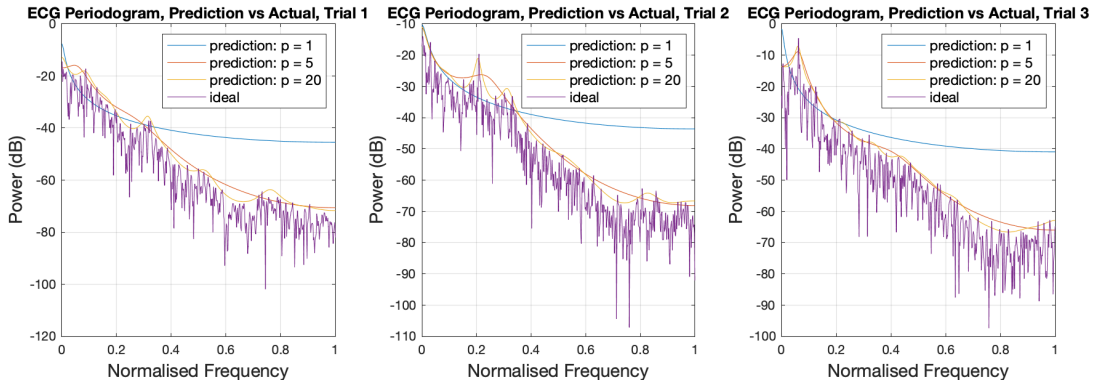

Figure 10: Prediction of periodogram of RRI data using AR model with different degrees of p.

For all of the trials, it is observed that the higher the order the better the AR model. By increasing the order of the AR model, the model managed to fit onto more harmonics while at the same time produce peaks closer to the frequency of the actual peaks. This is because increasing the degree of the model, it is provided with a larger degree of freedom to model the actual data. However, with an increase in the order, we also increase the computational complexity of the model; making it increasingly expensive and unpractical. Hence, a balance between computational complexity and model accuracy needs to be balanced.

1.6 1.6 Robust Regression

The singular values of $\mathbf{X}$ and $\mathbf{X}_{noise}$ are plotted in Figure 11:

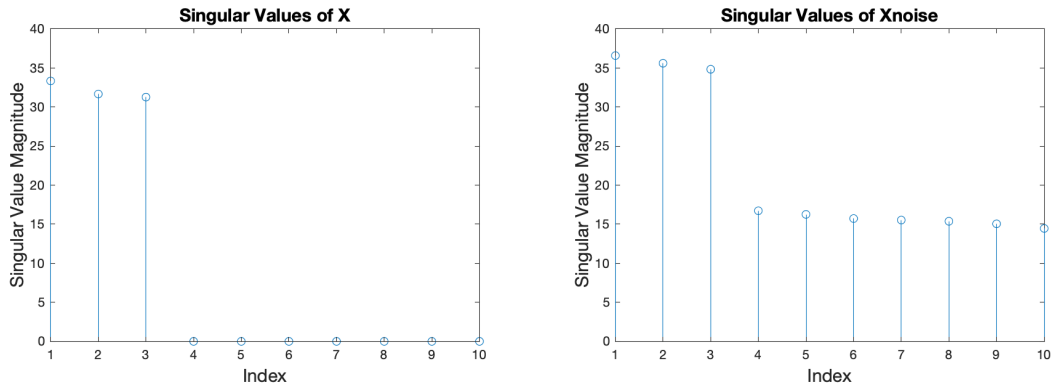

Figure 11: Singular Values of $\mathbf{X}$ and $\mathbf{X}_{noise}$.

From Figure 11, it is seen that $\mathbf{X}$ has three dominant singular values. Hence, it has a rank of three. It is observed that $\mathbf{X}_{noise}$ also has three dominant poles. The noise has slightly increased the dominant singular values of $\mathbf{X}$, as well as introduced new non-zero singular values. The squared error between singular values of $\mathbf{X}$ and $\mathbf{X}_{noise}$ is plotted in Figure 12.

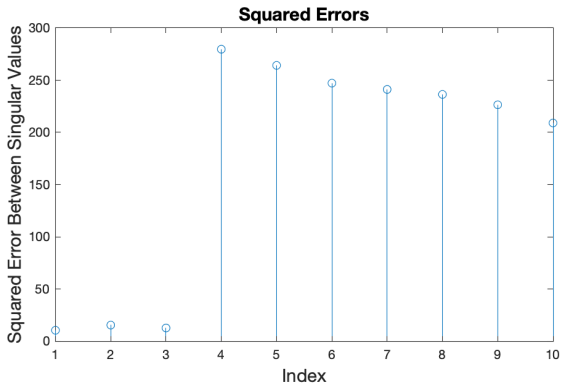

Figure 12: Squared error between singular values of $\mathbf{X}$ and $\mathbf{X}_{noise}$.

From Figure 12, it is again noted that the noise does indeed increase the three dominant singular values observed in $\mathbf{X}$. However, it is noted that the effect of noise is more prominent on the zero singular values. It would be difficult to identify the rank of the matrix $\mathbf{X}_{noise}$ if the noise increased the previously zero singular values to a point where it is relatively close to the dominant singular values.

b. the difference (error) between the variables (columns) of the noiseless input matrix, X, and those in the noise corrupted matrix Xnoise and denoised matrix $\hat{\mathbf{X}}_{noise}$ can be quantified using the Frobenius Norm of their differences. The Frobenius Norm is given by:

$$||A||_F \equiv \sqrt{\sum_{i}^{m}\sum_{j}^{n}|a_{ij}|^2}$$

The Frobenius Norm of the difference between the required matrices are given as:

Matrix	Frobenius Norm
$\|\|\mathbf{X} - \mathbf{X}_{noise}\|\|$	2435
$\|\|\mathbf{X} - \hat{\mathbf{X}}_{noise}\|\|$	733
$\|\|\hat{\mathbf{X}}_{noise} - \mathbf{X}_{noise}\|\|$	1704

It is observed that the difference between $\mathbf{X}$ and $\mathbf{X}_{noise}$ is larger than the difference between $\mathbf{X}$ and $\hat{\mathbf{X}}_{noise}$. This implies that the low rank approximation has successfully denoised $\mathbf{X}_{noise}$. Note that the sum $||\mathbf{X} - \hat{\mathbf{X}}_{noise}||_F^2$ and $||\hat{\mathbf{X}}_{noise} - \mathbf{X}_{noise}||_F^2$ is approximately $||\mathbf{X} - \mathbf{X}_{noise}||_F^2$.

c. The $\mathbf{B}_{OLS}$ and $\mathbf{B}_{PCR}$ are calculated and in turn are used to calculate $\mathbf{Y}_{OLS}$ and $\mathbf{Y}_{PCR}$. The error of $\mathbf{Y}_{OLS}$ and $\mathbf{Y}_{PCR}$ are again calculated using the Frobenius Norm and are given per below:

Matrix	Frobenius Norm
$\|\|\mathbf{Y} - \mathbf{Y}_{OLS}\|\|$	3552
$\|\|\mathbf{Y} - \hat{\mathbf{Y}}_{OCR}\|\|$	3577

It is seen that for the training set, the ordinary least squared scheme (OLS) has achieved a lower error as compared to the Principal Component Regression (PCR). However, the scheme should be tested on data which was not used to train the model in order to truly evaluate the performance of each schema. For the validation set, $\mathbf{Y}_{test}$ is used to test the performance of each schema and the results are given as:

Matrix	Frobenius Norm
$\|\|\mathbf{Y}_{test} - \mathbf{Y}_{test-OLS}\|\|$	2374
$\|\|\mathbf{Y}_{test} - \hat{\mathbf{Y}}_{test-PCR}\|\|$	2354

For a fair comparison, not that $\mathbf{B}_{OLS}$ and $\mathbf{B}_{PCR}$ are still calculated using $\mathbf{Y}$ and not $\mathbf{Y}_{test}$. It is observed that for the validation set, the PCR scheme performed better than the OLS scheme, giving a lower error. It is deduced that the OLS scheme has overfited the data giving a lower training error but higher validation error.

d. However, one single observation is insufficient to validate the performance of the two schemas. They are hence evaluated over 100 randomly generated test datasets. The performance of each of the schemas are given below:

Matrix	Frobenius Norm
$\|\|\mathbf{Y}_{test} - \mathbf{Y}_{test-OLS}\|\|$	2352
$\|\|\mathbf{Y}_{test} - \hat{\mathbf{Y}}_{test-PCR}\|\|$	2328

It is hence concluded that the PCR scheme outperforms the OLS schema, giving a lower average error.

2 Adaptive Signal Processing

2.1 The Least Mean Square (LMS) Algorithm

a. The correlation matrix, $\mathbf{R}_x$ is given as:

$$\begin{aligned}\mathbf{R}_x &= E\left\{\mathbf{x}(n)\mathbf{x}(n)^T\right\} \\ &= E\begin{bmatrix} x(n-1)x(n-1) & x(n-1)x(n-2) \\ x(n-2)x(n-1) & x(n-2)x(n-2)\end{bmatrix} \\ &= \begin{bmatrix} E\{x(n-1)x(n-1)\} & E\{x(n-1)x(n-2)\} \\ E\{x(n-2)x(n-1)\} & E\{x(n-2)x(n-2)\}\end{bmatrix}\end{aligned}$$

To simplify each of the elements in the matrix, we take the given equation for the AR(2) process:

$$x(n) = a_1 x(n-1) + a_2 x(n-2) + \eta(n)$$

multiplying both sides by $x(n-m)$:

$$x(n)x(n-m) = a_1 x(n-1)x(n-m) + a_2 x(n-2)x(n-m) + \eta(n)x(n-m)$$

Now take the expectations on both sides and let $E\{x(n-1)x(n-1)\} = \gamma(m)$:

$$\gamma(m) = E\{x(n)x(n-m)\} = a_1 E\{x(n-1)x(n-m)\} + a_2 E\{x(n-2)x(n-m)\} + E\{\eta(n)x(n-m)\}$$

Taking $m = 0, 1, 2$ in three separate equations:

$$\begin{aligned}\gamma(0) &= a_1 E\{x(n-1)x(n)\} + a_2 E\{x(n-2)x(n)\} + E\{\eta(n)x(n)\} \\ &= a_1\gamma(1) + a_2\gamma(2) + \sigma_n^2 \\ \gamma(1) &= a_1 E\{x(n-1)x(n-1)\} + a_2 E\{x(n-2)x(n-1)\} + E\{\eta(n)x(n-1)\} \\ &= a_1\gamma(0) + a_2\gamma(1) \\ \gamma(2) &= a_1 E\{x(n-1)x(n-2)\} + a_2 E\{x(n-2)x(n-2)\} + E\{\eta(n)x(n-2)\} \\ &= a_1\gamma(1) + a_2\gamma(0)\end{aligned}$$

Solving the three simultaneous equations with $a_1 = 0.1$, $a_2 = 0.8$ and $\sigma_n^2 = 0.25$ to give:

$$\gamma(0) = \frac{25}{27}$$
$$\gamma(1) = \frac{25}{54}$$
$$\gamma(2) = \frac{85}{108}$$

Going back to the equation for R_x:

$$\begin{aligned}\mathbf{R}_x &= \begin{bmatrix} E\{x(n-1)x(n-1)\} & E\{x(n-1)x(n-2)\} \\ E\{x(n-2)x(n-1)\} & E\{x(n-2)x(n-2)\} \end{bmatrix} \\ &= \begin{bmatrix} \gamma(0) & \gamma(1) \\ \gamma(1) & \gamma(0) \end{bmatrix} \\ &= \begin{bmatrix} \frac{25}{27} & \frac{25}{54} \\ \frac{25}{54} & \frac{25}{27} \end{bmatrix}\end{aligned}$$

To find the range of values of μ in which the LMS algorithm will converge, the eigenvalues of $\mathbf{R}_x$ have to satisfy the following inequality:

$$0 < \mu < \frac{2}{\lambda_{max}}$$

where λ is an eigenvalue of $\mathbf{R}_x$. If the condition is satisfied, the LMS algorithm will converge to the Wiener-Hopf solution. The eigenvalues of $\mathbf{R}_x$ are found to be $\frac{25}{54}$ and $\frac{25}{18}$. Hence, the LMS algorithm will converge if $0 < \mu < 1.44$.

b. The squared prediction error, $e^2(n)$, for both one trial and average across 100 realisations, are plotted in Figure 13:

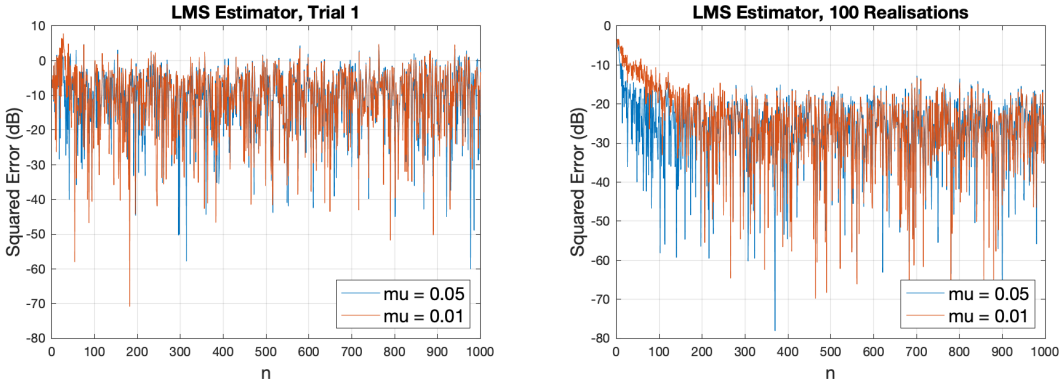

Figure 13: Squared prediction error, $e^2(n)$ of the LMS algorithm.

It is seen that the larger the μ, the faster the estimator converges. This is because a larger μ allows for a bigger step size and hence quicker descent to the steady state.

c. To study the steady state of the excess mean square error (EMSE) for different values of μ, we take the mean across time of the ensemble-averaged learning curves across 100 realisations. From Figure 13, it is observed that for both $\mu = 0.01$ and $\mu = 0.05$, the LMS algorithm has reached steady state starting from $n = 300$. Hence the mean of the steady state is calculated using all the values for $n > 300$. The mean of the steady state EMSE for both $\mu = 0.01$ and $\mu = 0.05$ are given Table 3. The theoretical misadjustment and the empirical misadjustments are also calculated to compared the results:

μ	**Steady State EMSE (dB)**	**Empirical Misadjustment**	**Theoretical Misadjustment**
0.01	0.0028	0.0112	0.093
0.05	0.133	0.452	0.0463

Table 3: Empirical misadjustment vs theoretical misadjustment for different values of μ

It is observed that the empirical misadjustments is slightly different from the empirical misadjustments. However, it is abundantly clear that a smaller μ gives a smaller steady state EMSE which in turn gives a smaller misadjustment.

d. The steady state values of the adaptive filter coefficients for the different step-sizes are plotted in Figure 14:

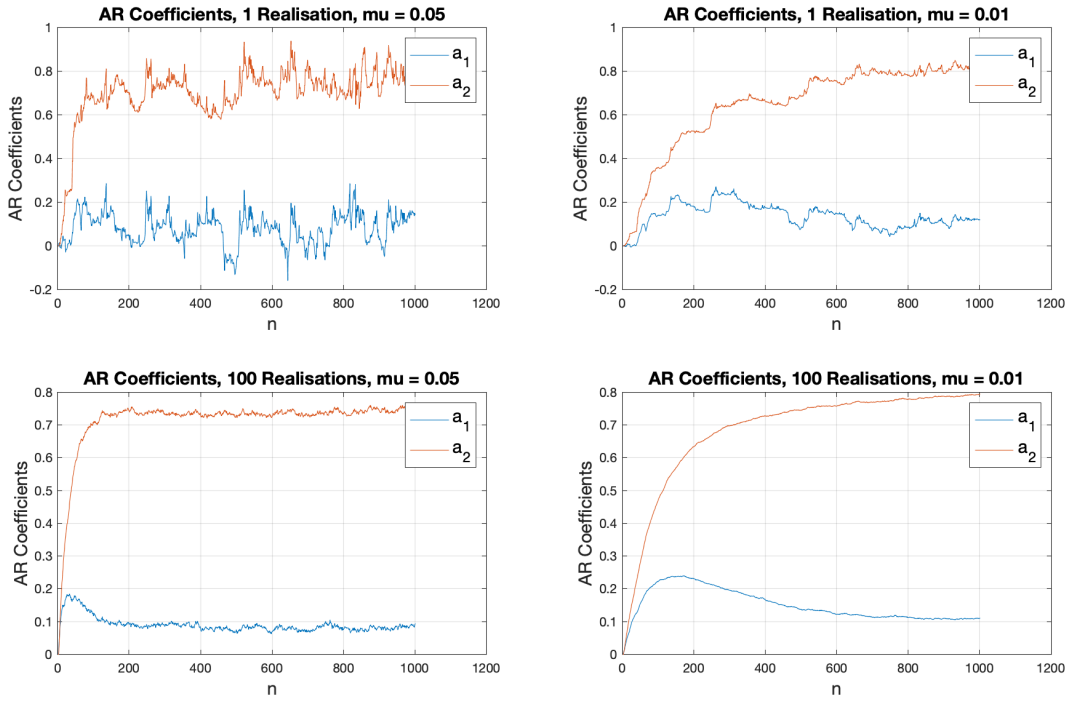

Figure 14: Evolution of the adaptive filter coefficients with different μ.

Again, we observe that the LMS algorithm converges to steady state faster when the μ is bigger. The steady state mean is calculated and are shown below in Figure 15 and also in Table 4:

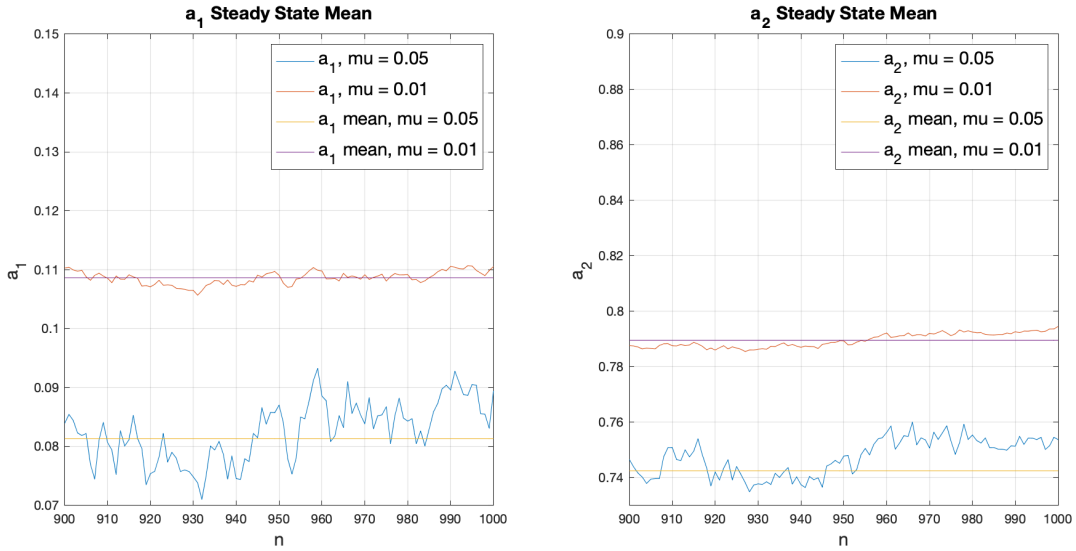

Figure 15: Steady state of the adaptive filter coefficients with different μ.

μ	**True Values**	**0.01**	**0.05**
Steady state a_1	0.1	0.1086	0.0813
Steady state a_2	0.8	0.7895	0.7425

Table 4: Steady state of the adaptive filter coefficients with different μ.

Note that the steady state mean is taken across $n > 900$ for both $\mu = 0.01$ and $\mu = 0.05$. It is evident that a when a smaller μ is used, the weights of the LMS algorithm converge closer to the true values.

e. To minimize $J(n)$, we differentiate $J(n)$ with respect to $\mathbf{w}$:

$$J(n) = \frac{1}{2}(e^2(n) + \gamma||\mathbf{w}(n)||_2^2)$$
$$\frac{\partial J}{\partial \mathbf{w}} = e(n)\frac{\partial e}{\partial \mathbf{w}} + \gamma \mathbf{w}(n)$$

Expressing the error, $e(n) = \mathbf{x}(n) - \mathbf{w}^T(n)\mathbf{x}(n)$:

$$\frac{\partial J}{\partial \mathbf{w}} = e(n)\frac{\partial}{\partial \mathbf{w}}(\mathbf{x}(n) - \mathbf{w}^T(n)\mathbf{x}(n)) + \gamma \mathbf{w}(n)$$
$$\frac{\partial J}{\partial \mathbf{w}} = e(n)(-\mathbf{x}(n)) + \gamma \mathbf{w}(n) \tag{8}$$

To use gradient descent to find the optimal w to minimise J, we turn to the following equation:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mu \nabla_w J(n)$$

substituting $\nabla_w J(n)$ with equation (8) gives:

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

which matches with the equation given in the handout.

f. Figure 16 shows the evolution of the AR coefficients for different μ and γ:

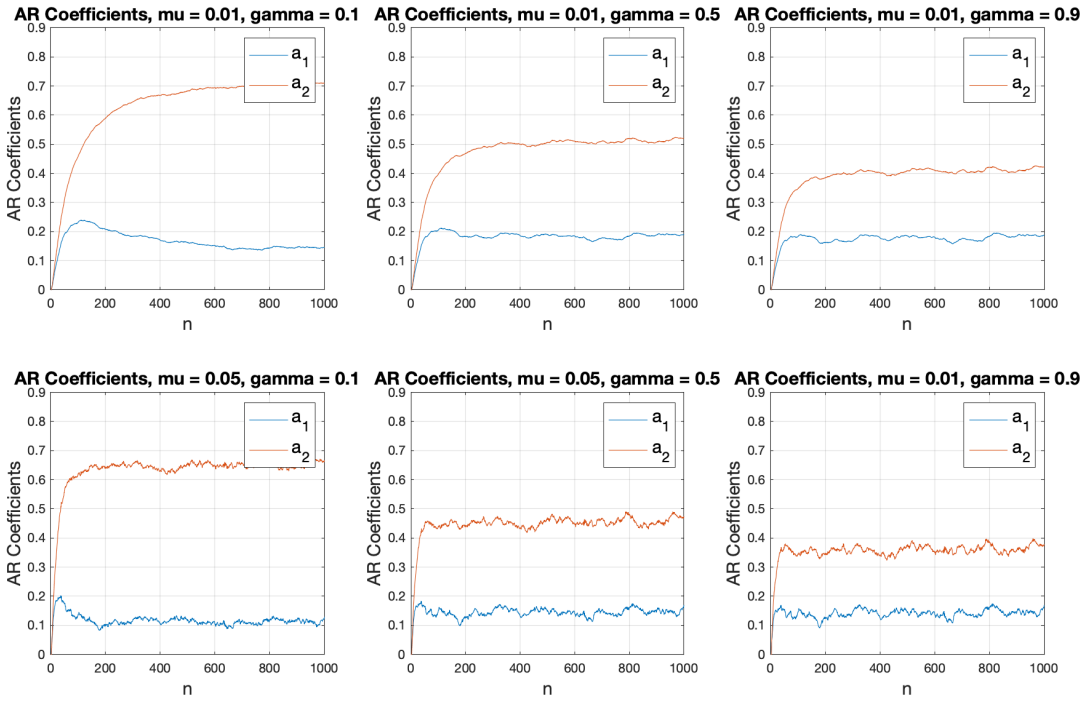

Figure 16: Evolution of the AR coefficients for different μ and γ.

It is observed that the steady-state values of the Leaky LMS algorithm does not converge to the true a_1 and a_2 values. In fact, the bigger the γ, the further the steady-state value is to the true value of the AR coefficients.

The LMS algorithm updates the coefficients in such a way that it minimises the squared error. This is equivalent to solving the linear equation:

$$\mathbf{R}_{xx}\mathbf{w}_{opt} = x$$

where $\mathbf{R}_{xx}$ is the autocorrelation matrix. It is discussed in 2.1.a that the LMS algorithm will converge to the Wiener-Hopf solution under certain bound conditions. The Leaky LMS algorithm is introduced when those conditions are not met or in other words, when $\mathbf{R}_{xx}$ is non-invertible. The Leaky LMS algorithm introduces a regularisation parameter, γ which relaxes the bound condition. However, the Leaky MLS solution does not converge to the Wiener-Hopf solution. This is because instead of minimising the squared error, the Leaky LMS algorithm tries to minimise a cost function which is slightly different cost function. Hence, the solution of the Leaky LMS algorithm converges to:

$$\lim_{k\to+\infty} E[\mathbf{w}_{leaky\_opt}] = (\mathbf{R} + \gamma\mathbf{I})^{-1}\mathbf{p}$$

Note that the Leaky LMS algorithm is identical to the LMS algorithm when $\gamma = 0$. In fact, as γ increases, the more the Leaky LMS algorithm is regularised. Hence the bigger the γ, the more the steady state values diverge from the true Wiener-Hopf solution (the true values of the AR coefficients).

2.2 Adaptive Step Sizes

a. The wight error (true coefficients - predicted coefficients) for different algorithms are plotted below:

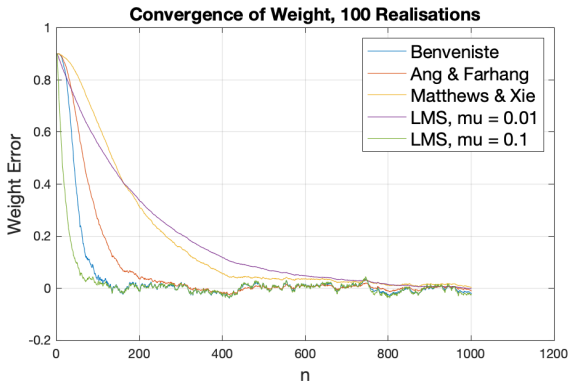

Figure 17: Predicted weight error of different algorithms.

The steady state error for each of the algorithm is given in Table 5. Note that the steady state value is the mean of the last 100 values for each of the algorithm.

Algorithm	**Steady State Error**
Benveniste	0.0102
Ang & Farhang	0.0088
Matthews & Xie	0.0075
LMS, $\mu = 0.1$	0.0112
LMS, $\mu = 0.01$	0.0099

Table 5: Steady state error of different algorithms (mean of last 100 values).

It is observed that all the algorithms converge to about the same steady state. However all of the algorithms exhibit different rate of convergence. The rate of convergence of the algorithms, arranged from fastest to slowest is given as: LMS ($\mu = 0.1$), Benveniste, Ang Farhang, Matthews and Xie, and LMS ($\mu = 0.1$).

b. Given the update equation of the normalised LMS (NLMS) algorithm:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{\beta}{\epsilon + ||\mathbf{x}(n)||^2} e(n)\mathbf{x}(n) \tag{9}$$

We need to prove that it is equivalent to the update equation:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e_p(n)\mathbf{x}(n) \tag{10}$$

based upon a *posteriori* error, e_p:

$$e_p(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n+1) \tag{11}$$

We start by substituting (10) into (11):

$$\begin{aligned} e_p(n) &= d(n) - \mathbf{x}^T(n)[\mathbf{w}(n) + \mu e_p(n)\mathbf{x}(n)] \\ &= d(n) - \mathbf{x}^T(n)\mathbf{w}(n) - \mu e_p(n)||\mathbf{x}(n)||^2 \\ &= \frac{d(n) - \mathbf{x}^T(n)\mathbf{w}(n)}{1 + \mu||\mathbf{x}(n)||^2} \\ &= \frac{e(n)}{1 + \mu||\mathbf{x}(n)||^2} \end{aligned}$$

Substituting this back into (10):

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) + \mu e_p(n)\mathbf{x}(n) \\ &= \mathbf{w}(n) + \mu \frac{e(n)}{1 + \mu||\mathbf{x}(n)||^2}\mathbf{x}(n) \\ &= \mathbf{w}(n) + \frac{1}{\frac{1}{\mu} + ||\mathbf{x}||^2} e(n)\mathbf{x}(n) \end{aligned}$$

From which we can deduce that $\epsilon = \frac{1}{\mu}$ and $\beta = 1$.

c. The evolution of the weight estimates for the GNGD algorithm and Benveniste's algorithm is plotted in Figure 18. Note that the plot is the mean taken from 100 realisations.

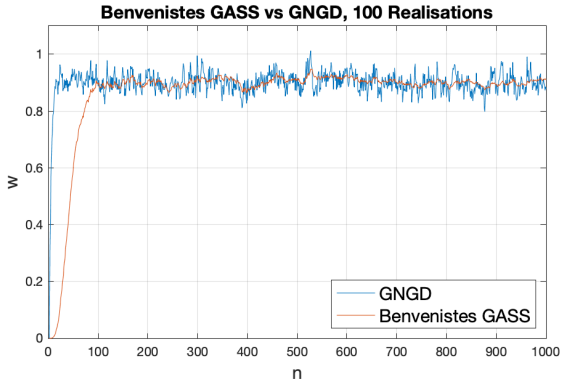

Figure 18: Evolution of the weight estimates for the GNGD algorithm and Benveniste's algorithm.

It is observed that the GNGD algorithm converges much faster as compared to Benveniste's algorithm. Both the algorithms converges to similar steady state values. The GNGD algorithm however exhibits high variance in its steady state value. The results are given below:

Algorithm	**Steady State Mean**	**Steady State Absolute Error**	**Steady State Variance**
Benveniste	0.9054	0.0054	0.000238
GNGD	0.9030	0.0030	0.0012

Table 6: GNGD algorithm vs Benveniste's algorithm.

Note that again, these statistics are calculated by considering the last 100 values. To calculate the computational complexity of the GNGD algorithm and Benveniste's algorithm, we consider the update equation for

each of the algorithm. We then break the update equation into its component sub-step. Note: the variable M represents the model order.

Step	Sub-step	Multiplications	Additions
$\phi(n)$	$\mu(n-1)\mathbf{x}(n-1)$	M	0
	$\mu(n-1)\mathbf{x}(n-1)\mathbf{x}^T(n-1)$	M^2	0
	$\mathbf{I}-\mu(n-1)\mathbf{x}(n-1)\mathbf{x}^T(n-1)$	0	M
	$[\mathbf{I}-\mu(n-1)\mathbf{x}(n-1)\mathbf{x}^T(n-1)]\phi(n-1)$	M^2	M^2-M
	$e(n-1)\mathbf{x}(n-1)$	M	0
	$[\mathbf{I}-\mu(n-1)\mathbf{x}(n-1)\mathbf{x}^T(n-1)]\phi(n-1)+e(n-1)\mathbf{x}(n-1)$	0	M
$\mu(n+1)$	$\mathbf{x}^T(n-1)\phi(n)$	M	M - 1
	$\rho e(n)$	1	0
	$\rho e(n)\mathbf{x}^T(n-1)\phi(n)$	1	0
	$\mu(n)+\rho e(n)\mathbf{x}^T(n-1)\phi(n)$	0	1
$\mu(n+1)$	$\mu(n)e(n)$	M	M - 1
	$\mu(n)e(n)\mathbf{x}(n)$	1	0
	$\mathbf{w}(n)+\mu(n)e(n)\mathbf{x}(n)$	1	0
Total		$O(2M^2+4M+3)$	$O(M^2+2M)$

Table 7: Benveniste's algorithm computational complexity.

Step	Sub-step	Multiplications	Additions
$\epsilon(n+1)$	$\mathbf{x}^T(n)\mathbf{x}^T(n-1)$	M	M - 1
	$e(n)e(n-1)$	1	0
	$e(n)e(n-1)\mathbf{x}^T(n)\mathbf{x}^T(n-1)$	1	M
	$\|\|\mathbf{x}(n-1)\|\|^2$	M	M-1
	$(\epsilon(n-1)+\|\|\mathbf{x}(n-1)\|\|^2)^2$	1	1
	$\rho\mu\frac{e(n)e(n-1)\mathbf{x}^T(n)\mathbf{x}^T(n-1)}{(\epsilon(n-1)+\|\|\mathbf{x}(n-1)\|\|^2)^2}$	3	0
	$\epsilon(n)+\rho\mu\frac{e(n)e(n-1)\mathbf{x}^T(n)\mathbf{x}^T(n-1)}{(\epsilon(n-1)+\|\|\mathbf{x}(n-1)\|\|^2)^2}$	0	1
$\mu(n+1)$	$\|\|\mathbf{x}(n)\|\|^2$	M	M - 1
	$\frac{\beta}{e(n)+\|\|\mathbf{x}(n)\|\|^2}$	1	1
	$\frac{\beta}{e(n)+\|\|\mathbf{x}(n)\|\|^2}e(n)$	1	0
	$\frac{\beta}{e(n)+\|\|\mathbf{x}(n)\|\|^2}e(n)\mathbf{x}(n)$	M	M - 1
	$\mathbf{w}(n)+\frac{\beta}{e(n)+\|\|\mathbf{x}(n)\|\|^2}e(n)\mathbf{x}(n)$	0	M
Total		$O(4M+8)$	$O(5M-3)$

Table 8: GNGD algorithm computational complexity.

From this analysis, it is observed that the GNGD algorithm has linear complexity while the Benveniste algorithm has quadratic complexity. Hence we conclude that the GNGD algorithm is less computationally expensive to run when compared to the Benveniste algorithm.

2.3 Adaptive Noise Cancellation

a. To find the optimal Δ, we start from the Mean Square Error:

$$\begin{aligned} E\left\{(s(n)-\hat{x})^2\right\} &= E\left\{(x(n)+\eta(n)-\hat{x})^2\right\} \\ &= E\left\{\eta^2(n)\right\}+E\left\{(x(n)-\hat{x})^2\right\}+2E\left\{\eta(n)\left(x(n)-\hat{x}\right)\right\} \end{aligned} \tag{12}$$

From (12), we see that the error term can be broken down into three components. In the ideal scenario $E\left\{(s(n)-\hat{x})^2\right\} = E\left\{\eta^2(n)\right\}$ while the second and third term equals zero. Hence to minimise the mean-squared error, we can only manipulate the second and third term. However, by manipulating Δ, we can only vary the third term, $2E\left\{\eta(n)\left(x(n)-\hat{x}\right)\right\}$. As such, we need to choose a value of Δ to minimise the third term which in turn minimise the overall error term.

Given the definitions of $\eta(n)$ and $x(n)$:

$$\eta(n) = v(n) + 0.5v(n-2)$$
$$\hat{x}(n) = \mathbf{w}^T\mathbf{u}(n)$$

Note that for the third term, x(n) and $\eta(n)$ are uncorrelated, hence the term reduces to:

$$E\left\{(\hat{x})\right\}$$

we can expand the third term from (12) as:

$$\begin{aligned} E\left\{\eta(n)\hat{x}\right\} &= E\left\{\left(v(n) + 0.5v(n-2)\right)\mathbf{w}^T\mathbf{u}(n)\right\} \\ &= E\left\{\left(v(n) + 0.5v(n-2)\right)\left(\sum_{i=0}^{m} w_i s(n-\Delta-i)\right)\right\} \\ &= E\left\{\left(v(n) + 0.5v(n-2)\right)\left(\sum_{i=0}^{m} w_i (x(n-\Delta-i) + \eta(n-\Delta-i))\right)\right\} \end{aligned}$$

Since $x(n)$ and $v(n)$ are uncorrelated, we can write:

$$\begin{aligned} E\left\{\eta(n)\hat{x}\right\} &= E\left\{\eta(n)\left(v(n) + 0.5v(n-2)\right)\left(\sum_{i=0}^{m} w_i \eta(n-\Delta-i)\right)\right\} \\ &= E\left\{\left(v(n) + 0.5v(n-2)\right)\left(\sum_{i=0}^{m} w_i (v(n-\Delta-i) + 0.5v(n-\Delta-i-2))\right)\right\} \end{aligned}$$

Hence, we can observe that for all values of $\Delta > 2$, the expectations reduces to adding the expectations of white noise samples at different time indexes, which are uncorrelated. To prove this, we apply the ALU algorithm to a sine-wave corrupted with noise, η. In Figure 19, It is seen that the filtered signal with $\Delta > 2$ are cleaner than of $\Delta \leq 2$. This is further confirmed from the plot in Figure 20 where we see the reduction in noise is far more significant when $\Delta > 2$. Furthermore, it is also observed that there is a significant decrease in the Mean Prediction Squared Error (MPSE) when Δ is increased from 2 to 3. All of these results corroborate the theoretical results.

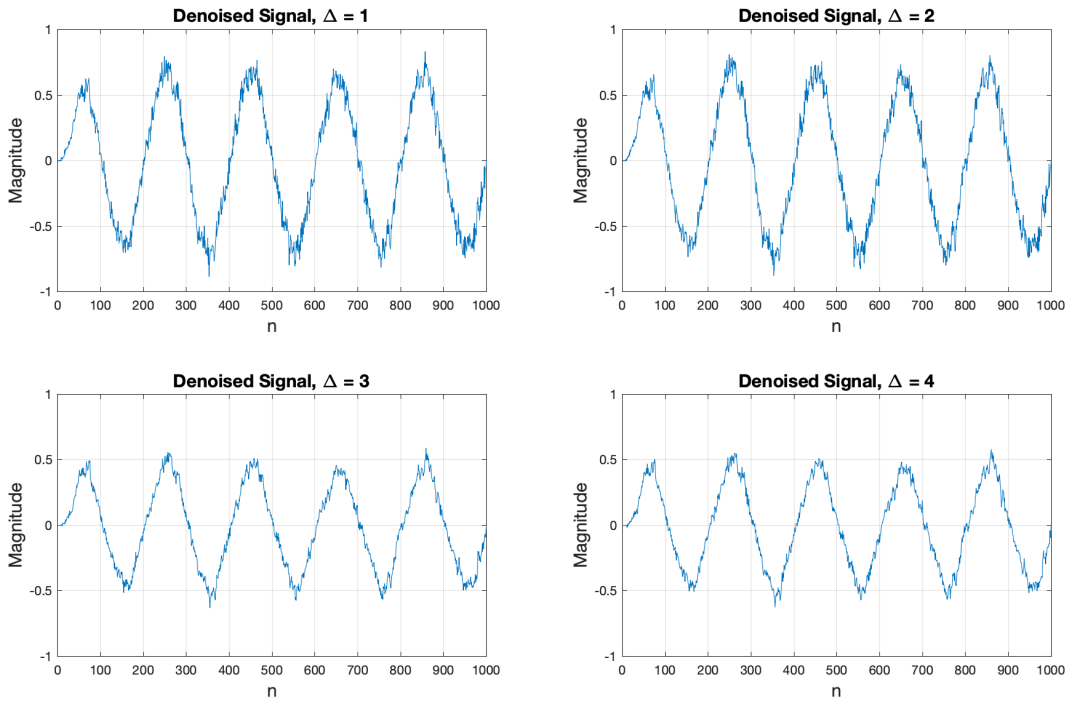

Figure 19: Average of 100 realisations of denoised signal for different Δ values.

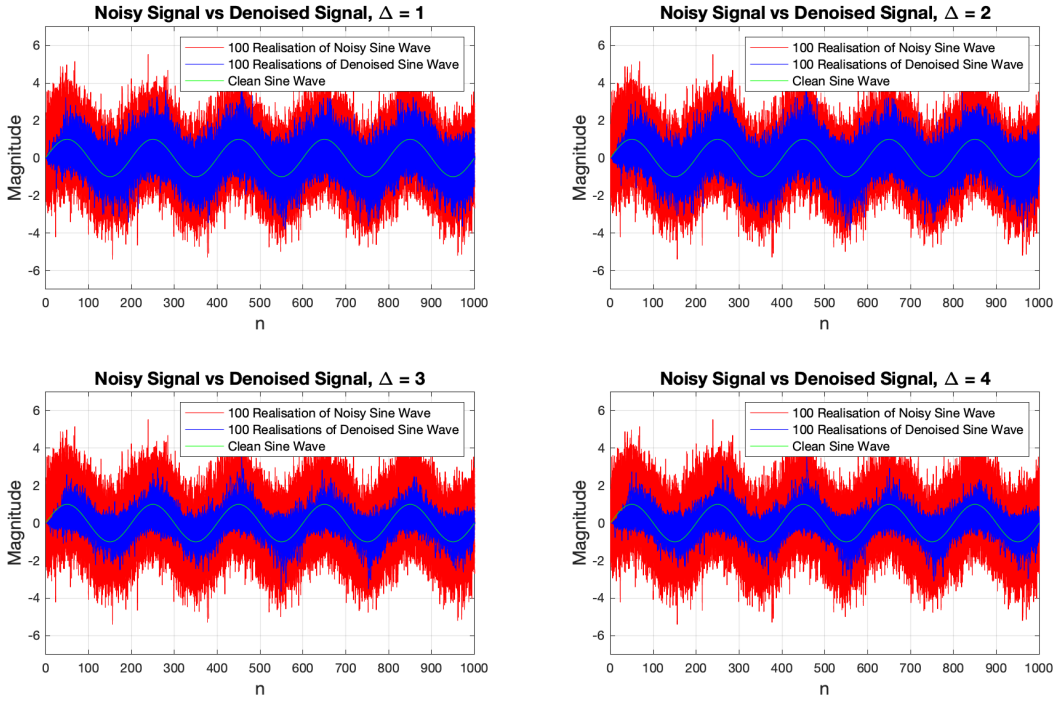

Figure 20: Reduction in noise using the ALE algorithm for different Δ values.

Δ	MPSE
1	0.4361
2	0.4320
3	0.3133
4	0.3127

Table 9: Mean Prediction Squared Error (MPSE) of Adaptive Line Enhancer (ALE) for different Δ values.

b. Figure 21 shows the effect of Δ on the MPSE. It is observed that after a steep decrease in the MPSE, any increase in Δ would increase the MPSE. To further understand this, we observe the denoised signal vs the clean signal in Figure 22. Increasing the value of Δ will increase the delay between the clean signal, $x(n)$ and the denoised signal, $\hat{x}(n)$ which in turn increases the MPSE.

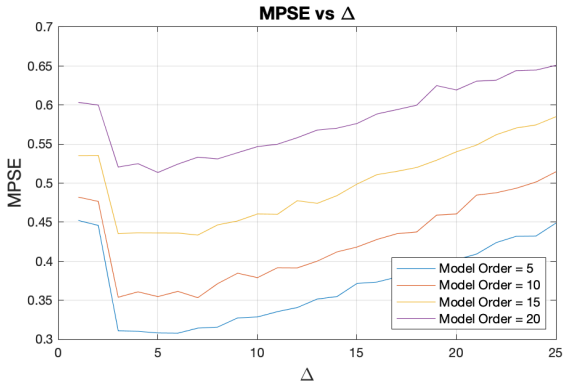

Figure 21: Effects of different Δ on MPSE.

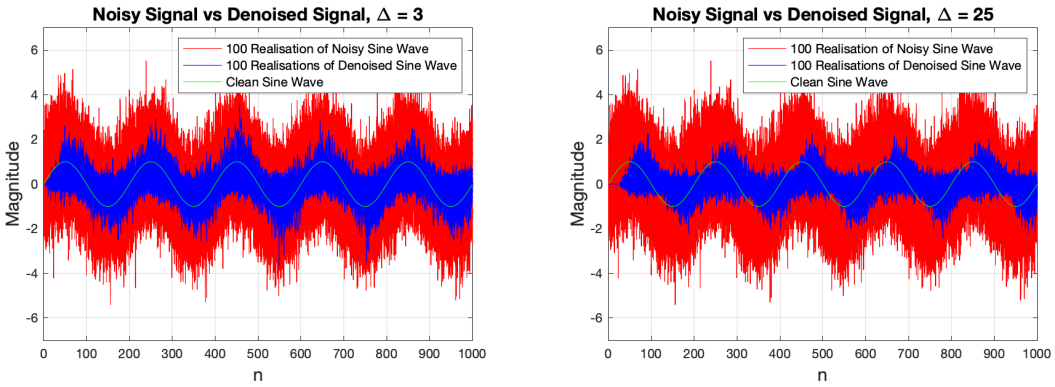

Figure 22: Reduction in noise using the ALE algorithm for different Δ values.

In Figure 23 we observe the effects of model order on the MPSE. Generally, a higher model order would result in a better estimate of the clean signal. However, we observe that this is not the case for the ALE algorithm. This is because since we are deriving the solution numerically, factors such as convergence, stability and computational complexity should also be taken into account. Increasing the model order in this case, while keeping the μ constant, results in an increase in the MPSE.

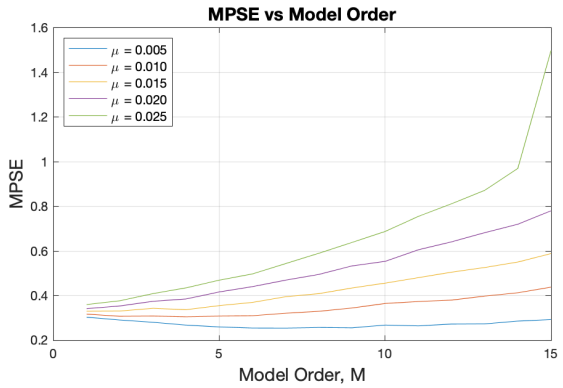

Figure 23: Effects of Model Order on MPSE.

c. In Figure 24 we observe that the Adaptive Noise Cancellation (ANC) algorithm performs significantly better than the Adaptive Linear Enhancer algorithm at denoising the noisy signal. However in Figure 25, notice that the ANC algorithm takes some time to converge to the clean signal hence for small values of n, the ALE algorithm outperforms the ANC algorithm.

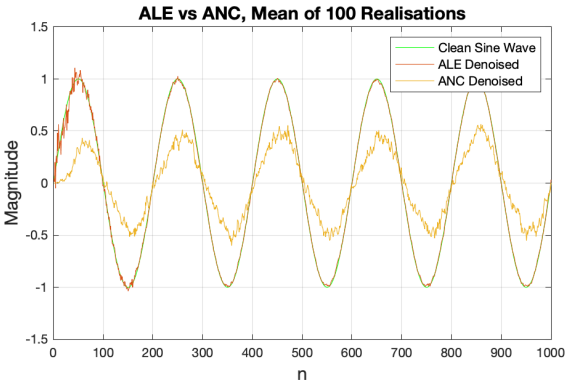

Figure 24: Adaptive Linear Enhancer (ALE) vs Adaptive Noise Cancellation (ANC).

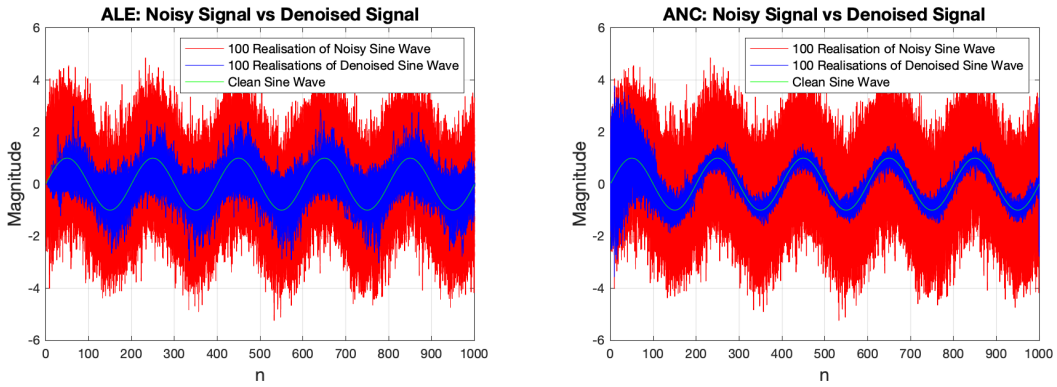

Figure 25: Time-frequency plot: Adaptive Linear Enhancer (ALE) vs Adaptive Noise Cancellation (ANC).

d. The spectrogram of the EEG Data is plotted in Figure 26. We observe that there is indeed a strong 50Hz component in the signal.

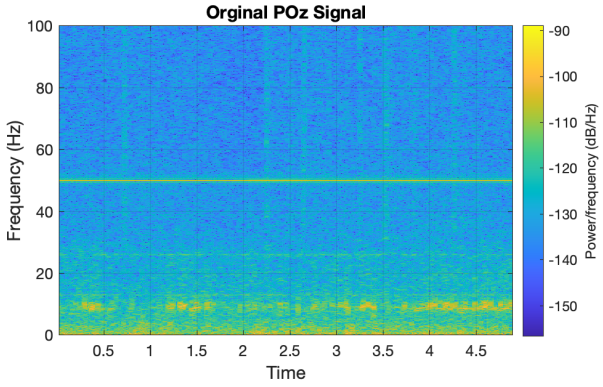

Figure 26: Spectrogram of EEG Data.

To remove the 50Hz component, we employ the ANC algorithm. Figure 22 shows the effects of using different order of the ANC algorithm on the EEG signal. It is observed that the high the order of the ANC algorithm, the more effective it is at removing the 50Hz component in the signal. However, a higher order also introduced artefacts. With M = 25 and $\mu = 0.01$, the ANC algorithm is suppressing signal in the entire range of 40Hz to 60Hz.

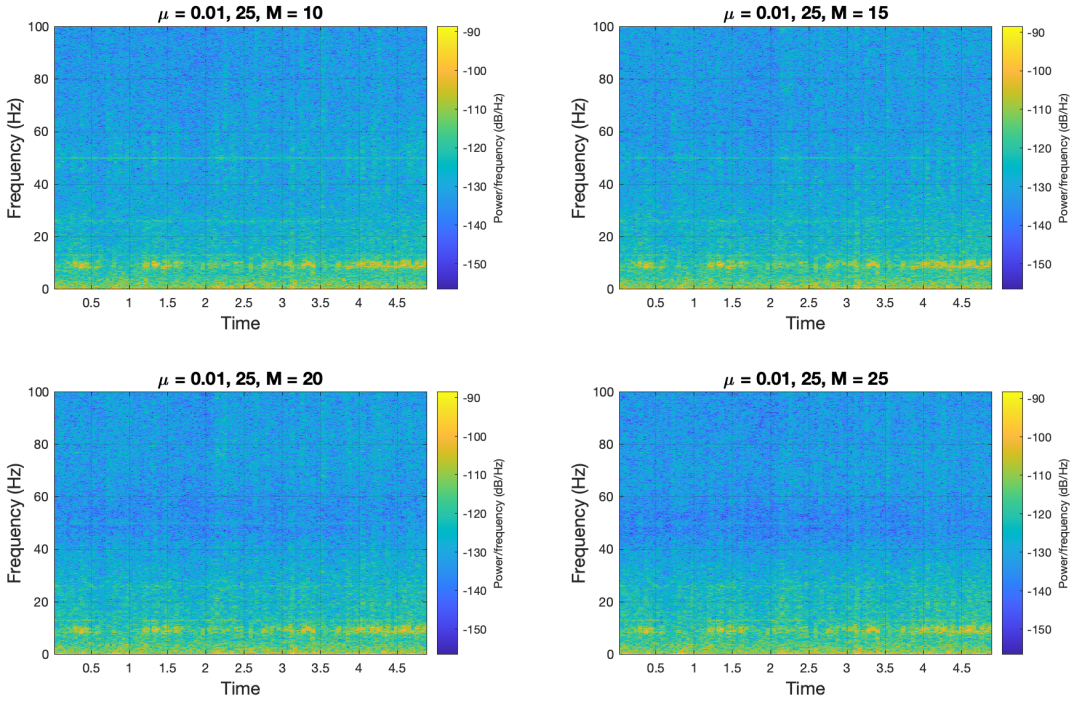

Figure 27: Effects of using different order of the ANC algorithm on the EEG data.

To counter this problem, we tweak the value of μ. The effects of using a different value of μ on the EEG data is given below, It is seen that using a smaller value of μ yields better results. This conforms with our previous results regarding μ i.e. the steady state variance of the algorithm will be smaller when the μ is small.

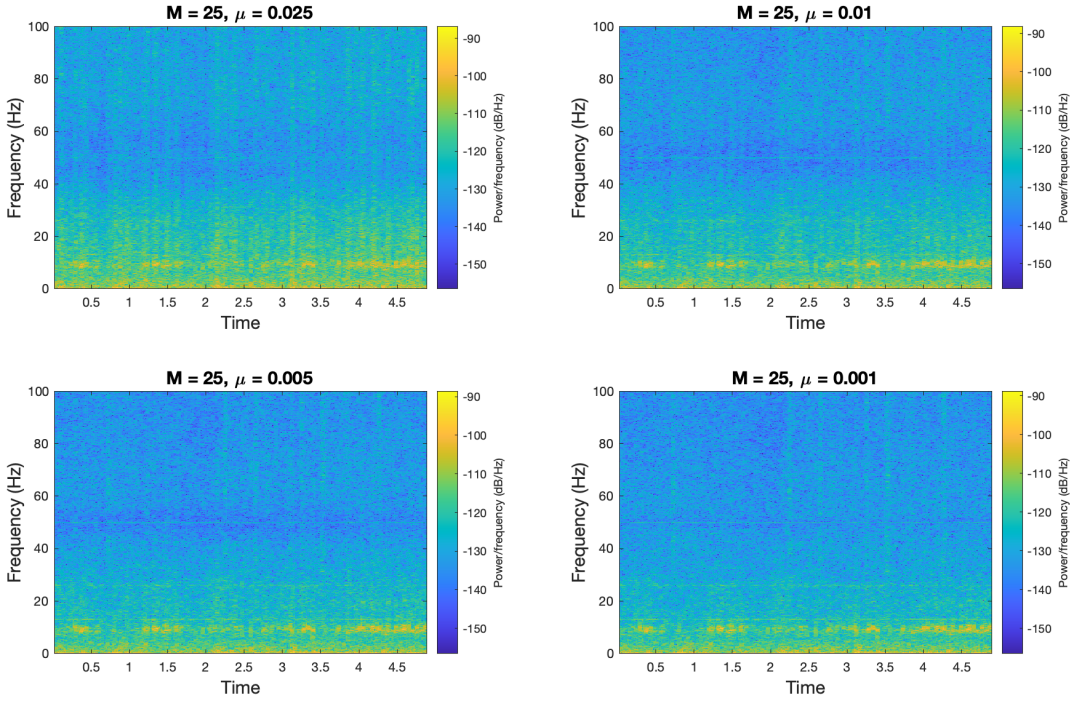

Figure 28: Effects of using different μ in the ANC algorithm on the EEG data.

Hence from the empirical analysis, we choose $\mu = 0.001$ and M = 25. To have a closer analysis of the results we use the standard periodogram from 1.2.b. The periodogram of the original EEG signal and also of the denoised EEG signal is plotted in Figure 29. It is observed that the two periodograms have very similar

magnitude at all frequencies besides 50Hz. At 50Hz there is a strong suppression in the denoise signal which is desirable. This is further confirmed with the periodogram of the error signal (peridogram of original EEG signal - peridogram of denoised EEG signal). It is clear that there is a strong supression at the 50Hz frequency while the rest of the frequencies are basically unaffected.

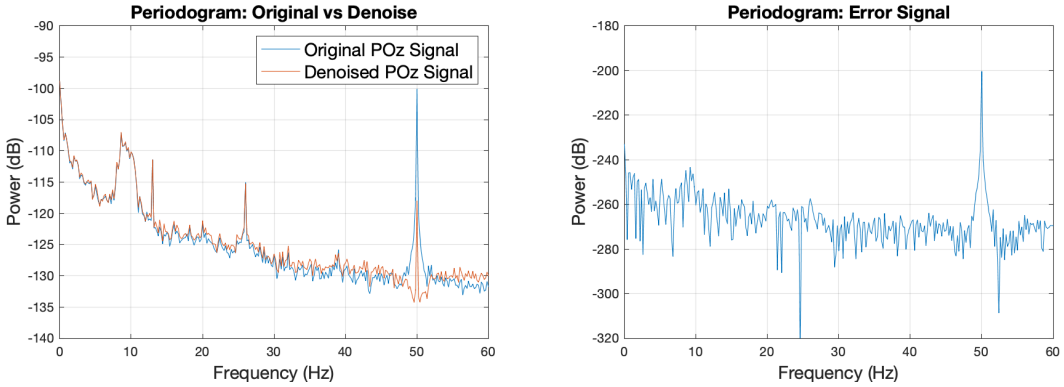

Figure 29: Periodogram of original EEG signal vs periodogram of denoised EEG data (left), periodogram of error signal (right).

3 Widely Linear Filtering and Adaptive Spectrum Estimation

3.1 Complex LMS and Widely Linear Modelling

a. From Figure 30, it is observed that the Widely Linear Moving Average (WLMA) process is non-circular. As a result the Complex Least Mean Sqaured (CLMS) algorithm is not guaranteed to capture the complete second-order statistical relationship between the input and the output as it is only catered for circular data. The Augmented Complex Least Mean Squared (ACLMS) algorithm however would provide a much better performance as it has a higher degree of freedom to fully exploit the second order statistics in the data. This is shown in Figure 31 where the steady state error of the ACLMS algorithm is far lower than of the CLMS algorithm. The results are further summarised in Table 10.

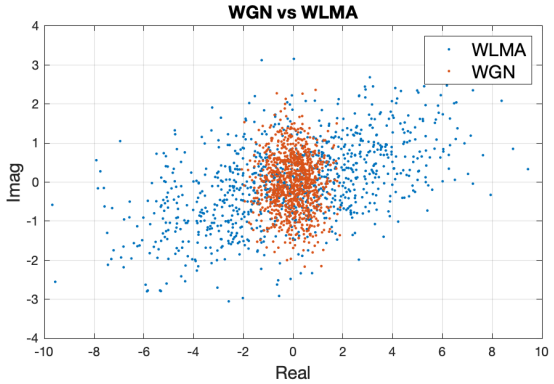

Figure 30: Distribution of White Gaussian Noise (WGN) and Widely Linear Moving Average (WLMA).

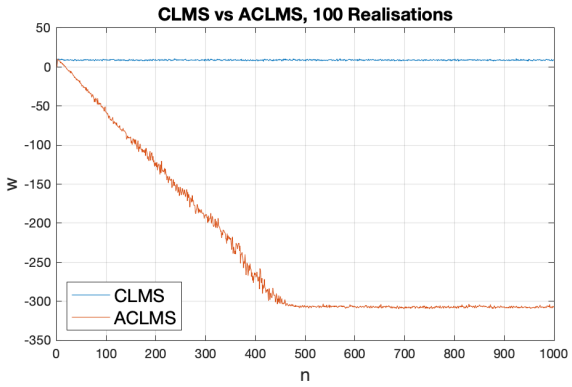

Figure 31: The learning curve, $10log|e(n)|^2$ of ACLMS vs CLMS.

Algorithm	Steady State Error (dB)
CLMS	8.8496
ACLMS	-307.2672

Table 10: Steady state error, CLMS vs ACLMS (Mean of last 500 readings).

b. A useful metric used to measure circularity is the distance between the origin and the mean of the data. From Figure 32, it is observed that the mean of High Wind is furthest away from the origin while the mean of Low Wind is closest to the origin. We hence deduce that Low Wind has the highest circularity, followed by Medium Wind and High Wind. This is further proven according to the calculated circularity coefficient given in Table 11. Note that the lower the circularity coefficient, the higher the circularity of the data.

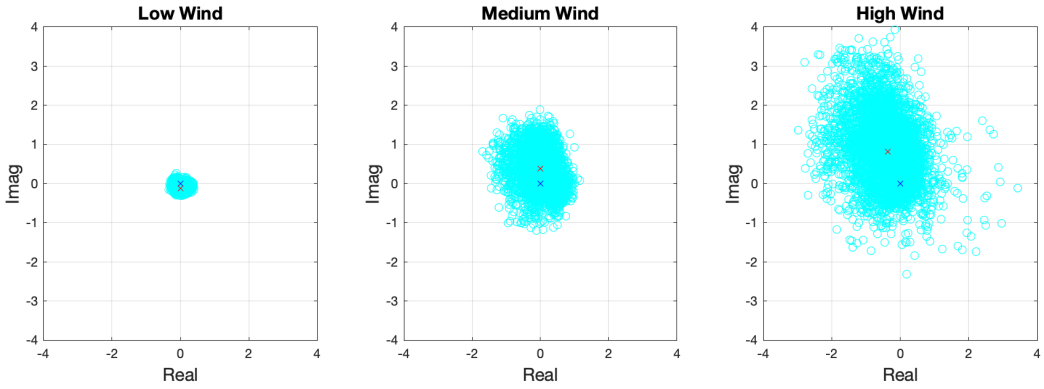

Figure 32: The learning curve, $10log|e(n)|^2$ of ACLMS vs CLMS.

Wind Speed	Circularity Coefficient
Low Wind	0.159
Medium Wind	0.4542
High Wind	0.624

Table 11: Circularity coefficient of different wind speeds.

From Figure 33, we observe Mean Prediction Squared Error (MPSE) of CLMS vs ACLMS against the model order. It is observed that for a fixed μ, the MPSE is higher when the circularity of the data is low. This is because non-circularity in the data introduces extra complexity to the data, making it harder to model. Note also that the MPSE of ACLMS always outperforms that of CLMS. This is due to the fact that ACLMS has a higher degree of freedom to model the non-circular data and that the CLMS is only suitable to model circular data.

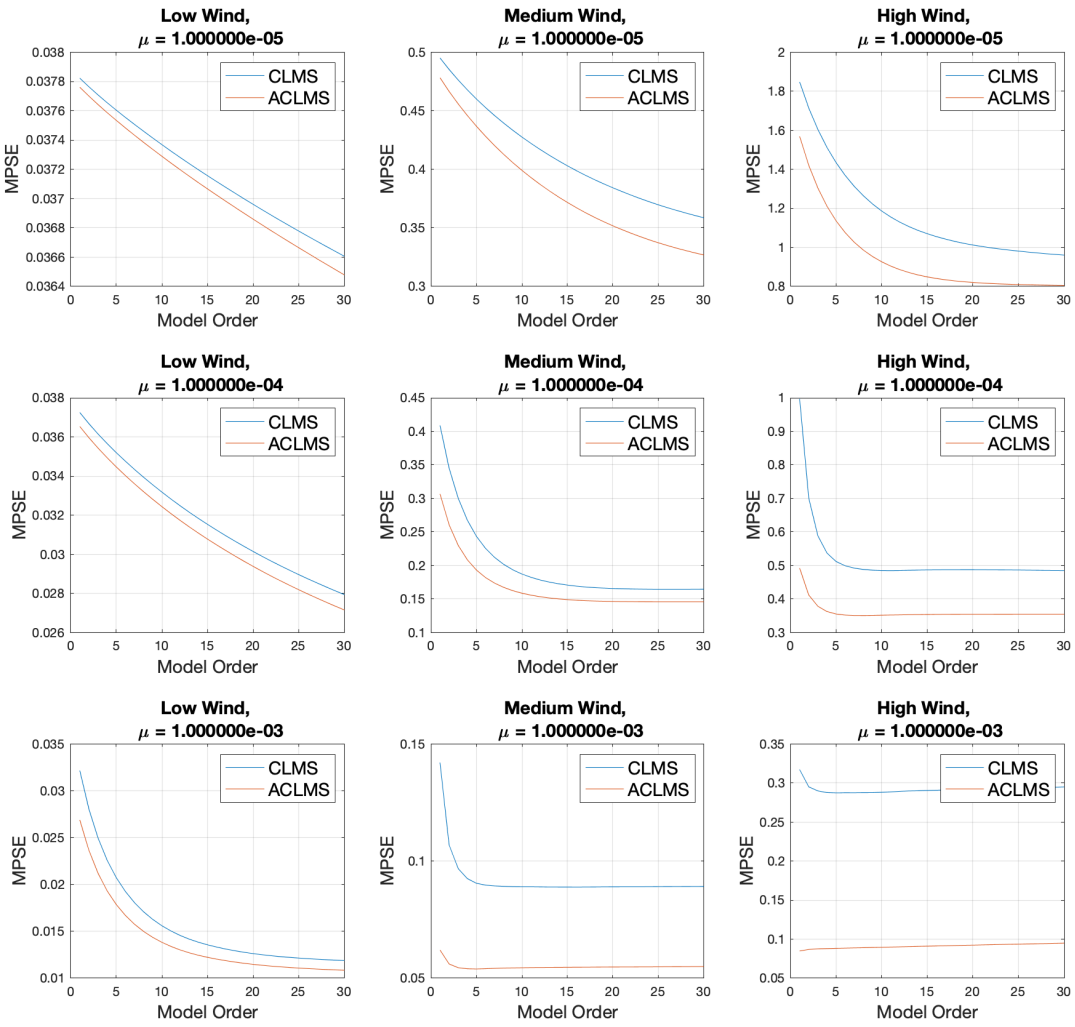

Figure 33: Mean Prediction Squared Error (MPSE) against model order for CLMS vs ACLMS for different μ.

c. There are two criteria where a 3-phase signal have to satisfy for it to be balanced:

- The magnitude of the voltages in all three phases has to be the same.
- The phase difference each of the different phases needs to be exactly 120.

These two conditions thus give us two possible conditions where the signals can be unbalanced. Figure 34 shows the circular diagrams of a balanced signal, a set of unbalanced signals (magnitude) and another set of unbalanced signals (phase). A circular system is obviously circular in shape while an unbalanced system will form ovals. Hence circular systems are rotational invariant while unbalanced systems are not.

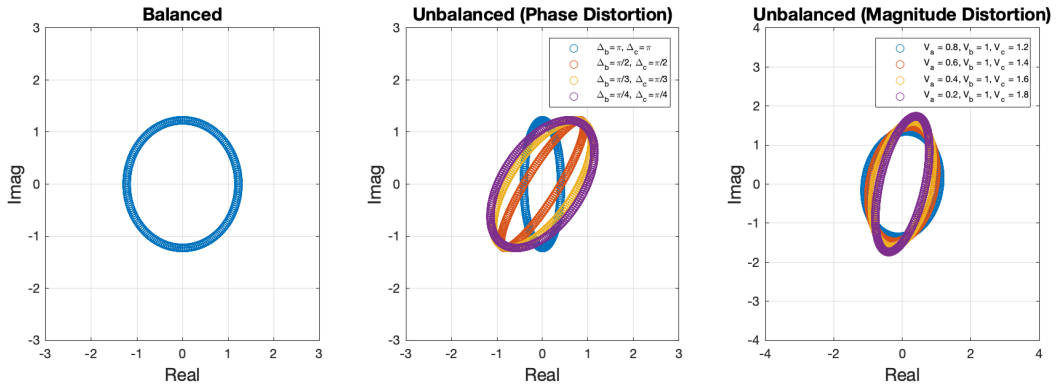

Figure 34: Circular diagrams of a balanced signal, a set of unbalanced signals (magnitude) and a set of unbalanced signals (phase).

d. Given that a strictly linear autoregressive models of order one has the form:

$$v(n+1) = h^*(n)v(n) \tag{13}$$

For a balanced system, we can express the voltage as:

$$v(n) = \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_0}{f_s} n + \phi)} \tag{14}$$

Substituting v(n) from (14) into (13):

$$\begin{aligned}
\sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_0}{f_s}(n+1)+\phi)} &= h^*(n)\sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_0}{f_s} n+\phi)} \\
e^{j(2\pi \frac{f_0}{f_s})} &= h^*(n) \\
&= Re\{h(n)\} - Im\{h(n)\} \\
&= |h(n)| e^{-j\left(arctan \frac{Im(h(n))}{Re(h(n))}\right)}
\end{aligned}$$

Equating the phase of the left and right side and rearranging gives us the required proof for the balanced complex voltage:

$$\begin{aligned}
2\pi \frac{f_0}{f_s} &= -arctan\left(\frac{Im\{h(n)\}}{Re\{h(n)\}}\right) \\
f_0 &= -\frac{f_s}{2\pi} arctan\left(\frac{Im\{h(n)\}}{Re\{h(n)\}}\right) \\
f_0 &= \frac{f_s}{2\pi} arctan\left(\frac{Re\{h(n)\}}{Im\{h(n)\}}\right)
\end{aligned}$$

For the unbalanced voltage, we start from the widely linear autoregressive model of order one:

$$v(n+1) = h^*(n)v(n) + g^*(n)v^*(n) \tag{15}$$

Given the Clarke Transform:

$$v(n) = A(n)e^{j(2\pi \frac{f_0}{f_s} n+\phi)} + B(n)e^{-j(2\pi \frac{f_0}{f_s} n+\phi)} \tag{16}$$

Substituting (16) into (15):

$$\begin{aligned}
&A(n+1)e^{j(2\pi \frac{f_0}{f_s}(n+1)+\phi)} + B(n+1)e^{-j(2\pi \frac{f_0}{f_s}(n+1)+\phi)} \\
&= h^*(n)A(n)e^{j(2\pi \frac{f_0}{f_s} n+\phi)} + h^*(n)B(n)e^{-j(2\pi \frac{f_0}{f_s} n+\phi)} \\
&+ g^*(n)A^*(n)e^{-j(2\pi \frac{f_0}{f_s} n+\phi)} + g^*(n)B^*(n)e^{j(2\pi \frac{f_0}{f_s} n+\phi)}
\end{aligned}$$

Then we gather the same exponential terms together, equate them and rearranging them, giving two separate equations:

$$\begin{aligned} A(n+1)e^{j(2\pi\frac{f_0}{f_s}(n+1)+\phi)} &= h^*(n)A(n)e^{j(2\pi\frac{f_0}{f_s}n+\phi)} + g^*(n)B^*(n)e^{j(2\pi\frac{f_0}{f_s}n+\phi)} \\ &= \left(h^*(n)A(n) + g^*(n)B^*(n)\right)e^{j(2\pi\frac{f_0}{f_s}n+\phi)} \end{aligned}$$

$$\begin{aligned} B(n+1)e^{j(2\pi\frac{f_0}{f_s}(n+1)+\phi)} &= h^*(n)B(n)e^{-j(2\pi\frac{f_0}{f_s}n+\phi)} + g^*(n)A^*(n)e^{-j(2\pi\frac{f_0}{f_s}n+\phi)} \\ &= \left(h^*(n)B(n) + g^*(n)A^*(n)\right)e^{-j(2\pi\frac{f_0}{f_s}n+\phi)} \end{aligned}$$

We then make the assumption that $A(n+1) \approx A(n)$ and $B(n+1) \approx B(n)$, we can further simplify the above equations into:

$$e^{j(2\pi\frac{f_0}{f_s})} = \frac{h^*(n)A(n) + g^*(n)B^*(n)}{A(n+1)} \approx h^*(n) + g^*(n)\frac{B^*(n)}{A(n)} \tag{17}$$

$$e^{-j(2\pi\frac{f_0}{f_s})} = \frac{h^*(n)B(n) + g^*(n)A^*(n)}{B(n+1)} \approx h^*(n) + g^*(n)\frac{A^*(n)}{B(n)} \tag{18}$$

Now we take the conjugate of (18) and equate it to (17):

$$h^*(n) + g^*(n)\frac{B^*(n)}{A(n)} = h(n) + g(n)\frac{A(n)}{B^*(n)}$$

We then perform a substitution of $Y = \frac{B^*(n)}{A(n)}$ to obtain a quadratic equation in Y:

$$g^*(n)Y^2 + (h^*(n) - h(n))Y + g(n) = 0$$

To solve this, we use the quadratic formula which gives us:

$$\begin{aligned} Y &= \frac{-h^*(n) - h(n) \pm \sqrt{(h^*(n) - h(n))^2 - 4g^*(n)g(n)}}{2g^*(n)} \\ &= \frac{2Im\{h(n)\}j \pm \sqrt{-4Im^2\{h(n)\} + 4|g(n)|^2}}{2g^*(n)} \\ &= j\frac{Im\{h(n)\} \pm \sqrt{Im^2\{h(n)\} - |g(n)|^2}}{g^*(n)} \end{aligned}$$

Substituting this back into equation (17):

$$\begin{aligned} e^{j(2\pi\frac{f_0}{f_s})} &= h^*(n) + g^*(n)j\frac{Im\{h(n)\} \pm \sqrt{Im^2\{h(n)\} - |g(n)|^2}}{g^*(n)} \\ &= h^*(n) + Im\{h(n)\}j \pm j\sqrt{Im^2\{h(n)\} - |g(n)|^2} \\ &= Re\{h(n)\} \pm j\sqrt{Im^2\{h(n)\} - |g(n)|^2} \end{aligned}$$

Since $f_s > f_0 > 0$, we take the positive solution. Note that M is an arbitrary constant:

$$\begin{aligned} e^{j(2\pi\frac{f_0}{f_s})} &= Re\{h(n)\} \pm j\sqrt{Im^2\{h(n)\} - |g(n)|^2} \\ &= |M|e^{j\left(arctan\left(\frac{\sqrt{Im^2\{h(n)\} - |g(n)|^2)}}{Re\{h(n)\}}\right)\right)} \end{aligned}$$

Equating the phase on the left-hand side and the right-hand side, and we arrive at the required proof after some rearranging:

$$\begin{aligned} 2\pi\frac{f_0}{f_s} &= arctan\left(\frac{\sqrt{Im^2\{h(n)\} - |g(n)|^2}}{Re\{h(n)\}}\right) \\ f_0 &= \frac{f_s}{2\pi}arctan\left(\frac{\sqrt{Im^2\{h(n)\} - |g(n)|^2}}{Re\{h(n)\}}\right) \end{aligned}$$

e. The signal generated has a fundamental frequency, $f_0 = 50Hz$ which matches the operating frequency of power transmission in the UK. The evolution of the frequency estimate using the CLMS and ACLMS algorithm is plotted in Figure 35 for the balanced system, and the two types of unbalanced systems. It is observed that for the balanced system, both the ACLMS and CLMS arrived to similar steady state values which are accurate. However, the CLMS algorithm converged to the steady state value faster than the ACLMS algorithm. This is to be expected as the ACLMS algorthm has a higher degree of freedom and hence more parameters for the algorithm to optimise.

For the unbalanced system on the other hand, the ACLMS algorithm converged to a more accurate steady state value than the CLMS algorithm. This is again expected as the CLMS algorithm is only designed to model balanced systems.

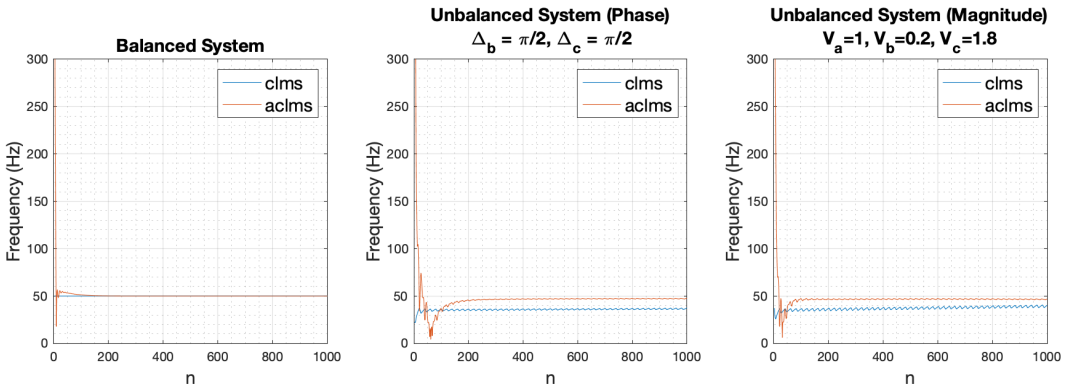

Figure 35: CLMS vs ACLMS for balanced and unbalanced systems.

In Figure 36, we observed the effects of the magnitude of distortion on the performance of the two algorithms. It is again evident that the magnitude of distortion has less effect on the ACLMS algorithm as it still converges close to the true values. In contrast, the steady state value of the CLMS algorithm gets increasingly further away from the true value as the magnitude of the distortion increases.

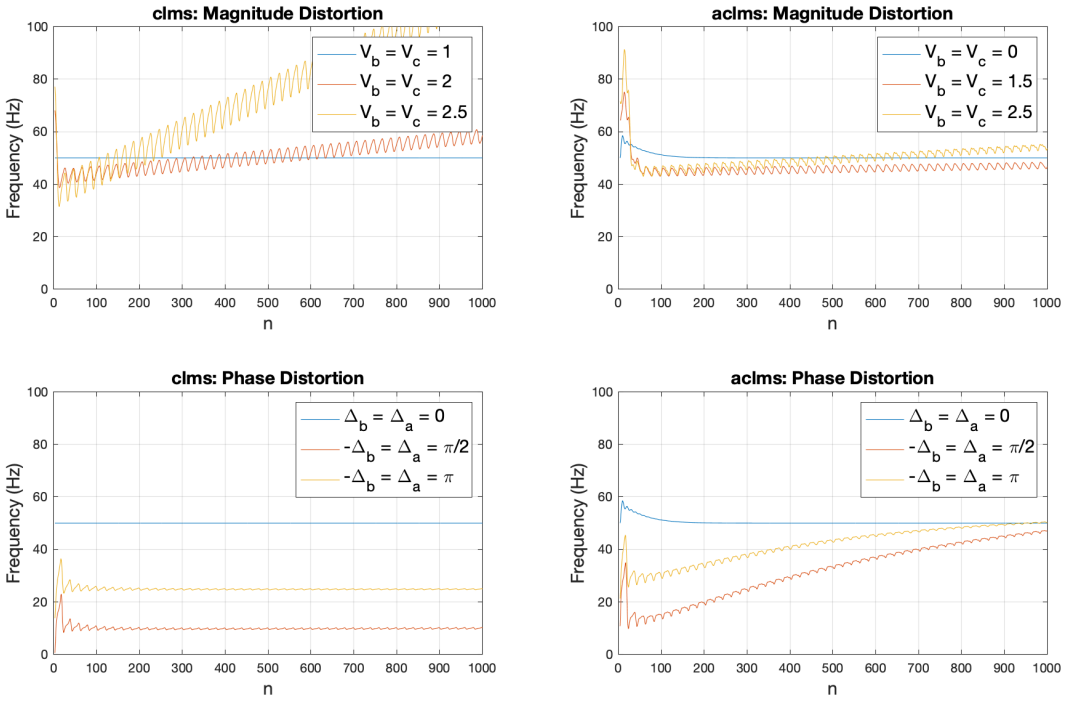

Figure 36: Steady state of CLMS and ACLMS for different magnitudes of distortion.

3.2 Adaptive AR Model Based Time-Frequency Estimation

Figure 37 shows the change in the fundamental frequency of the Frequency Modulated (FM) signal. It is observed that the signal is clearly non-stationary. There are also sudden changes in frequencies at $n = 500$ and $n = 1000$ which can be especially difficult to model. Figure 37 also shows the frequency spectrum of the FM signal modeled as AR processes of different order. It is observed that the frequency spectrum, even of higher model order, failed to accurately model the FM signal.

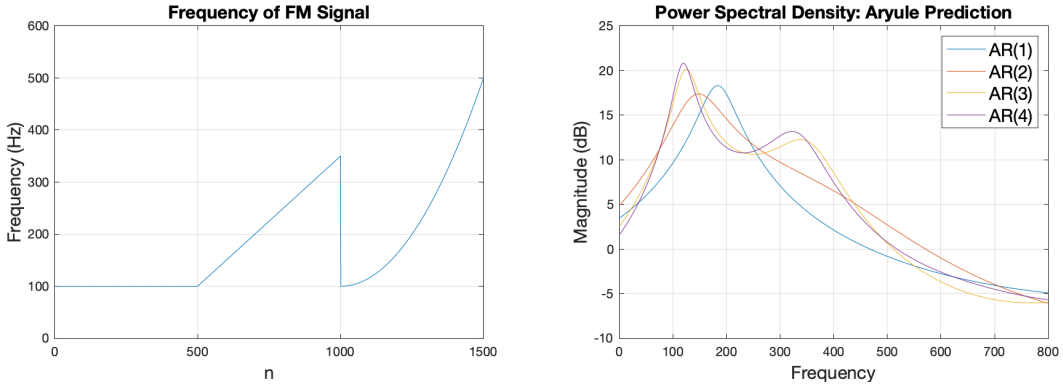

Figure 37: Change in fundamental frequency of the FM signal (left); AR frequency spectral representation of the FM signal.

To overcome this, we can block the overall signal into different components: constant, linear and exponential, and perform spectral analysis separately on each of those blocks. This is plotted in Figure 38. It is observed that the by blocking the FM signal into separate components, the AR spectral representation is able to better model the FM signal. AR(1) was able to model the constant frequency signal accurately. However, for the linearly increasing frequency signal and also the exponentially increasing frequency signal, an AR(20) model is required to accurately model the signal.

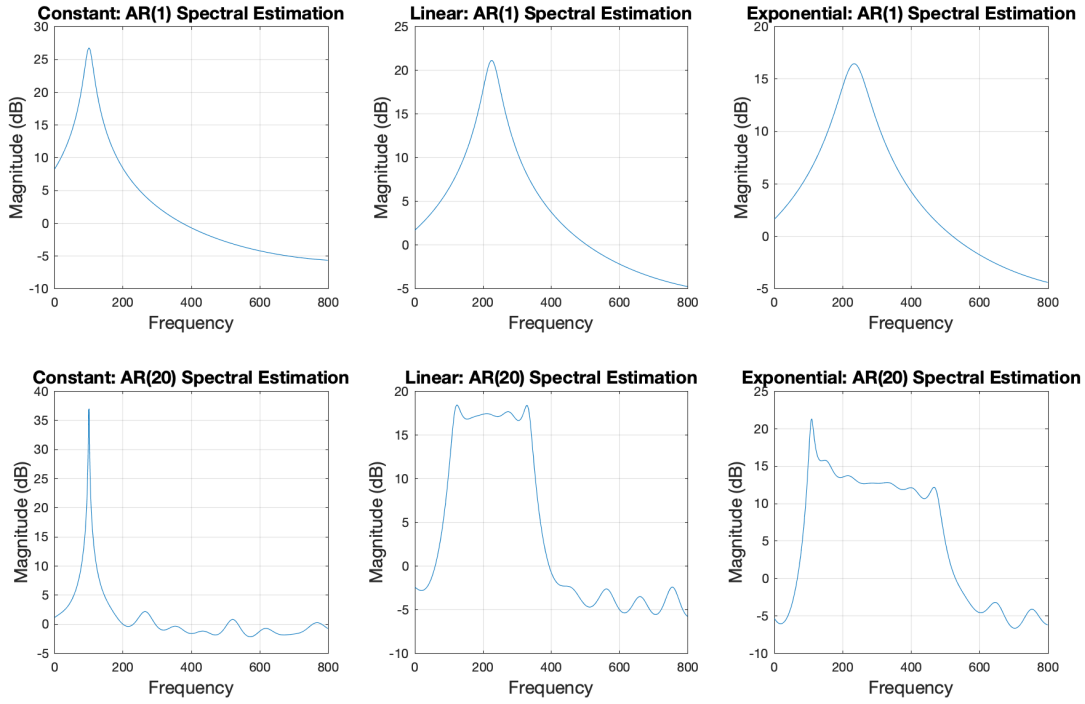

Figure 38: Frequency spectrum of the blocked FM signal modeled as AR processes of different order.

b. Although we are able to model the signal when we block them into different components, this method is impractical in real life. Besides that, this strategy also fails to capture the suddent change in frequency of the FM signal. To overcome these problems, we apply the CLMS algorithm. The CLMS algorithm is able to

infer the value of the AR coefficient in an adaptive manner, unlike the spectral representation method which considers the entire dataset all at once. Furthermore, the CLMS algorithm is able to capture the non-stationary in the data. Figure 39 shows the time-frequency estimate using the CLMS algorithm using different values of μ.

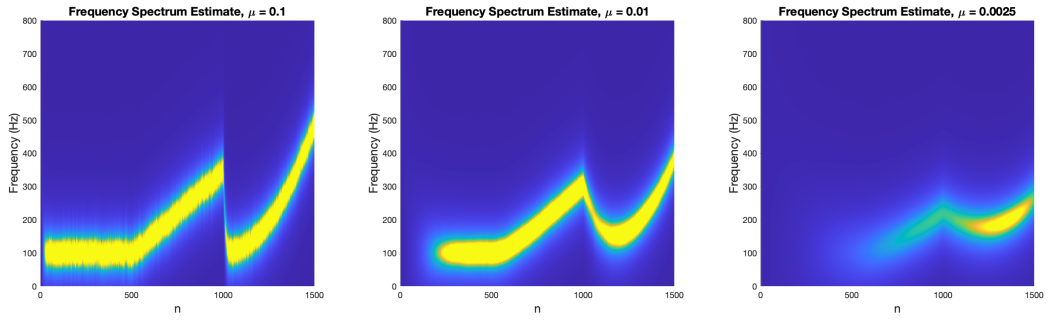

Figure 39: Evolution of the time-frequency estimate using CLMS with different values of μ.

As previously discussed, a bigger μ would result faster convergence and a bigger variance of the steady state value. In this case, the variance can be seen in the thickness of the yellow lines i.e. the thcker the lines, the bigger the variance. It is clear that the lines are the thickest when $\mu = 0.1$ and is the thinnest when $\mu = 0.0025$. For the rate of convergence, the frequency estimate reaches convergence at the values of n in which the yellow line starts. The frequency estimate converges at about $n = 30$ for $\mu = 0.1$, at about $n = 250$ for $\mu = 0.01$ and about $n = 700$ for $\mu = 0.1$. These results corresponds to our previous discussion regarding the relationship between μ, the variance of the steady state value and also the rate of convergence of the algorithm.

3.3 A Real Time Spectrum Analyser Using Least Mean Square

a. To find the least sqaures (LS) solution for the problem, we start by expanding the left-hand side of the cost function:

$$\begin{aligned} ||\mathbf{y} - \hat{\mathbf{y}}||^2 &= (\mathbf{y} - \hat{\mathbf{y}})^H (\mathbf{y} - \hat{\mathbf{y}}) \\ &= (\mathbf{y} - \mathbf{F}\mathbf{w})^H (\mathbf{y} - \mathbf{F}\mathbf{w}) \\ &= \mathbf{y}^H\mathbf{y} - \mathbf{w}^H\mathbf{F}^H\mathbf{y} - \mathbf{y}^H\mathbf{F}\mathbf{w} + \mathbf{w}^w(\mathbf{F}^H\mathbf{F})\mathbf{w} \end{aligned}$$

We then differentiate the above equation with respect to $\mathbf{w}$ to look for its minimum point:

$$\frac{\partial ||\mathbf{y} - \hat{\mathbf{y}}||^2}{\partial \mathbf{w}} = -\mathbf{F}^H\mathbf{y} - \mathbf{y}^H\mathbf{F} + \mathbf{w}^H(\mathbf{F}^H\mathbf{F} + \mathbf{F}^H\mathbf{F})$$

Finally, setting the differential to zero and rearranging the variables brings us to the required proof:

$$\begin{aligned} -\mathbf{F}^H\mathbf{y} - \mathbf{y}^H\mathbf{F} + \mathbf{w}^H(\mathbf{F}^H\mathbf{F} + \mathbf{F}^H\mathbf{F}) &= \mathbf{0} \\ 2\mathbf{w}^H(\mathbf{F}^H\mathbf{F}) &= 2\mathbf{y}^H\mathbf{F} \\ \mathbf{F}^H\mathbf{F}\mathbf{w} &= \mathbf{F}\mathbf{y} \\ \mathbf{w} &= \mathbf{F}^H\mathbf{F}^{-1}\mathbf{F}\mathbf{y} \end{aligned}$$

The formula for the Inverse Discrete Fourier Transform (DFT) of the signal x is given as:

$$\hat{x} = \frac{1}{N}\sum_{k=0}^{N-1} X(K) e^{\frac{j2\pi nk}{N}}$$

which in matrix form can be expressed as:

$$\begin{bmatrix} \hat{x}(0) \\ \hat{x}(1) \\ \vdots \\ \hat{x}(N-1) \end{bmatrix} = \begin{bmatrix} W_{0,0} & W_{1,0} & \cdots & W_{0,N-1} \\ W_{1,0} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ W_{N-1,0} & \cdots & \cdots & W_{N-1,N-1} \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ \vdots \\ X(N-1) \end{bmatrix}$$

in which can be compactly expressed as:

$$\hat{\mathbf{x}} = \mathbf{W}\mathbf{X} \tag{19}$$

Comparing this to the matrix notation of the Least Squares matrix notation:

$$\hat{\mathbf{y}} = \mathbf{F}\mathbf{w}$$

It is clear that the two processes have the same general form in which we can deduce that the Inverse Fourier Transform will a similar solution:

$$\mathbf{X} = (\mathbf{W}^H\mathbf{W})^{-1}\mathbf{W}\mathbf{x} \tag{20}$$

Hence, we deduce that the Fourier coefficients, $\mathbf{X}$ would also return a solution where by the squared error between $\mathbf{x}$ and $\hat{\mathbf{x}}$ i.e. $||\hat{\mathbf{x}} - \mathbf{x}||^2$ is minimised.

b. $\hat{\mathbf{x}} = \mathbf{W}\mathbf{X}$ is an approximation of the signal $\mathbf{x}$. We infer that $\mathbf{X}$ is the projection of $\mathbf{x}$ onto the column space of $\mathbf{W}$ and that this projection is based off a least square cost function. Comparing this with Fourier Transform, note that DFT only requires a finite number sets of basis vectors. This is because the DFT assumes the signal to be periodic with a period of N.

c. Figure 40 shows the time-frequency estimation of the FM signal using the DFT-CLMS algorithm. It is observed that the estimation forms a "shadow" of what is observed in the actual time-frequency graph in Figure 37 that is once a frequency is detected, the frequency will remain in the spectrum throughout for any later values of n. This is due to the fact that the weights of all the frequencies are updated recursively and they are not reinitialised at every n. For the past frequency weights to be removed, an error term will need to be propagated to the later n. With the error term, the weights will converge back to zero once the frequency has faded.

To combat this problem, we employ the Leaky DFT-CLMS algorithm which essentially allows the weights to fade when the frequency in the signal also fades away. The result of the DFT-CLMS algorithm is plotted in Figure 41. It is observed that the Leaky DFT-CLMS algorithm clearly outperforms the initial DFT-CLMS approach.

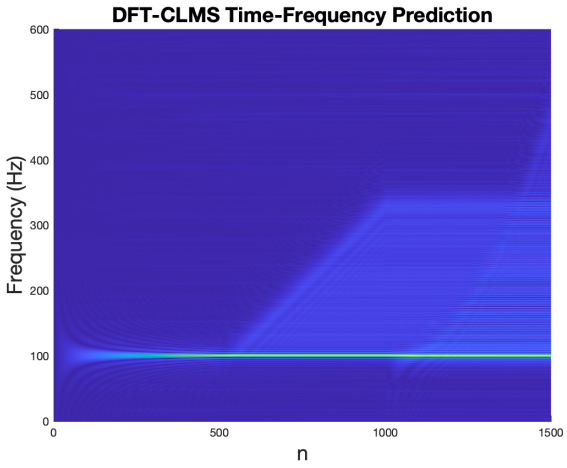

Figure 40: Evolution of the time-frequency estimate using DFT-CLMS.

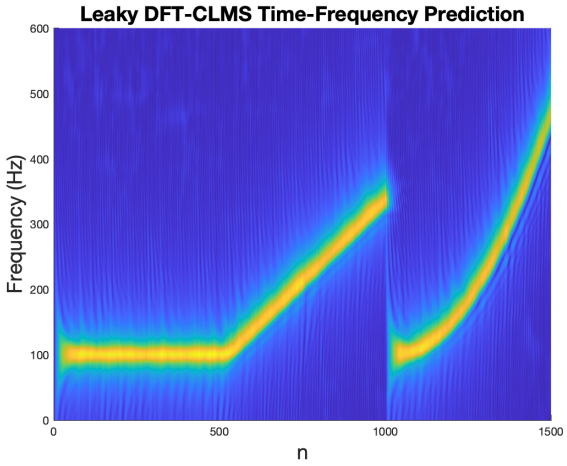

Figure 41: Evolution of the time-frequency estimate using Leaky DFT-CLMS.

d. We now apply the DFT-CLMS algorithm and the Leaky DFT-CLMS algorithm on the EGG signal to identify its spectrum. We compare this to the PSD estimate from 1.2b. It is observed both the DFT-CLMS and Leaky DFT-CLMS are able to identify the peaks at 13Hz, 26Hz and 50Hz. However, the two algorithms are not able to identify the peak at 39Hz as the frequency component is not as strong. The intensity of each of the peaks in the PSD estimate is well reflected in the time-frequency estimate (Note the intensity of the colour of the spectrum). Hence we conclude that both the time-frequency estimate (from DFT-CLMS and Leaky DFT-CLMS) and also the PSD estimate are able to represent the EEG signal accurately.

Note also that since the EEG signal is stationary and because of that the performance of the Leaky DFT-CLMS algorithm is similar to of the DFT-CLMS algorithm. This is due to the fact that the frequency of the signal does not change with n and that the "shadow" cast on later values of n does not affect the results.

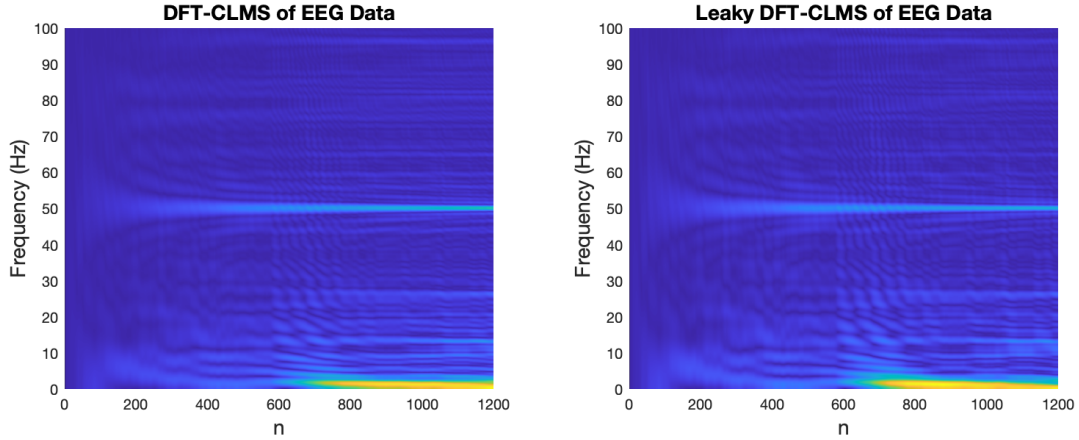

Figure 42: EEG Signal: Evolution of the time-frequency estimate using Leaky DFT-CLMS.

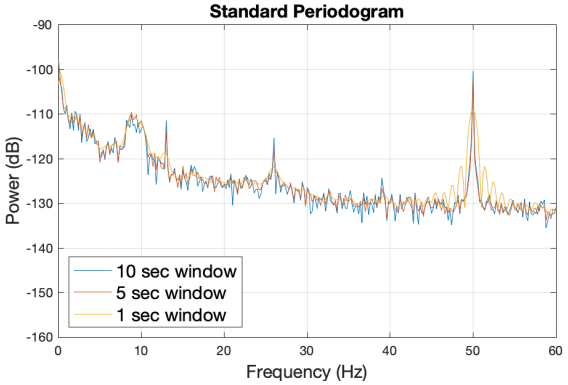

Figure 43: EEG Signal: PSD Estimate.

4 From LMS to Deep Learning

1. Figure 44 shows the zero-mean signal plotted against its AR(4) LMS algorithm prediction. The predicted signal started off with large errors. However, the error starts to diminish as the LMS algorithm converges to the right values. This can be seen by observing the plot: the predicted signal gets increasingly similar to the true signal as n increases.

The prediction gain, $R_p = 10log_{10}(\frac{\sigma_y^2}{\sigma_e^2}) = 5.1958$ dB, where σ_y^2 is the LMS output variance and σ_e^2 is the error variance. Since the prediction gain is positive and high, it indicates that the predicted signal power is much higher than the error power. This in turn implies that the prediction is accurate. The mean squared error, MSE of the prediction is 40.1027.

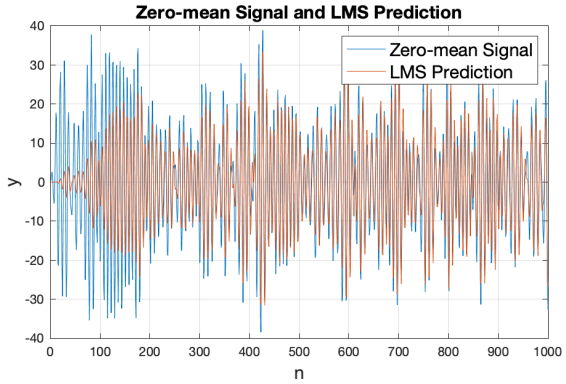

Figure 44: Zero-mean signal and one-step LMS prediction.

2. Since the generating process for the data is usually unknown and non-linear, a non-linear component is needed to make the LMS algorithm more expressive. Figure 45 shows the zero-mean signal plotted against its AR(4) LMS algorithm prediction with tanh activation function. It is clear that using tanh alone as the activation function is inadequate for the LMS algorithm to accurately predict the signal. This is because the range of a tanh function is from -1 to 1 but it is being used to predict a signal which has a range of roughly from -40 to 40.

From the plot, it is seen that the signal clips at the boundaries of the range of the tanh function: +1 and -1, in which is the best prediction the algorithm can make. The prediction gain, $R_p = -23.19$ dB is negative because the power of the error signal is higher than the power of the predicted signal. A comparison between the performance of this algorithm as compared to the previous algorithm is given below:

Algorithm	**R_p (dB)**	**MSE**
Original LMS applied onto zero-bias signal	5.1958	40.1027
Tanh LMS applied onto zero-bias signal	-23.1901	196.7410

Table 12: Prediction gain, R_p and Mean Square Error, MSE of different algorithms.

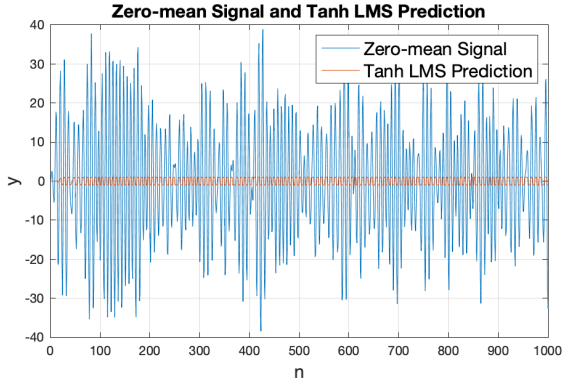

Figure 45: Zero-mean signal and one-step LMS prediction with tanh activation function.

3. To solve the problem from question 2, we introduce a linear scale, a to the tanh function. Since the zero-mean

signal has a range $\in [-40, 40]$, the obvious choice would be to set $a = 40$. However, since the original signal has a maximum amplitude of around $\in [-40, 50]$, we opted to set $a = 50$ so that if we were to modify the LMS algorithm to capture the bias, the algorithm and the scaling can be kept consistent. Figure 45 shows the zero-mean signal plotted against its AR(4) LMS algorithm prediction with a scaled tanh activation function. A comparison between the performance of this algorithm as compared to the previous algorithms is given below:

Algorithm	$\mathbf{R_p}$ **(dB)**	**MSE**
Original LMS applied onto zero-bias signal	5.1958	40.1027
Tanh LMS applied onto zero-bias signal	-23.1901	196.7410
Scaled Tanh LMS applied onto zero-bias signal	15.0642	6.5531

Table 13: Prediction gain, R_p and Mean Square Error, MSE of different algorithms.

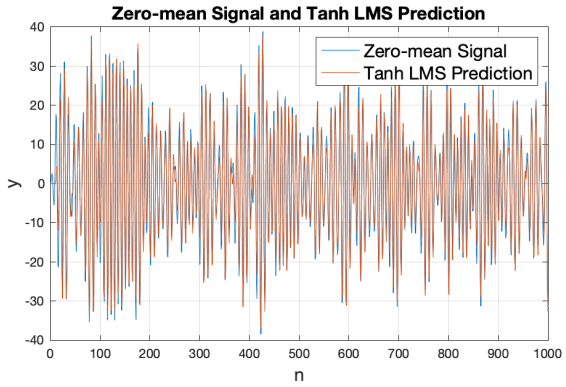

Figure 46: Zero-mean signal and one-step LMS prediction with scaled tanh activation function.

It is seen that the LMS with tanh activation function fits the true signal a lot better than the original LMS algorithm; It has a higher prediction gain and also a lower MSE. It is hence concluded that the scaled LMS algorithm with tanh activation function is a much better algorithm at predicting the signal.

4. The tanh LMS algorithm is used to predict the biased signal. The prediction and the true signal is plotted in Figure 47. It is observed that the tanh LMS algorithm fits the biased signal better than the original LMS algorithm. However, the algorithm under-performed as compared to fitting the zero-mean signal. This is reflected in the prediction gain, R_p of each algorithm and is shown in Table 14:

Algorithm	$\mathbf{R_p}$ **(dB)**	**MSE**
Original LMS applied onto zero-bias signal	5.1958	40.1027
Tanh LMS applied onto zero-bias signal	-23.1901	196.7410
Scaled Tanh LMS applied onto zero-bias signal	15.0642	6.5531
Scaled Tanh LMS applied onto biased signal	12.3128	13.4863

Table 14: Prediction gain, R_p and Mean Square Error, MSE of different algorithms.

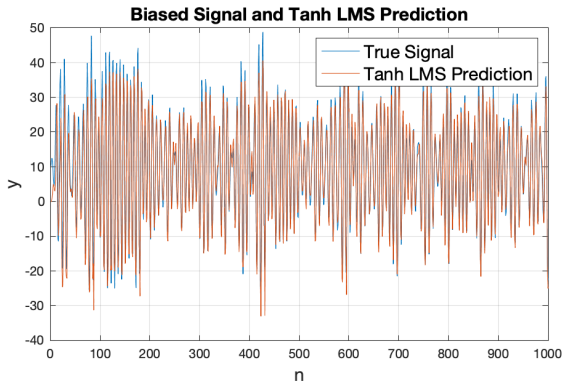

Figure 47: Zero-mean signal and one-step LMS prediction with scaled tanh activation function and bias prediction.

Since there the algorithm needs to learn an additional parameter (i.e. the bias), it is expected that its performance would degrade, hence giving a lower prediction gain and a higher MSE. Again, the algorithm starts off with higher prediction error. However, this error diminishes as the algorithm starts to converge to the correct values as n increases.

5. To address the slow convergence time, the algorithm is trained on the first 20 samples for 100 epochs. The pre-trained weights are then used to train the rest of the samples. The performance of this algorithm is given below:

Algorithm	$\mathbf{R_p}$ **(dB)**	**MSE**
Original LMS applied onto zero-bias signal	5.1958	40.1027
Tanh LMS applied onto zero-bias signal	-23.1901	196.7410
Scaled Tanh LMS applied onto zero-bias signal	15.0642	6.5531
Scaled Tanh LMS applied onto biased signal	12.3128	13.4863
Scaled Tanh LMS applied onto biased signal, with Pre-Trained Weights	16.2982	5.2202

Table 15: Prediction gain, R_p and Mean Square Error, MSE of different algorithms.

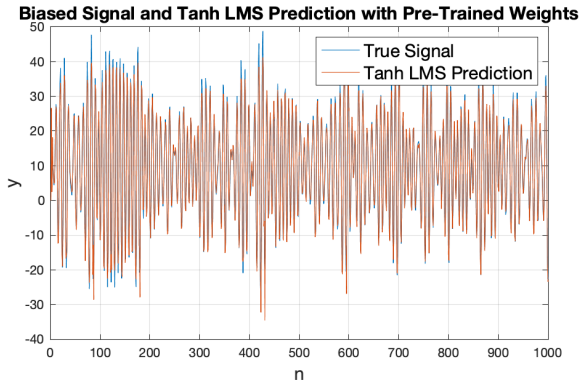

Figure 48: Zero-mean signal and one-step LMS prediction with scaled tanh activation function, bias prediction and pre-trained weights.

It is observed that pre-training the weights greatly improve the prediction gain and lowers the MSE. This is because the prediction error during the early convergence period is diminished. The effect can be observed in Figure 48 where the predicted signal converges to the true signal much faster as compared to its counterpart without pre-trained weights. However, this algorithm comes at a cost of computational complexity.

6. Deep neural networks use backpropagation to update the weights of the nodes so that the loss converges to zero. The weights are updated according to the error at the last node. The error is then backpropagated to the rest of the nodes, layer by layer, consequently updating every weight until it reaches the input layer. The process involves three steps:

1. Forward compute values at all nodes according to the defined weights. (Note that initially, all of the weights are initialised at random.)
2. Backward compute all δ.
3. Update all the weights: $w_{ij}^{(l)} \longleftarrow w_{ij}^{(l)} - \eta_t x_i^{(l-1)} \delta_j^{(l)}$

The definitions of the variables are given below:

1. Learning rate, η_t
2. Signal, $s_j^{(l)} = \sum_{i=0}^{d^{(l-1)}} w_{ij^{(l)}} x_i^{(l-1)}$
3. $x_j^{(l)} = \theta(s_j^{(l)})$ where θ is the non-linear function e.g. tanh, sigmoid.
4. Loss, ℓ; Note that in the last layer, $\ell = y - \hat{y}$
5. Gradient term, $\delta_i^{(l-1)} = \frac{\partial \ell_k}{\partial s_i^{(l-1)}} = \theta'(s_i^{l-1}) \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} w_{ij}^{(l)}$

The superscript and the subscript of the variables are explained in the Figure 49:

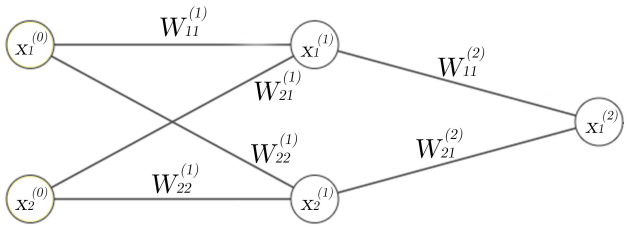

Figure 49: Superscript and subscript notations explained.

7. Figure 50 shows the individual X components as well as the noisy signal y. We would like to estimate y based on input X using deep neural network and we would compare its performance as compared to a using a simple dynamic perceptron.

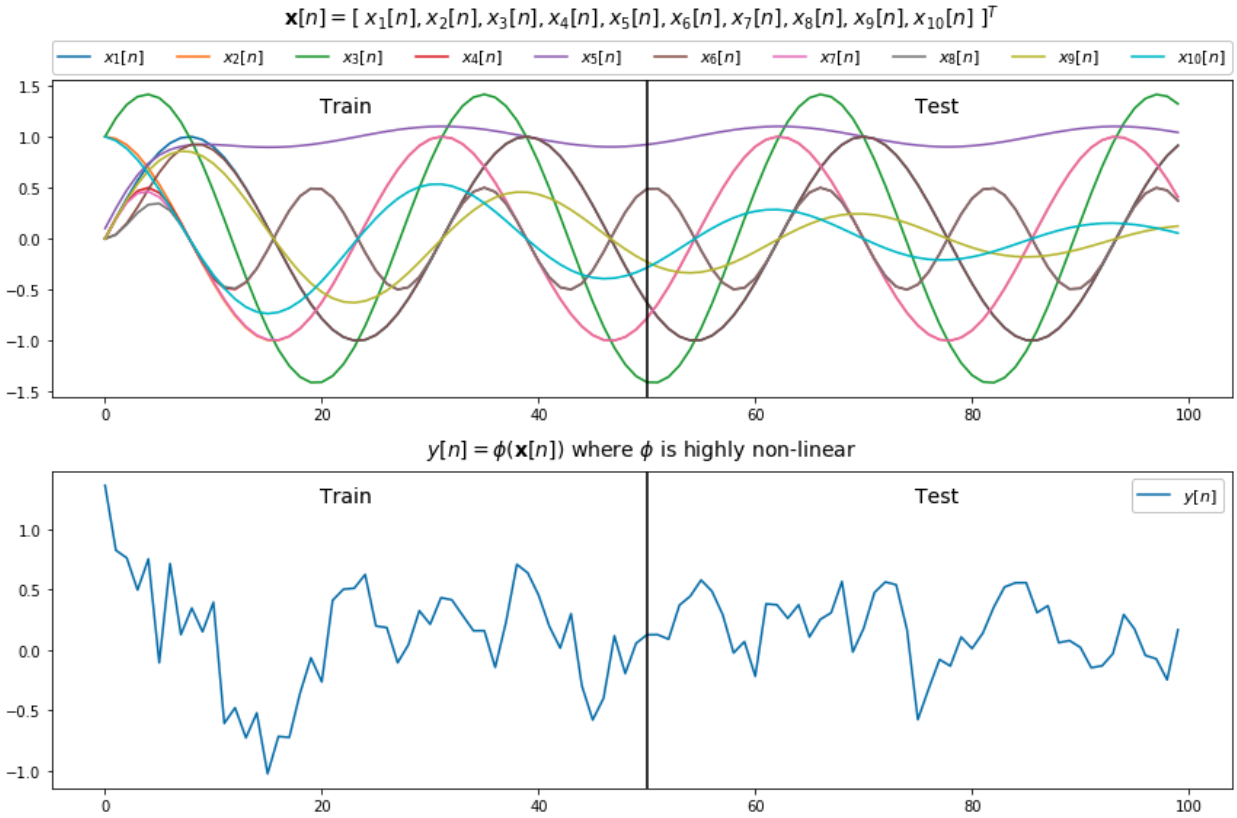

Figure 50: Components of signal x and signal y.

Figure 51 shows the predictions made using the different algorithms. We observe that the simple perceptron estimation gave a smooth result but fail to model the signal accurately. The deep neural network on the other hand has more sharp edges and is able to capture the trend of the signal but is also unable to accurately predict the signal. From Figure 52, we observe the test loss and train loss of each of the approaches. We observe that the deep network achieved a slightly lower (0.9) test loss as compared to both the simple perceptron approach (1.0). Besides that, we see that for all the approaches, the test loss is higher than of the train loss. This suggests that the models have overfit the data and some regularisation needs to be put in place so that the models better fit the data.

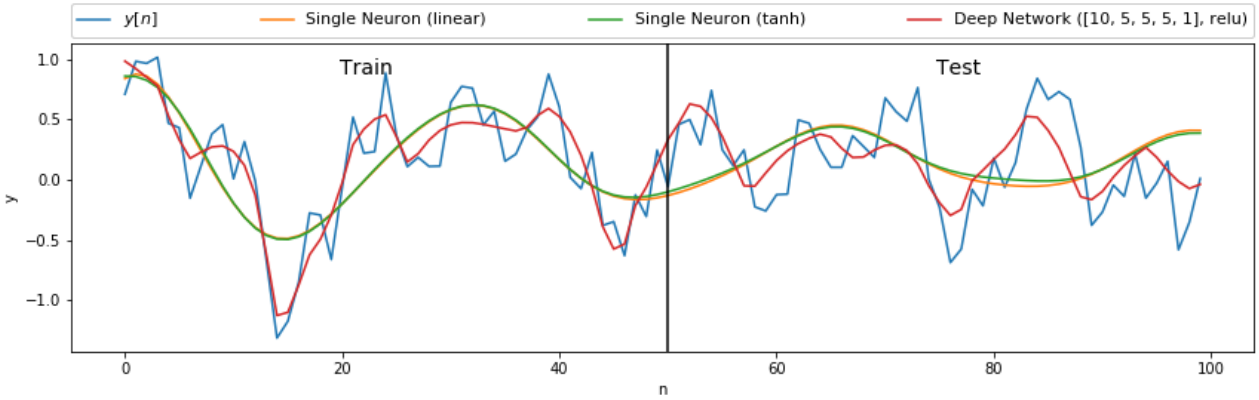

Figure 51: Predictions of signal y using simple perceptron and deep network. Noise power = 0.01.

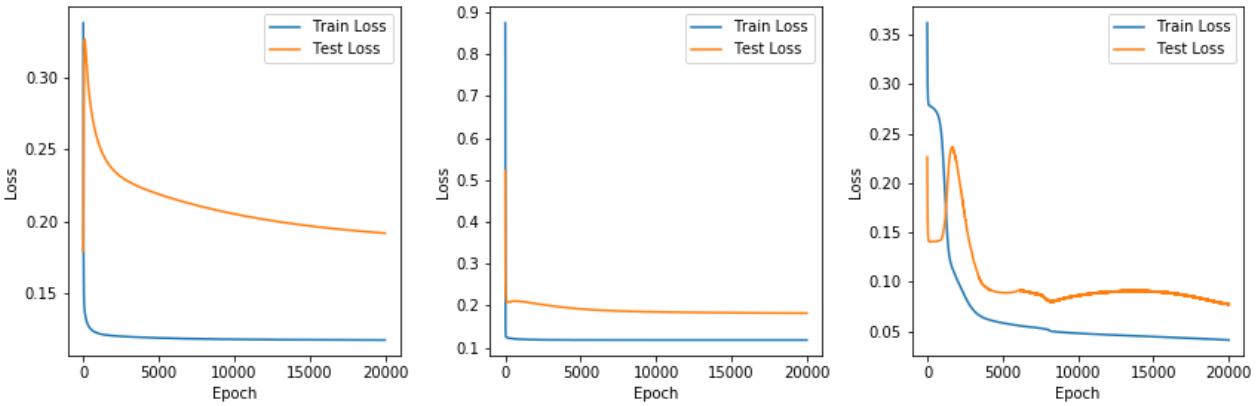

Figure 52: Loss vs epoch for different approaches. Left: single neuron (linear), middle: single neuron (tanh), right: deep network. Noise power = 0.01.

8. We now vary the noise power to analyse the effect of a noisier signal on the prediction result. Figures 53 and 54 show the prediction and losses of the different methods for noise power equal to 0.1. We see that the deep network overfits the data severely and that the simple perceptron even achieved a lower test loss compared to the deep network. Again, both the simple perceptron approaches (linear and tanh) achieve similar performance with a training and test error of around 0.1. It is also obersered that the training and test error of both the simple perceptron reaches to about the same level in which is desirable.

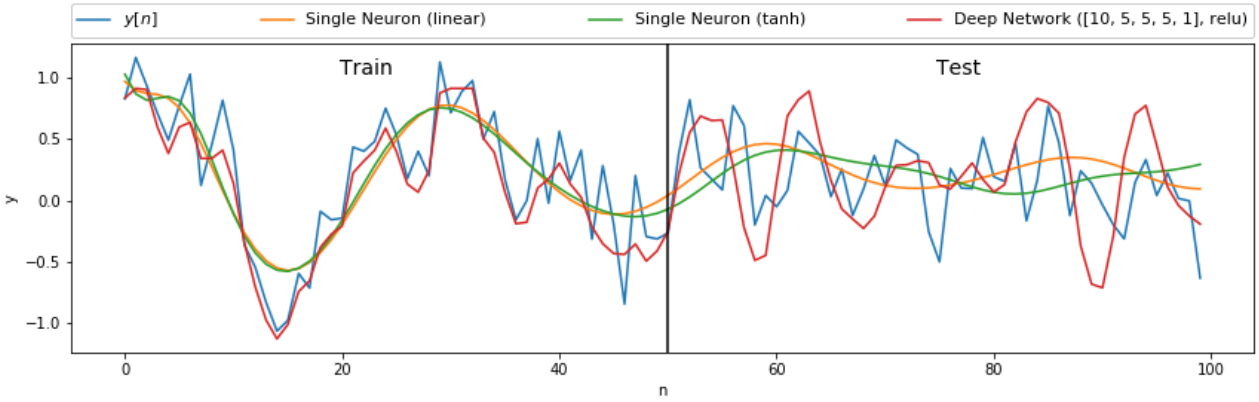

Figure 53: Predictions of signal y using simple perceptron and deep network. Noise power = 0.1.

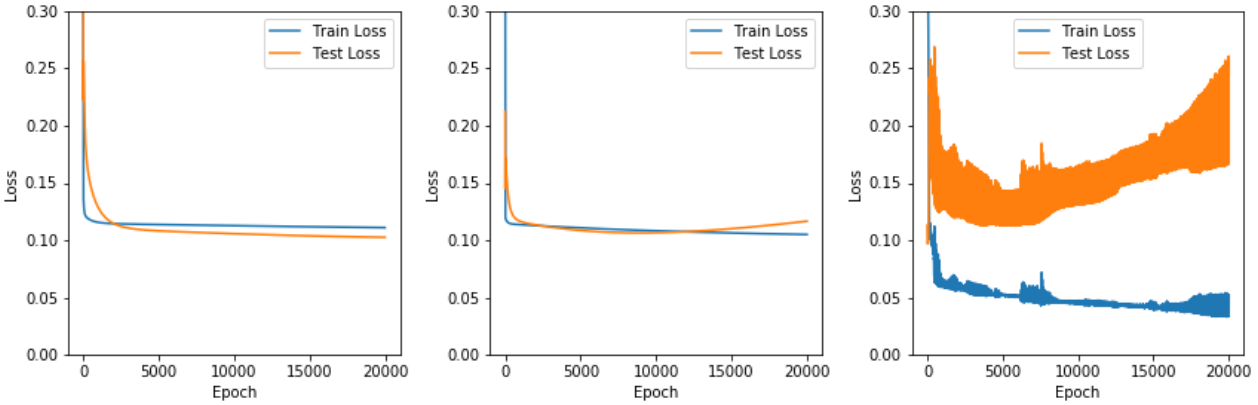

Figure 54: Loss vs epoch for different approaches. Left: single neuron (linear), middle: single neuron (tanh), right: deep network. Noise power = 0.1.

Figures 55 and 56 show the prediction and losses of the different methods for noise power equal to 0.001. We see that in this case the simple perceptrons overfit the data more severely than the deep network. Besides that the deep network achieved a much lower test loss than the simple perceptrons.

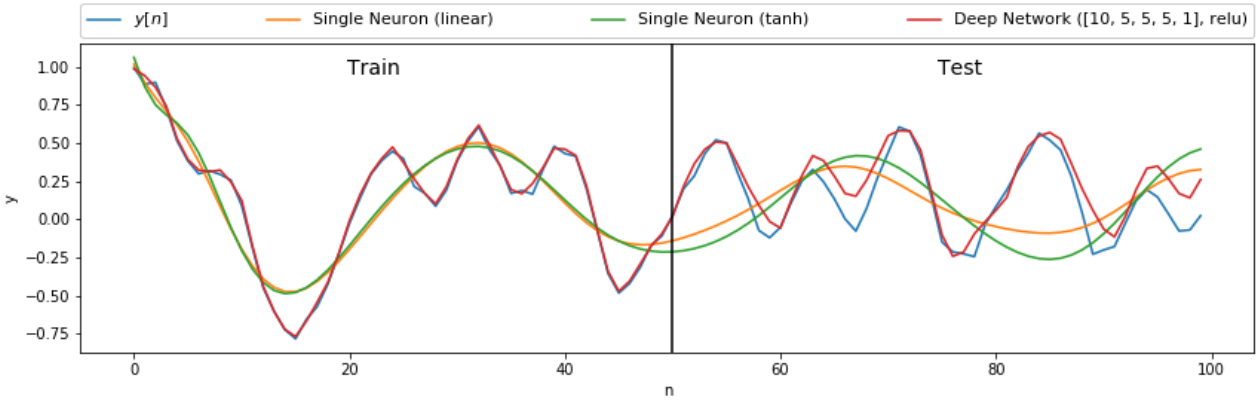

Figure 55: Predictions of signal y using simple perceptron and deep network. Noise power = 0.001.

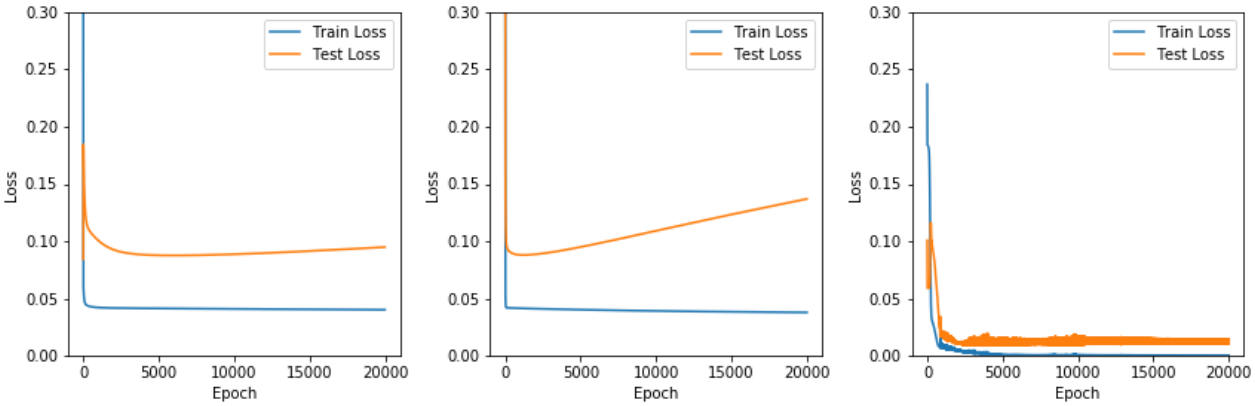

Figure 56: Loss vs epoch for different approaches. Left: single neuron (linear), middle: single neuron (tanh), right: deep network. Noise power = 0.001.

There are a couple of drawbacks of using deep learning for predictions:

1. The training of the deep networks are computationally expensive. This causes long training time and also incur high operational cost.
2. The high complexity of deep neural networks causes overfitting of the data. This causes a high validation error even if the training error is low which is undesirable. In order to counter this problem, a large amount of data is needed to match the complexity of the model which can be impractical.

Note that the rest of the report is done in Jupyter Notebook.